財 団 法 人 Ｊ Ｋ Ａ
平成 20 年度デジタルコンテンツの保護・活用に関する調査研究等補助事業

デジタルコンテンツ制作の
先端技術応用に関する調査研究

報　告　書

平成 21 年 3 月

財団法人 デジタルコンテンツ協会

KEIRIN

この事業は、競輪の補助金を受けて実施したものです。

URL : http://ringring-keirin.jp/

目　　次

図　　表

第 1 章　はじめに

1.1　調査研究の目的

映像コンテンツの高精細化に伴い、特にインタラクティブなコンテンツにおいては、より高度な表現を求めて、物理シミュレーションや AI 処理などの先端技術の活用が高まっている。そこで先端技術のインタラクティブ映像制作への応用を図るため、内外の関係情報を調査すると共に、新世代のデジタル制作技術基盤の方向性を追求し、核となる緊要な先端技術についての活用方策の提言を行い映像産業の振興に寄与することを目的とした調査を実施した。

これまでの調査により、我が国での映像産業の技術分野の牽引役は、ゲーム産業が担っていると考えられる。そこで、21 世紀のデジタルコンテンツを念頭におき、我が国のゲーム産業を中心としたインタラクティブなコンテンツに関連する研究の、映像制作への応用を図るため、ゲーム・プラットフォームやゲームを開発するための技術の変遷や教育について、内外の関連する動向・事例を調査し、その方策を検討する。

1.2　本年度の活動

具体的な活動状況として、デジタル映像制作・表示の技術動向及びその適用における諸問題に関する内外の情報を収集し、学識経験者及び産業界の専門家により構成される研究委員会を計 6 回開催し、調査研究を行った。

海外の技術動向調査としては、米国におけるゲーム産業技術動向の実態について、2009 年 3 月 23 日～27 日に米国サンフランシスコにて開催される世界最大のゲーム開発者向けカンファレンス GDC09 に参加し、調査を実施した。

また、「IGDA カリキュラムフレームワーク紹介および GDC09 報告会」と題し、GDC08 で公開された「IGDA カリキュラムフレームワーク 3.2 beta」の日本語版の紹介、および GDC09 におけるゲームの最新技術動向などについて、報告するセミナーを実施した。

1.2.1　第 1 回委員会

日　時：平成 20 年 7 月 14 日（月）　16：00～20：00

場　所：財団法人デジタルコンテンツ協会　会議室

主な議題

- SimCity, The Sims, Spore におけるゲーム AI 技術とプロシージャル
- AIIDE (Artificial Intelligence and Interactive Digital Entertainment Conference) 紹介
- 平成 20 年度事業内容検討

1.2.2　第 2 回委員会

日　時：平成 20 年 8 月 29 日（金）15：00～17：00

場　所：東京大学本郷キャンパス工学部 2 号館 9 階 92B 教室　工学部 2 号館

主な議題

- 本年度の調査項目について

1.2.3　第 3 回委員会

日　時：平成 20 年 10 月 1 日（水）16：00～18：00

場　所：食糧会館　特別会議室

主な議題

- 米国調査について
- 技術マップについて
- 物理エンジンの作り方
- セミナーについて

1.2.4　第 4 回委員会

日　時：平成 20 年 11 月 26 日（水）18：00～20：00

場　所：財団法人デジタルコンテンツ協会　会議室

主な議題

- 報告書の分類について
- 海外調査について

1.2.5　第 5 回委員会

日　時：平成 21 年 2 月 27 日（金）18：00～20：00

場　所：財団法人デジタルコンテンツ協会　会議室

主な議題

- 報告書の目次について
- 報告書の進捗確認
- セミナーについて

1.2.6　第 6 回委員会

日　時：平成 21 年 3 月 31 日（火）16：30～17：30

場　所：財団法人デジタルコンテンツ協会　会議室

主な議題

- 報告書について
- セミナーについて

1.2.7　セミナー「IGDA カリキュラムフレームワーク紹介および GDC09 報告会」

日　時：平成 21 年 3 月 31 日（土）　17：30～19：30

場　所：財団法人デジタルコンテンツ協会　会議室

モデレーター：馬場章委員長（東京大学大学院　教授）

パネリスト：新清士委員、三宅陽一郎委員、藤原正仁委員

1.3　推進体制

本研究委員会は、（財）デジタルコンテンツ協会（DCAj）における事業開発事業として、事業開発政策委員会のもと推進体制を組んでいる。

委員会メンバーは下記の通りで、東京大学大学院　情報学環　馬場章教授の下、推進する体制とした。

事務局は、DCAj 事業開発本部先導的事業推進部がこれを担当する。

表 1.3-01　平成 20 年度デジタルコンテンツ制作の先端技術応用に関する調査研究委員会名簿

	氏名	社名・所属	役職
委員長	馬場　章	東京大学大学院 情報学環	教授
委員	新　清士	IGDA 日本	代表
	津田　順平	株式会社コーエー ソフトウェア事業部　技術支援部	シニアエキスパート
	大塚　武	ソニー株式会社　B2B ソリューション事業本部 サービス＆ソリューション事業部 ネットワークアプリケーション事業開発本部 3 課	統括課長
	久保田　靖夫	大日本印刷(株) C&I 事業部	理事
	稲見　昌彦	慶應義塾大学大学院 メディアデザイン研究科	教授
	長谷川晶一	電気通信大学 電気通信学部知能機械工学科	准教授
	三上　浩司	東京工科大学 片柳研究所　クリエイティブラボ	プロデューサー
	藤原　正仁	東京大学大学院 情報学環	特任助教
	斎藤　直宏	株式会社バンダイナムコゲームス コンテンツ制作本部　制作統括ディビジョン	技術部 ゼネラルマネージャー
	小澤　賢侍	株式会社プレミアムエージェンシー	取締役／スタジオマネージャー
	三宅　陽一郎	株式会社フロム・ソフトウェア 技術部	
ｵﾌﾞｻﾞｰﾊﾞｰ	松井　悠	株式会社グルーブシンク	代表取締役
	村上　裕樹	株式会社グルーブシンク	
	細江　慎治	株式会社スーパースィープ	代表取締役
	佐藤　耕司		フリーライター
事務局	須藤　智明	財団法人デジタルコンテンツ協会 事業開発本部	課長代理

第 2 章　ゲーム・プラットフォームの変遷

2.1 ゲーム・プラットフォームの変遷

当記事における企業名・商品名・価格は発表当時のデータを使用している。

また、同ハードウェアのマイナーバージョンアップについては、同一機器と見なして記述している（ここでのマイナーバージョンアップは、ハードウェアのバージョンアップに伴ってソフトウェアの規格が変更されなかったものとする)。

2.1.1　1970 年代におけるゲームプラットフォーム

1970 年代初頭のゲーム・プラットフォームは、1 ハードにつき 1 タイトルの専用機が主であった。マグナボックス社の「オデッセイ」、アタリ社の「ポン」などが家庭用ゲーム機の始祖とされている。また、日本における最初の家庭用ゲーム機 1975 年には株式会社エポック社が「テレビテニス」を発売した。

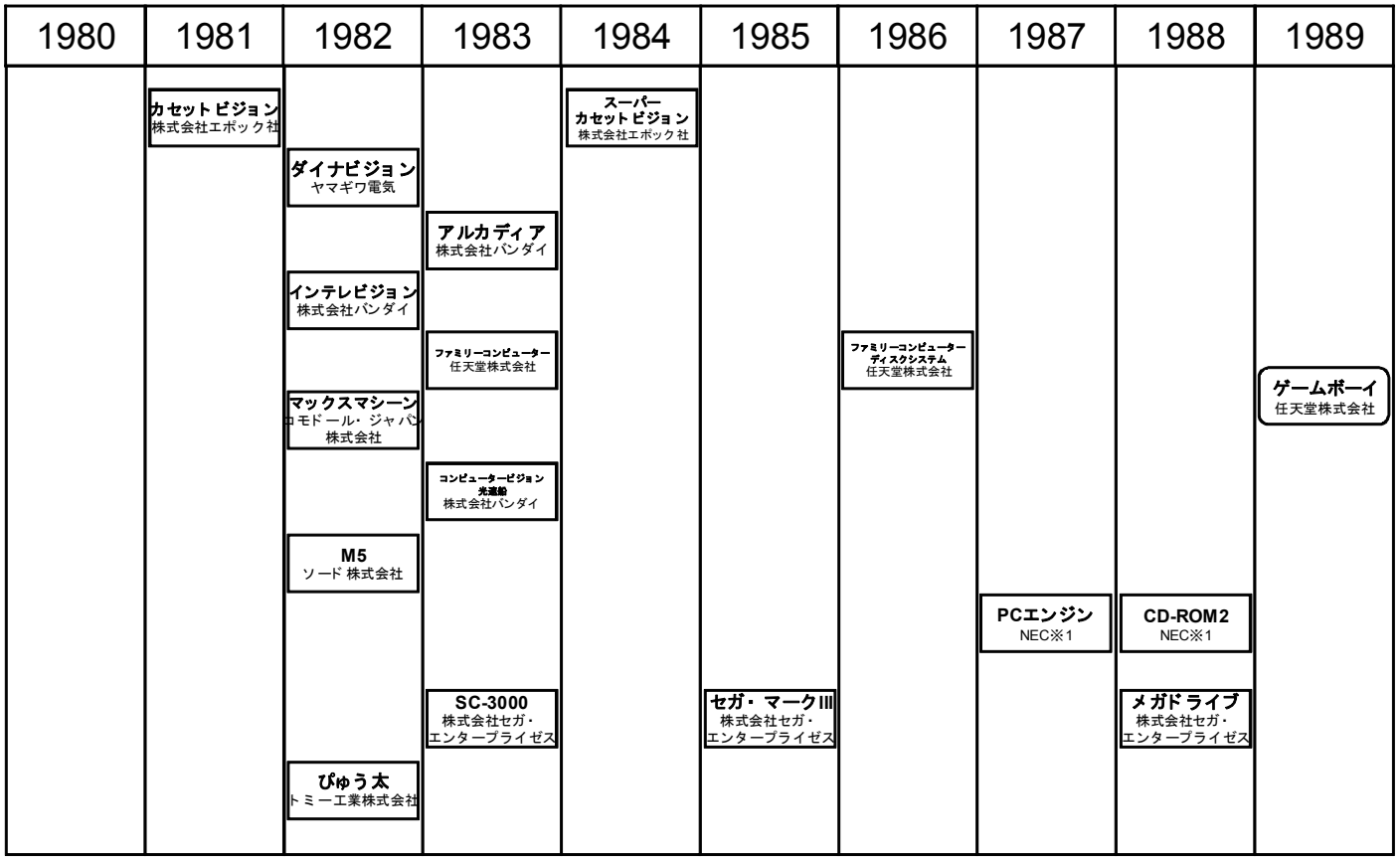
1977 年、世界初の ROM カートリッジ型家庭用ゲーム機「Atari 2600（ATARI 社)」がアメリカで発売。「Atari 2600」は、自社開発のソフトウェアだけではなく、サードパーティーメーカーの存在を確立させた。

2.1.2　1980 年代におけるゲーム・プラットフォーム

1980 年代初頭は国産の家庭用ゲーム機が多数リリースされた時期でもある。トミー工業（現：株式会社タカラトミー）の「ぴゅう太」、株式会社バンダイの「光速船」、株式会社セガ・エンタープライゼスの「SC-3000」、株式会社エポック社が「カセットビジョン」、「スーパーカセットビジョン」の販売を開始した。

1983 年、任天堂株式会社は「ファミリーコンピューター」を発売、1986 年には外部拡張デバイスとして「ファミリーコンピュータディスクシステム」を発売した。1985 年には株式会社セガ・エンタープライゼスが「セガ・マーク III」を発売。また、1987 年には日本電気ホームエレクトロニクス株式会社（NEC）が「PC エンジン」（翌 88 年に拡張デバイス CD-ROM2）を発売。1988 年に株式会社セガが「メガドライブ」を発売。また、1989 年には任天堂株式会社が日本初となる携帯ゲーム機「ゲームボーイ」をリリース。国内ゲームハードのタイプが著しく増加した時期である。

ゲームプラットフォームの変遷　1980年代



※1：日本電気ホームエレクトロニクス株式会社

図 2.1-01　1980 年代におけるゲーム・プラットフォームの変遷図

（日本国内での正規販売が行われたハードウェアを抜粋して掲載している）

2.1.3　1990 年代におけるゲーム・プラットフォーム

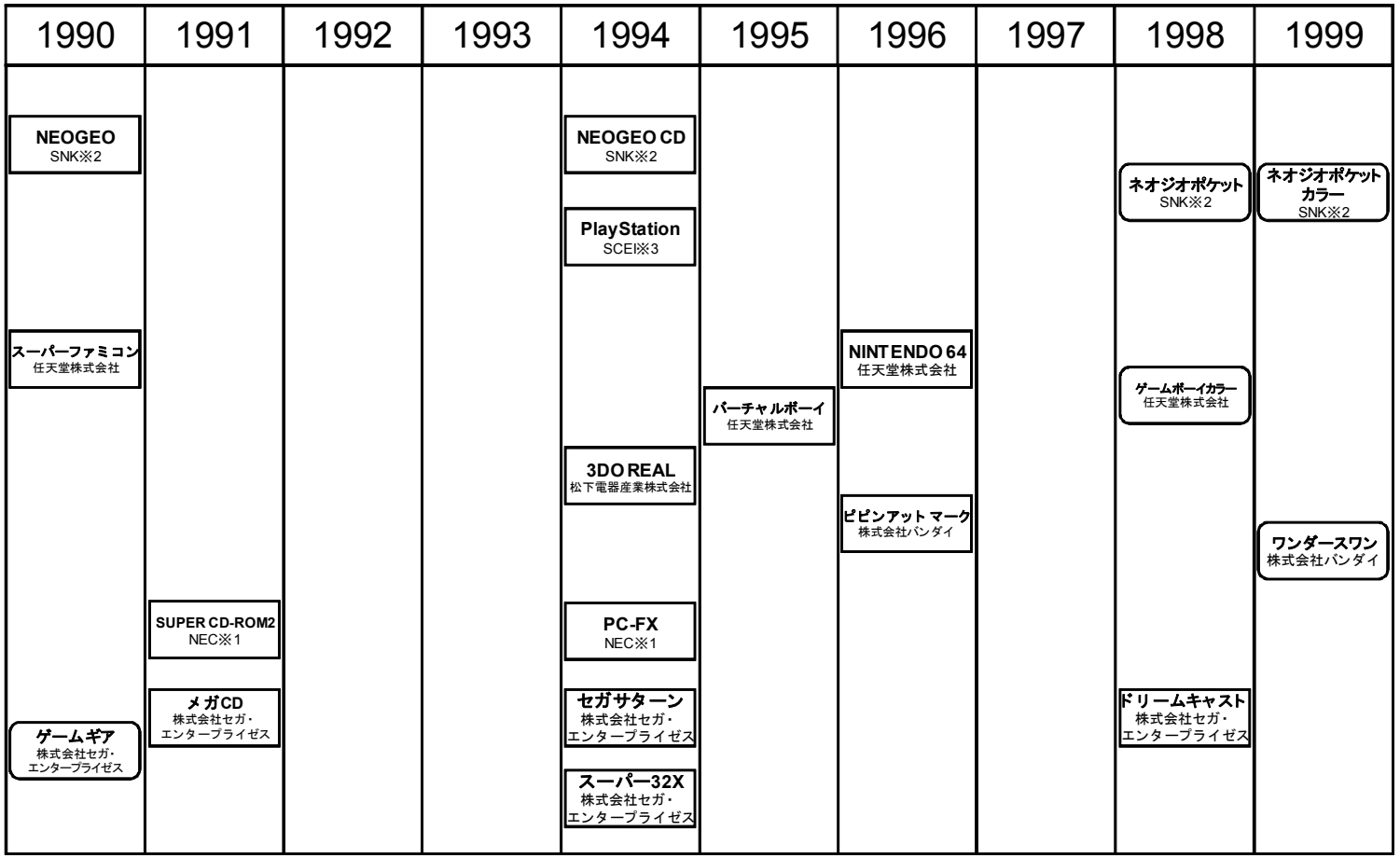
1990 年は数々のゲームハードがリリースされた年となる。4 月に株式会社エス・エヌ・ケイが「NEOGEO」を発売、10 月に株式会社セガ・エンタープライゼスが携帯ゲーム機「ゲームギア」を、11 月、任天堂株式会社は「スーパーファミコン」を発売した。翌 91 年 12 月、株式会社セガ・エンタープライゼスは「メガドライブ」の拡張ハードウェア「メガ CD」を、日本電気ホームエレクトロニクス株式会社が「PC エンジン」の拡張ハードウェア「SUPER CD ROM2」を相次いで投入。ゲームが ROM カセットから CD-ROM への移行を開始し始めた。

1994 年 2 月、松下電器産業株式会社が「3DO REAL」を発売したのを皮切りに、9 月に株式会社エス・エヌ・ケイが「NEOGEO CD」を、11 月に株式会社セガ・エンタープライゼスが「セガサターン」を、12 月に株式会社ソニー・コンピュータエンタテインメントが「PlayStation」を発売。また、同じく 12 月に株式会社セガ・エンタープライゼスが「スーパー32X」を、日本電気ホームエレクトロニクス株式会社が「PC-FX」を発売。日本国内における「次世代ハード戦争」の幕開けである。

1995 年 7 月、任天堂株式会社は単体ハードウェア「バーチャルボーイ」をリリース。1996 年 3 月には株式会社バンダイ・デジタル・エンタテインメントがアップル（コンピューター）社と共同開発を行った「ピピン@アットマーク」を発売。同年 6 月には任天堂株式会社が「NINTENDO 64」を発売した。

1998 年 10 月、任天堂株式会社は携帯ゲーム機「ゲームボーイ」の上位互換機、「ゲームボーイカラー」を、株式会社エス・エヌ・ケイも同じく携帯ゲーム機「ネオジオポケット」をリリースした。同年 11 月、株式会社セガ・エンタープライゼスは同社最後のハードウェアとなる「ドリームキャスト」を発売。また、1999 年 3 月、株式会社バンダイが携帯ゲーム機「ワンダースワン」を発売。1990 年代は、数々の国内ゲームメーカーがハード・ソフトともに充実した製品を発売、日本がゲーム大国としての座を不動のものにした 10 年間といえる。

ゲームプラットフォームの変遷　1990年代



※1：日本電気ホームエレクト ロニクス株式会社
※2：株式会社エス・ エヌ・ ケイ
※3：株式会社ソニー・ コンピュータ ーエンタ ーテインメント

図 2.1-02　1990 年代におけるゲーム・プラットフォームの変遷図

（日本国内での正規販売が行われたハードウェアを抜粋して掲載している）

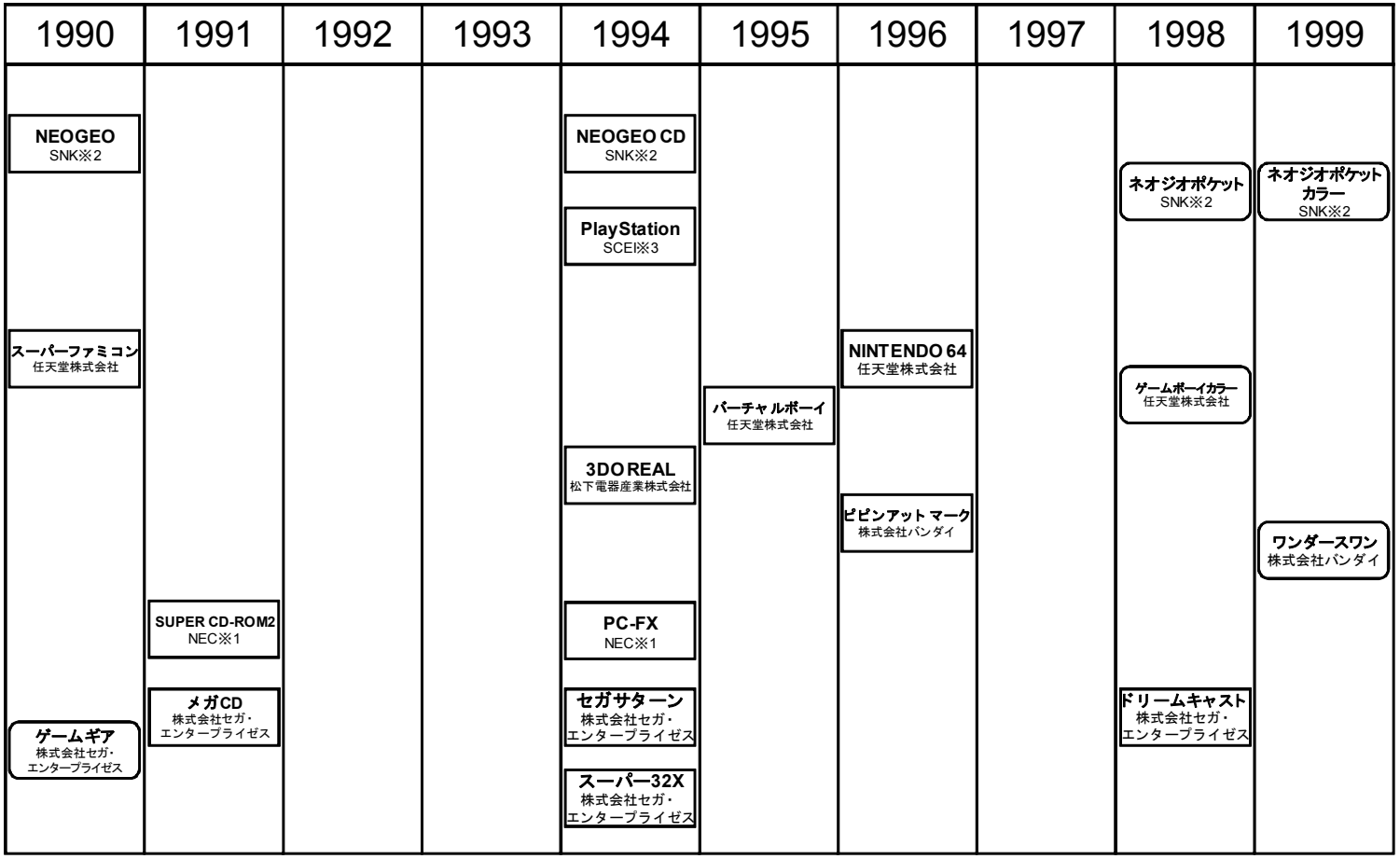
2.1.4　2000 年代におけるゲーム・プラットフォーム

2000 年 3 月、株式会社ソニー・コンピュータエンタテインメントは「PlayStation2」を発売。同年末、株式会社バンダイが携帯ゲーム機「ワンダースワンカラー」を、翌 2001 年 3 月任天堂株式会社が携帯ゲーム機「ゲームボーイアドバンス」を、9 月に「NINTENDO GAMECUBE」を発売した。2002 年 2 月、マイクロソフト株式会社が家庭用ゲーム事業に参入し「Xbox」をリリースした。

2004 年 12 月、任天堂株式会社が携帯ゲーム機「ニンテンドーDS」を、株式会社ソニー・コンピュータエンタテインメントが携帯ゲーム機「プレイステーション・ポータブル」を発売。現在、両ハードともにマイナーバージョンアップした上位機種をリリースし続けている。

2005 年 12 月、マイクロソフト株式会社が「Xbox 360」をリリース、翌 06 年 11 月には株式会社ソニー・コンピュータエンタテインメントが「PLAYSTATION 3」を、12 月には任天堂株式会社が「Wii」をリリース、再び「次世代ゲーム戦争」の幕が開くこととなった。

ゲームプラットフォームの変遷　1990年代



※1：日本電気ホームエレクトロニクス株式会社
※2：株式会社エス・エヌ・ケイ
※3：株式会社ソニー・コンピューターエンターテインメント

図 2.1-03　2000 年代におけるゲーム・プラットフォームの変遷図

（日本国内での正規販売が行われたハードウェアを抜粋して掲載している）

2.2 ゲーム・プラットフォームの概要

本項目では、年代別にゲームプラットフォームの概要を記載する。なお、本項目における企業名・商品名、価格は発表当時のものを使用している。
ハードウェアスペックは Winnie Forster : “The Encyclopedia of Game Machines: Consoles, Handhelds & Home Computers 1972-2005“, pp204 - 214, (2005)より引用した。

2.2.1 ファミリーコンピューター

任天堂株式会社が 1983 年に発売した「ファミリーコンピューター」は ROM カートリッジを使用する据え置きゲーム機。コントローラーは 4 方向十字キー、2（A・B）ボタン、スタート・セレクトボタン（コントローラー2 側にはスタート・セレクトボタンがなく、マイクユニットが用意されていた)。映像・音声出力は RF 出力端子で行う。

拡張ユニットとして、1986 年 2 月に書き換えが可能なクイックディスクを使用した「ファミリーコンピューター ディスクシステム」が、同年シャープ株式会社がファミリーコンピューターとディスクシステムを一体化させた「ツインファミコン」が発売された。
また、ファミリーコンピューター登場から 10 年後の 1993 年 11 月には、筐体デザインを一新、コンポジットケーブルでの映像出力に対応し、販売価格を 7,000 円に値下げした「AV 仕様ファミリーコンピューター」が発売された。なお、現在一部のファミリーコンピューター専用タイトルは、「Wii」のゲーム配信サービス「バーチャルコンソール」で販売されている。

■発売元　任天堂株式会社
■発売年月日　1983 年 7 月 15 日
■価格　14,800 円（発売当時）
■ソフトウェアの形式　ROM カートリッジ
■代表的なソフトウェア　「ドンキーコング」「スーパーマリオブラザーズ（任天堂株式会社)」「ファイナルファンタジー（株式会社スクウェア)」「ドラゴンクエスト（株式会社エニックス)」など
■周辺機器　「ファミリーコンピューター ディスクシステム」「光線銃」
■スペック
CPU：Ricoh（6502）　1.79MHz
解像度：256×240（52 色中 16 色同時使用可）

音源：3PSG、1 ノイズ

2.2.2 PC エンジン

日本電気ホームエレクトロニクス（NEC）株式会社が 1987 年に発売した「PC エンジン」は ROM カートリッジ「Hu カード」を使用する据え置きゲーム機。コントローラーは 8 方向十字キー、2（I・II）ボタン、RUN・セレクトボタン。映像・音声出力は RF 端子で行う。

翌 88 年、ゲーム機として初の CD-ROM メディアを採用した拡張ハードウェア「CD-ROM2(シーディーロムロム)」をリリース。

PC エンジンはハードウェアのバージョンアップが頻繁で、1989 年 11 月には廉価版の「PC エンジンシャトル」が、12 月には映像・音声出力を AV 出力に変更した「PC エンジンコアグラフィックス」、さらにグラフィックチップを 2 つ搭載した「PC エンジンスーパーグラフィックス」を発売。また、1990 年 12 月には据え置きゲーム機のソフトがプレイできるポータブルゲーム機「PC エンジン GT」が、1991 年 6 月には「PC エンジンコアグラフィックス II」が、9 月には CD-ROM2 ユニットと PC エンジンを一体化させた「PC エンジン Duo」がリリースされた。

また、PC エンジンシリーズには外部拡張ユニットが多数用意され、専用タイトルのリリースが行われていたことも特徴の一つである。

なお、現在一部の PC エンジン専用タイトルは、「Wii」のゲーム配信サービス「バーチャルコンソール」で販売されている。

■発売元　日本電気ホームエレクトロニクス株式会社
■発売年月日　1987 年 10 月 30 日
■価格　24,800 円（発売当時）
■ソフトウェアの形式　ROM カートリッジ
■代表的なソフトウェア：「R-TYPE」、CD-ROM2「天外魔境（株式会社ハドソン）」、「イース I・II（株式会社日本ファルコム）」
■スペック
CPU：Hu6280　7.6MHz
解像度：320×224～512×256（512 色中 32 色同時使用可）
音源：6Digital/PCM

2.2.3 メガドライブ

株式会社セガ・エンタープライゼスが 1985 年の「セガ・マーク III」に続いて 1988 年

10 月にリリースした「メガドライブ」は、ROM カートリッジを使用した据え置きゲーム機。コントローラーは 8 方向キー、3（A・B・C）ボタン、スタートボタンで構成されている。家庭用ゲーム機で初めて 16bitCPU を搭載し、自社タイトル以外のサードパーティータイトルも充実していた。映像・音声出力は AV ケーブルで行う。外部拡張機器として発売された「メガモデム」と電話回線を接続してゲームを配信する「ゲーム図書館」サービスなども行われていた。

1991 年 12 月には拡張ハードウェアで、ソフト供給に CD-ROM を採用した「メガ CD」を、1994 年 12 月にはメガドライブを 32bit マシンにアップグレードする「スーパー32X」を発売。また、同社の前世代機「セガ・マーク III」や「マスターシステム」のタイトルも別売り拡張ハードウェア「メガアダプタ」を使用することでプレイできる。なお、現在一部のメガドライブ専用タイトルは、「Wii」のゲーム配信サービス「バーチャルコンソール」で販売されている。

■発売元　株式会社セガ・エンタープライゼス
■発売年月日　1988 年 10 月 29 日
■価格　21,000 円（発売当時）
■ソフトウェアの形式　ROM カートリッジ
■代表的なソフトウェア　「ファンタシースター」シリーズ（セガ・エンタープライゼス）、「ぷよぷよ」（セガ・エンタープライゼス）、「ソニック・ザ・ヘッジホッグ」シリーズ（セガ・エンタープライゼス）
■スペック
CPU：メイン:MC 68000　7.67MHz、サブ:Z80A　3.58MHz
解像度：256×224 ～ 320×244（512 色中 64 色同時使用可）
音源：3PSG、1 ノイズ、6FM、1Digital/PCM

［1］　[セガハード大百科]メガドライブ
http://sega.jp/archive/segahard/md/

2.2.4　ゲームボーイ

任天堂株式会社が 1989 年に発売した「ゲームボーイ」は ROM カートリッジを使用した携帯ゲーム機。コントロール部は 8 方向キー、2（A・B）ボタン、スタート・セレクトボタンで構成されている。ゲーム画面はモノクロ 4 階調のモニターで表示される。

1994 年には、本体色を「赤」や「緑」、「スケルトン」などに変更したカラーバリエーションモデルの「ゲームボーイブロス」が発売、1996 年には、軽量化と価格の低下を実現した「ゲームボーイポケット」が発売された。また、1998 年には画面にバックライト

を搭載した「ゲームボーイライト」が発売されている。
　また、本体同士のデータ通信を行う「通信ケーブル」が発売され、本体を持ち寄ってプレイできるのも特徴のひとつ。

■発売元　任天堂株式会社
■発売年月日　1989 年 4 月 21 日
■価格　12,800 円（発売当時）
■ソフトウェアの形式　ROM カードリッジ
■代表的なソフトウェア　「テトリス」、「ポケットモンスター」シリーズ、「星のカービィ」（任天堂株式会社）
■周辺機器　「通信ケーブル」「4 人用アダプタ」
■スペック
　CPU：Sharp X80　4.2MHz
　解像度：160×144（4 階調モノクロ液晶）
　音源：4FM

［2］　ゲームボーイ
　　http://www.nintendo.co.jp/n02/dmg/hardware/gb/index.html

2.2.5　ネオジオ

　株式会社エス・エヌ・ケイが 1990 年に発売した「ネオジオ」は ROM カートリッジを使用した据え置きゲーム機。コントローラーは、8 方向ジョイスティックと 4（A・B・C・D）ボタン、スタート・セレクトボタンで構成されている。
　他の家庭用ゲーム機とコンセプトを異にする点としては、アーケードゲームとの密接な連動にある。ROM カートリッジ本体が 3 万円と、他ハードカートリッジに比べ高価であったが、これは業務用にリリースされていた NEOGEO、「Multi Video System」とプラットホームを共有し、ゲームセンターで実働中のゲームを比較的短期間のうちに自宅で遊べるようにしたためである。
　また、1994 年、メディアを ROM カートリッジから CD-ROM メディアに、同梱コントローラーをパッド式に変更した「ネオジオ CD」をリリース。その後、96 年には 2 倍速の CD ドライブを搭載、サイズの小型化を行った「ネオジオ CD-Z」を発売した。
　なお、現在一部の NEOGEO 専用タイトルは、「Wii」のゲーム配信サービス「バーチャルコンソール」で販売されている。

■発売元　株式会社エス・エヌ・ケイ

■発売年月日　1991 年 7 月 1 日
■価格　58,000 円（発売当時）
■ソフトウェアの形式　ROM カートリッジ
■代表的なソフトウェア　「餓狼伝説」シリーズ、「龍虎の拳」シリーズ、「King Of Fighters」シリーズ（いずれも株式会社エス・エヌ・ケイ）
■スペック
CPU：メイン:MC 68000　12MHz、サブ:Z80A　3.58MHz
解像度：320×224(65,356 色中 4,096 色使用可能)
音源：3PSG、1 ノイズ、4FM、6Digital/PCM、1ADPCM

2.2.6　ゲームギア

株式会社セガ・エンタープライゼスが 1990 年に発売した「ゲームギア」は ROM カートリッジを使用した携帯ゲーム機。コントロール部は 8 方向キーと 2（1・2）ボタン、スタートボタンで構成。ゲーム画面は 3．2 インチのカラー液晶で表示される。本体色が「赤」、「黄」、「スモーク」などのカラーバリエーションモデルや、懸賞企画として制作された限定カラーも存在する。1996 年には、本体名称を「キッズギア」に改称、ターゲットを若年層にシフトさせた。

また、外部拡張ユニットとして、「TV チューナーパック」や画面を約 1．5 倍のサイズに拡大する「ビッグウィンドー」などを発売していた。

■発売元　株式会社セガ・エンタープライゼス
■発売年月日　1990 年 10 月 6 日
■価格　19,800 円（発売当時）
■ソフトウェアの形式　ROM カードリッジ
■代表的なソフトウェア　「コラムス」、「ソニック・ザ・ヘッジホッグ」、「ぷよぷよ」（株式会社セガ・エンタープライゼス）
■スペック
CPU：Z80A　3.58MHz
ディスプレイ：バックライト付 3.2 インチカラー液晶
解像度：160×146（4,096 色中 32 色同時使用可）
音源：3PSG、1 ノイズ

［3］　[セガハード大百科]ゲームギア
http://sega.jp/archive/segahard/gg/

2.2.7　スーパーファミコン

任天堂株式会社が 1990 年に発売した「スーパーファミコン」は ROM カートリッジを使用した据え置きゲーム機。コントローラーは 8 方向キーと 6（A・B・X・Y、L・R）ボタン、スタート・セレクトボタンで構成されている。同社の「ファミリーコンピューター」との下位互換はなく、専用のカートリッジを使用している。

また、同社は「ゲームボーイ」カートリッジを使用できる拡張ユニット「スーパーゲームボーイ」シリーズを発売。1998 年には、筐体のデザインを変更し、スリム化を実現した「スーパーファミコンジュニア」を発売している。

なお、現在一部のスーパーファミコン専用タイトルは、「Wii」のゲーム配信サービス「バーチャルコンソール」で販売されている。

■発売元　任天堂株式会社
■発売年月日　1990 年 11 月 21 日
■価格　25,000 円（発売当時）※価格は未検証
■ソフトウェアの形式　ROM カードリッジ
■代表的なソフトウェア　「スーパーマリオワールド」、「スターフォックス」、「ファイヤーエムブレム」シリーズ（任天堂株式会社）、「ドラゴンクエスト」シリーズ（株式会社エニックス）、「ファイナルファンタジー」シリーズ（株式会社スクウェア）
■スペック
CPU：65C816　3.58MHz
解像度：256×224 ～ 512×448（32,789 色中 256 色使用可能）
音源：8ADPCM

［4］　スーパーファミコン
http://www.nintendo.co.jp/n02/shvc/index.html

2.2.8　3DO REAL

アメリカ、3DO 社によるライセンス提供を受けて松下電器産業株式会社が 1994 年に発売した「3DO REAL」は CD-ROM メディアを使用した据え置きゲーム機。コントローラーは 8 方向キーと 3（A・B・C）ボタン、停止・再生ボタンで構成されている。

なお、同年三洋電機株式会社も、3DO 社よりライセンス提供を受けた「3DO TRY」を発売している。周辺機器として、専用マウスや、ビデオ CD 再生用アダプタ、カラオケミキサー、ビデオプリンターなど、ゲームに限らない機器の展開を打ち出していた。

■発売元　松下電器産業株式会社

■発売年月日　1994 年 2 月
■価格　54,800 円（発売当時）
■ソフトウェアの形式　CD-ROM
■代表的なソフトウェア　「D の食卓」（株式会社ワープ）、「スーパーストリートファイターII X」（株式会社カプコン）
■スペック
CPU：ARM 60　12.5MHz
解像度：320×240 ～ 640×480（1670 万色）
音源：12PCM

2.2.9　セガサターン

株式会社セガ・エンタープライゼスが 1994 年に発売した「セガサターン」は CD-ROM を使用した据え置きゲーム機。コントローラーは 8 方向キーと 8（A・B・C・X・Y・Z、L・R）ボタン、スタートボタンで構成されている。同社の「メガドライブ」との下位互換はない。

なお、互換機として日本ビクターから「V サターン」が、日立からはビデオ CD とフォト CD 再生機能が標準装備された「ハイサターン」が発売された。また、日立からポータブルマルチメディアプレーヤーとして、別売りの専用モニターを接続できる「ゲーム＆カーナビ ハイサターン」も発売されている。

周辺機器として、ゲームのセーブデータなどを保存する「パワーメモリ」、インターネットやパソコン通信、通信対戦が可能なアナログモデム「セガサターンモデム」や、ビデオ CD 規格のメディアを再生できる「ムービーカード」やアーケード筐体を模した 2 人用アーケードスティック「バーチャスティックプロ」などが販売されていた。

セガサターン
■発売元　株式会社セガ・エンタープライゼス
■発売年月日　1994 年 11 月 22 日
■価格　44,800 円（発売当時）
■ソフトウェアの形式　CD-ROM
■代表的なソフトウェア　「バーチャファイター」シリーズ、「サクラ大戦」シリーズ、「セガラリーチャンピオンシップ」シリーズ（株式会社セガ・エンタープライゼス）
■スペック
CPU：メイン：two Hitachi SH2　28.6MHz×2
解像度：320×224 ～ 704×480（1677 万色）
音源：32PCM、FM

［5］ [セガハード大百科]セガサターン
http://sega.jp/archive/segahard/ss/
［6］ 日立ニュースリリース：ポータブルマルチメディアプレーヤー「ゲーム＆カーナビ ハイサターン」を発売
http://www.hitachi.co.jp/New/cnews/9512/1201.html

2.2.10 PlayStation

株式会社ソニー・コンピュータエンタテインメントが 1994 年に発売した「PlayStation」は CD-ROM を使用した据え置きゲーム機。コントローラーは、8 方向キーと 8（○・×・△・□、L1、L2、R1、R2）ボタン、スタート、セレクトボタンで構成されている（後に、アナログスティックを 2 本増設したモデルも発売されている）。

周辺機器として、ゲームのプレイデータを保存する「メモリーカード」、モノクロ液晶モニターを備えた携帯ゲーム機「ポケットステーション」、振動機能が追加されたアナログコントローラー「DUALSHOCK」などが発売された。また、プレイステーションの互換、上位互換の業務用システム基板も、株式会社カプコン、株式会社コナミ、株式会社タイトー、株式会社ナムコで採用されていた。

現在、一部の PlayStation タイトルは「PLAYSTATION3」、「プレイステーション・ポータブル」のダウンロード販売サービス「PlayStationStore」で手に入れることができる。

■発売元　株式会社ソニー・コンピュータエンタテインメント
■発売年月日　1994 年 12 月 3 日
■価格　39,800 円（発売当時）
■ソフトウェアの形式　CD-ROM
■代表的なソフトウェア　「リッジレーサー」シリーズ（株式会社ナムコ）、「ファイナルファンタジー」シリーズ（株式会社スクウェア）、「どこでもいっしょ」シリーズ（株式会社ソニー・コンピュータエンタテインメント）、「バイオハザード」シリーズ（株式会社カプコン）
■スペック
CPU：LSI/MIPS R3000A　33.8MHz
解像度：256×240～640×480（1670 万色）
音源：24ADPCM

2.2.11 NINTENDO64

任天堂株式会社が 1996 年に発売した「NINTENDO64」は ROM カートリッジを使用

した据え置きゲーム機。コントローラーは。8 方向キーと 3D スティック、9 ボタン（A・B、4 ボタンの C ユニット、本体下部の Z、L・R）、スタートボタンで構成されている。

周辺機器として、4MB のメモリを増設する「メモリ拡張パック」、振動機能を追加できる「振動パック」、ZIP ディスクを応用した書き換えゲームサービスユニット「64DD」などが販売されていた。

なお、現在一部の NINTENDO64 専用タイトルは、「Wii」のゲーム配信サービス「バーチャルコンソール」で販売されている。

■発売元　任天堂株式会社
■発売年月日　1996 年 6 月 23 日
■価格　25,000 円（発売当時）
■ソフトウェアの形式　ROM カートリッジ
■代表的なソフトウェア　「スーパーマリオ 64」、「マリオカート 64」、「ゼルダの伝説 時のオカリナ」、「ニンテンドウオールスター!大乱闘スマッシュブラザーズ」（任天堂株式会社）
■スペック
CPU：NEC, SGI MIPS R4300i　93.75MHz
解像度：320×240 ～ 640×480（32-Bit）
音源：16～24ADPCM

［7］　NINTENDO64
http://www.nintendo.co.jp/n01/n64/hardware/index.html

2.2.12　ドリームキャスト

株式会社セガ・エンタープライゼスが 1998 年に発売した「ドリームキャスト」は、新企画の高密度記憶媒体 GD-ROM を使用した据え置きゲーム機。コントローラーはアナログ方向キー、方向キー、4 ボタン（A・B・X・Y）と、2 つのアナログトリガー（L・R）とスタートボタンで構成されている。なお、本体に 33.6Kbps のモデムが標準装備されており、Web ページの閲覧やインターネットゲームのプレイが可能。「セガサターン」との下位互換は実装されていない。

ゲームデータは、コントローラーに差し込むモノクロ液晶つきの「ビジュアルメモリ」ユニットで管理を行う。アーケードの一部のゲームではビジュアルメモリユニットの接続口が用意されているものもあり、家庭用とアーケードのデータ連動も可能だった。
このほか、周辺機器として、「サンバ・DE・アミーゴ」対応の「マラカスコントローラ

ー」や、「電脳戦記バーチャロン・オラトリオタングラム」対応の「ツインスティック」、「ゲットバス」などに対応の「つりコントローラー」など、特定のタイトル操作に特化したコントローラーが複数発売されていた。

■発売元　株式会社セガ・エンタープライゼス
■発売年月日　1998 年 11 月 27 日
■価格　29,900 円（発売当時）※価格は未検証
■ソフトウェアの形式　GD-ROM
■代表的なソフトウェア「ファンタシースターオンライン」、「ぐるぐる温泉」、「シェンムー」、「シーマン」（株式会社セガ・エンタープライゼス）
■スペック
CPU：Hitach SH-4　200MHz
解像度：640×480（32-Bit）
音源：64ADPCM

［8］　ドリームキャスト
http://sega.jp/dc/hard/dc/

2.2.13　ワンダースワン

株式会社バンダイが 1999 年に発売した「ワンダースワン」は、ROM カートリッジを使用した携帯ゲーム機。本体はモノクロ 8 階調の液晶と、4 つの X ボタン（X-1~X-4）、4 つの Y（Y-1～Y-4）ボタン、A ボタン、B ボタン、スタートボタンで構成されている。ゲームによって本体を縦横のいずれかで保持するためにボタンレイアウトは独特なもの。また、サードパーティーから C 言語公式プログラムツールとして「Wonder Witch」が発売されている。

また、互換機として 2000 年にはカラー表示が可能な FSTN 液晶に変更し、ボタンレイアウトを若干変更した「ワンダースワンカラー」を発売、2002 年には TFT 液晶に変更した「スワンクリスタル」が発売された。

3 種のハードで、ゲーム同梱のオリジナルカラーも含め、カラーバリエーションが 30 種類以上存在しているのも特徴のひとつである。

■発売元　株式会社バンダイ
■発売年月日　1999 年 3 月 4 日
■価格　4,800 円（発売当時）
■ソフトウェアの形式　ROM カートリッジ

■代表的なソフトウェア　「GUNPEY」、「スーパーロボット大戦 COMPACT」（株式会社バンダイ）、「ファイナルファンタジー」シリーズ（株式会社スクウェア）

■スペック

CPU：V30MZ　3.078MHz

解像度：224×144（4,049 色中 241 色同時使用可）

音源：3PSG、1 ノイズ

［9］　ハードウェア（スワンクリスタル・ワンダースワンカラー紹介）
http://www.swan.channel.or.jp/swan/hardware/hardware.html

2.2.14　PlayStation2

株式会社ソニー・コンピュータエンタテインメントが 2000 年に発売した「PlayStation2」は、CD-ROM、DVD-ROM を使用した据え置きゲーム機。コントローラーは、方向キーと 2 本のアナログスティック、10（○・×・△・□、L1、L2、L3、R1、R2、R3）ボタン、スタート、セレクトボタンで構成されている。また、コントローラーのボタン（L3、R3、スタート、セレクトボタンをのぞく）はアナログ入力機能が搭載されている。本体のみで、DVD ディスクの再生が可能。

周辺機器として、「DVD リモートコントローラーキット」、オンラインゲームプレイ用のユニット「PlayStation BB Unit」などが発売されていた。

2009 年時点でもハードウェアの販売、ソフトウェアのリリースが行われており、現行のモデルは初期型と比べ、軽量・小型化されているほか、ネットワーク接続端子（イーサネット）を標準装備している。

PlayStation 2

■発売元　株式会社ソニー・コンピュータエンタテインメント

■発売年月日　2000 年 3 月 4 日

■価格　39,800 円（発売当時）

■ソフトウェアの形式　CD-ROM、DVD-ROM

■代表的なソフトウェア「ファイナルファンタジー XI」（株式会社スクウェア）、「メタルギアソリッド」シリーズ（株式会社コナミ）、「グランツーリスモ」シリーズ（株式会社ソニー・コンピュータエンタテインメント）

■スペック

CPU：Toshiba, Sony Emotion Engine　295MHz

解像度：640×480～1280×1024（64-bit）

音源：48ADPCM 可変

［10］ PlayStation2 情報
http://www.jp.playstation.com/ps2/

2.2.15 ゲームボーイアドバンス

任天堂株式会社が 2001 年に発売した「ゲームボーイアドバンス」は、ROM カートリッジを使用した携帯ゲーム機。本体は方向キーと 4（A・B・L・R）ボタン、スタート、セレクトボタン、2.9 インチ TFT カラー液晶で構成されている。

2003 年 2 月には、充電型になり、デザインを一新した「ゲームボーイアドバンス SP」を、2005 年 3 月には軽量化を施した「ゲームボーイミクロ」を発売した。

なお、ゲームボーイアドバンス、SP は、「ゲームボーイ」、「ゲームボーイカラー」との下位互換を実現しているが、「ゲームボーイミクロ」はゲームボーイアドバンス専用カートリッジのみの動作となる。

周辺機器として、無線による通信プレイを可能にする「ワイヤレスアダプタ」や、「ポケモンカード」、「どうぶつの森カード」を読み込む「カード e リーダー」などが販売された。

ゲームボーイアドバンス
■発売元　任天堂株式会社
■発売年月日　2001 年 3 月 21 日
■価格　9,800 円（発売当時）
■ソフトウェアの形式　ROM カートリッジ
■代表的なソフトウェア　「ポケットモンスター」シリーズ、「ファミコンミニ」シリーズ、「MOTHER3」（任天堂株式会社）、「逆転裁判」シリーズ（株式会社カプコン）
■スペック
CPU：ARM7 variant　16.7MHz
解像度：240×160（32,768 色中 512 色同時使用可）
音源：32PSG、4PCM

［11］ ゲームボーイアドバンス
http://www.nintendo.co.jp/n08/hardware/gba/index.html

2.2.16 NINTENDO GAMECUBE

任天堂株式会社が 2001 年に発売した「NINTENDO GAMECUBE」は光ディスクを使用した据え置きゲーム機。コントローラーは、コントロールスティック、方向キー、7

（A・B・X・Y・L トリガー・R トリガー・Z トリガー）ボタン、C スティック、スタートボタンで構成されている。

周辺機器として、無線コントローラーユニット「ウェーブバード」、「ゲームボーイ」、「ゲームボーイアドバンス」ソフトを接続する「ゲームボーイプレーヤー」、コントローラーポートにゲームボーイアドバンスを接続する「GBA ケーブル」、オンラインゲームをプレイするための「ブロードバンドアダプタ」・「モデムアダプタ」などが発売された。
また、同年 12 月、松下電器産業株式会社が DVD 再生機能を追加した互換機「Q」を発売した。

NINTENDO GAMECUBE（ニンテンドー ゲームキューブ）
■発売元　任天堂株式会社
■発売年月日　2001 年 9 月 14 日
■価格　オープン価格（発売当時）
■ソフトウェアの形式　DVD-ROM　※形式未検証
■代表的なソフトウェア　「ピクミン」、「大乱闘スマッシュブラザーズ DX」、「ゼルダの伝説」シリーズ（任天堂株式会社）、
■スペック
CPU：IBM Power PC “Gekko”　485MHz
解像度：640×480（24-bit）
音源：64ADPCM

［12］ ニンテンドーゲームキューブ
http://www.nintendo.co.jp/ngc/

2.2.17　Xbox

マイクロソフト株式会社が 2002 年に発売した「Xbox」は CD、DVD メディアを使用した据え置きゲーム機。コントローラーは左右のアナログスティック、方向キー、8（A・B・X・Y・黒・白・L トリガー・R トリガー）ボタン、Back・スタートボタンで構成されている。コントローラー上部には、音声チャット用「ボイスコミュニケーター」キットの差し込み口が用意されている。また、本体に 8GB の HDD を内蔵しているほか、イーサネット端子が付属しており、本体発売後に稼働したオンラインサービス「Xbox Live」でのオンラインゲームをプレイできる。

周辺機器として、「Xbox DVD ビデオ再生キット」や「Xbox Video Chat」などが発売された。

現在、一部の Xbox タイトルは、「Xbox 360」のオンラインダウンロードサービス

「Xbox クラシックス」でプレイ可能。

■発売元　マイクロソフト株式会社
■発売年月日　2002 年 2 月 22 日
■価格　34,800 円（発売当時）
■ソフトウェアの形式　DVD-ROM
■代表的なソフトウェア　「HALO」シリーズ（マイクロソフト株式会社）、「トム・クランシー」シリーズ（ユービーアイソフト株式会社）、「DEAD OR ALIVE」シリーズ（株式会社テクモ）
■スペック
CPU：Intel PentiumIII/ Celeron 733MHz
解像度：640×480 ～ 1,920×1,080（32-bit）
音源：256ADPCM

［13］ Xbox
http://www.xbox.com/ja-JP/hardware/xbox/inside/default.htm

2.2.18　ニンテンドーDS

任天堂株式会社が 2004 年に発売した「ニンテンドーDS」は ROM カートリッジを使用した携帯ゲーム機。本体は、方向キー、6 ボタン（A・B・X・Y・L・R）、スタート・セレクトボタン、3 インチ TFT 液晶ディスプレイ 2 枚で構成されている。

本機の大きな特徴として、2 枚のディスプレイと、下部ディスプレイのタッチセンサーの存在、ワイヤレス通信機能の標準装備が挙げられる。

2006 年には、軽量化、デザインの変更を行われた「ニンテンドーDS Lite」が、2008 年には、液晶ディスプレイを 3.25 インチに拡大、SD カードスロット、カメラを 2 カ所に搭載した「ニンテンドーDSi」を発売した。

また、ニンテンドーDS、ニンテンドーDS Lite は、ゲームボーイアドバンスとの下位互換が可能（ニンテンドーDSi では下位互換機能はない。また、ニンテンドーDSi のみで動作するソフトも販売が予定されている）。

■発売元　任天堂株式会社
■発売年月日　2004 年 12 月 2 日
■価格　15,000 円（発売当時）
■ソフトウェアの形式　ROM カートリッジ
■代表的なソフトウェア　「東北大学未来科学技術共同研究センター川島隆太教授監修

脳を鍛える大人の DS トレーニング」、「New スーパーマリオブラザーズ」、「ポケットモンスター ダイヤモンド・パール」（任天堂株式会社）、「オシャレ魔女♥ラブ and ベリーDS コレクションー」（株式会社セガ）

■スペック（ニンテンドーDS）

CPU： 67MHz、33MHz

解像度：256×192（262,144 色）

音源：16ADPCM

［14］ ニンテンドーDS
http://www.nintendo.co.jp/ds/series/ds/index.html

［15］ ニンテンドーDS Lite
http://www.nintendo.co.jp/ds/series/dslite/index.html

［16］ ニンテンドーDSi
http://www.nintendo.co.jp/ds/series/dsi/index.html

2.2.19 プレイステーション・ポータブル

株式会社ソニー・コンピュータエンタテインメントが 2004 年に発売した「プレイステーション・ポータブル」は、UMD ディスクを使用した携帯ゲーム機。本体は、4．3 インチの ASV 液晶を搭載、方向キー、アナログパッド、6（○・×・△・□・L・R）ボタン、スタート・セレクト・PS ボタンで構成されている。ゲームインターフェースは、PLAYSTATION3 同様の「XMB」を採用している。また、WiFi、USB 接続により PLAYSTATION3 との接続が可能。

2007 年には本体の軽量化を施した「PSP-2000」が、2008 年には、液晶を高精細なモデルに変更した「PSP-3000」が発売された。

また、メディアプレイヤーとしての使用も可能で、メモリースティック内に保存された、動画・音楽・画像データの閲覧に対応している。現在は WiFi 接続により、インターネットのブラウジングも可能。現在、オンラインゲームダウンロードサービス「PlayStationStore」では、初代 PlayStation タイトルや、独自タイトルが用意されている。

■発売元　株式会社ソニー・コンピュータエンタテインメント

■発売年月日　2004 年 12 月 12 日

■価格　オープン価格（発売当時）

■ソフトウェアの形式　UMD

■代表的なソフトウェア　「モンスターハンター ポータブル」シリーズ（株式会社カ

プコン）

■スペック（プレイステーション・ポータブル 10000）

CPU：1～333MHz

解像度：480×272（1,677 万色）

音源：flexible（MP3、MPEG4、ATRAC3）

［17］ プレイステーション・ポータブル情報

http://www.jp.playstation.com/psp/

2.2.20 Xbox 360

マイクロソフト株式会社が 2005 年に発売した「Xbox 360」は DVD-ROM を使用した据え置きゲーム機。コントローラーは方向キーと左右のアナログスティック、10（A・B・X・Y・LB・RB・LT・RT・左右スティック押し込み）、バック・スタートボタン、Xbox ガイドボタンで構成されている。本体には、HDD ユニットを接続でき、ゲームのダウンロードやインストールに使用する。また、本体背面にはイーサネット端子があり、LAN ケーブルを接続することでオンラインサービス「Xbox Live」を利用できる。

「Xbox Live」サービスでは、ゲームタイトル、追加コンテンツや、デモ、映像などのダウンロードの他、ボイスチャット（マイクユニットが必要）、ビデオチャット（カメラユニットが必要）サービスなどが用意されている。初代 Xbox との下位互換は、一部のタイトルのみ、パッチを使用することでプレイ可能。

■発売元　マイクロソフト株式会社

■発売年月日　2005 年 12 月 10 日

■価格　39,795 円（税込み発売当時）

■ソフトウェアの形式　DVD-ROM

■代表的なソフトウェア　「HALO3」、「ブルードラゴン」（マイクロソフト株式会社）、「アイドルマスター」シリーズ（株式会社バンダイナムコゲームス）

■スペック

CPU：IBM PowerPC カスタム 3.2GHz

解像度：640×480～1920×1080

音源：マルチチャンネルサラウンド出力・48kHz 16bit オーディオ対応

［18］ Xbox 360

http://www.xbox.com/ja-JP/hardware/x/xbox360console/

2.2.21　PLAYSTATION3

株式会社ソニー・コンピュータエンタテインメントが 2006 年に発売した「PLAYSTATION3」は、Blu-ray、DVD-ROM を使用した据え置きゲーム機。コントローラーは、方向キーと 2 本のアナログスティック、10（○・×・△・□、L1、L2、L3、R1、R2、R3）ボタン、スタート、セレクトボタンで構成されている。また、コントローラーには 6 軸の傾き検知機能が搭載されている。本体には、内蔵ハードディスクが 20GB、40GB、60GB の 3 モデルが存在し、初代 20GB、60GB モデルにのみ、「PlayStation2」との下位互換機能が搭載されている。2009 年 1 月時点で販売されているのは 40GB モデルのみ。本体のみで、Blu-ray、DVD ディスクの読み込みが可能。

本体には、無線 LAN、イーサネット端子が搭載されて（20GB モデルには無線 LAN は非搭載）おり、オンラインサービス「PlayStationNetwork」に接続可能。
オンラインストア「PlayStationStore」では、過去の PlayStation タイトルや、独自タイトル、ゲーム内アイテムなどのほか、映像のダウンロード販売を行っている。

■発売元　株式会社ソニー・コンピュータエンタテインメント
■発売年月日　2006 年 11 月 11 日
■価格　HDD20GB モデル 62,790 円（発売当時）、HDD60GB モデル オープンプライス（発売当時）
■ソフトウェアの形式　DVD-ROM、BD-ROM
■代表的なソフトウェア
■スペック
CPU：Cell Broadband Engine
解像度：640×480～1920×1080
音源：Dolby Digital 5.1ch,DTS 5.1ch,LPCM 7.1ch,AAC,etc

［19］ PLAYSTATION3
http://www.jp.playstation.com/hardware/ps3/cechl00.html

2.2.22　Wii

任天堂株式会社が 2006 年に発売した「Wii」は Wii 用 12cm ディスクを使用した据え置きゲーム機。コントローラーは無線型の Wii リモコンを使用し、方向キー、6（A・B、1・2、＋・－）ボタン、Home ボタン、電源ボタンで構成されているほか、傾き検知、加速度センサーを内蔵している。また、同社の「ゲームキューブ」との下位互換機能が実装されており、本体にゲームキューブコントローラーを接続してプレイが可能。

本体には、WiFi 機能が内蔵されており、同社の「ニンテンドーDS」とのワイヤレス通

信に対応するほか、Web ブラウジングソフト、ニュースソフトなどをダウンロードして利用できる「Wii チャンネル」のほかに Wii 独自の「Wii ウェア」や、過去のハードタイトルをダウンロードできる「バーチャルコンソール」サービスを提供している。

■発売元　株式会社任天堂
■発売年月日　2006 年 12 月 2 日
■価格　25,000 円（発売当時）
■ソフトウェアの形式　Wii 用 12cm ディスク
■代表的なソフトウェア
■スペック
CPU：Broadway
解像度：640×480～720×480

［20］ Wii
http://www.nintendo.co.jp/wii/index.html

発売日、価格など　公式ホームページ
http://www.nintendo.co.jp/wii/console/index.html
発売日、価格など
http://game.watch.impress.co.jp/docs/20060914/ninten.htm

第 3 章　ゲーム開発技術ロードマップ

3.1　はじめに

三宅　陽一郎
株式会社フロム・ソフトウェア

本章は、「ゲーム開発技術の主要分野における歴史と、これからの技術発展に対する予測」を編纂した内容となっている。各分野の執筆は、その分野に精通した研究者か、充実したキャリアを持つ開発者が担当している。

この報告書の目的は以下の 3 点である。

① これまで編纂されることのなかった、ゲーム開発技術の歴史を編纂し残して行くこと。
② ゲーム開発技術の歴史を認識し、開発技術の大きな流れを国内で作っていくこと。
③ ゲーム開発技術の歴史と現在から、これからの開発技術への展望を得ること。

である。

残念ながら、これまでゲーム業界は、開発技術を文書化し蓄積して来るという習慣を多く持たなかった。しかし、そういった蓄積の長い期間に渡る欠落は、現役の開発者自身が現在、どのような技術的系譜の上に立っているかを見失わせ、同時に、日本における技術の潮流を作って行くことを阻んで来た。しかし、これから、より多くの技術的基盤の上に、コンテンツ開発に集中しながらイノベーションを起こして行かなければならない時代に入り、「開発技術情報を発表できる場所を持ち」、「公開された情報を蓄積し」、「その全体を毎年、編纂してゲーム産業全体で共有できる形で文書化する」といった情報環境を整備が必要とされている。こういった情報環境の整備は、ゲーム産業全体、各ゲーム開発に、充実したコンテンツを作って行くより広い足場を提供し、業界全体で技術の潮流を築く基盤となり、引いては日本のゲーム産業の国際的競争力の強化に繋がるものである。

しかし、長い間、放置されて来たゲーム開発技術の歴史を、現在において再構築しようとする試みは、その開発資料の少なさと散逸が大きな問題となり立ち塞がる。そこで、今年度の最初の段階としては、「ゲーム開発技術全体の地図を作成する」ことを目標とし、各開発技術における大きなカテゴリーと、その時間的発展の流れを編纂することで、本格的に開発技術を編纂して行くための大きな土台を作ることを方針にした。そういった意味で、本報告書は、大きな青写真であり、荒削りの土台であり、そして同時に、ゲーム開発

技術の歴史の大きなうねりを捉えるために絶好の文書ともなっている。そして、これから、この大きな地図の各部分を一つ一つ詳細に追求して行く作業が待っている。そのためには、より多くの開発者の協力を必要とするだろう。

また、そういった技術のロードマップを作成することの最大の目的は、未来に対する技術の展開の確かな予想を得ることであり、過去、現在の情報が精緻に編纂されるほど、未来に対する予測が可能となり正確になる。また、同時に、開発者は技術の未来を拓く当人でもあり、ロードマップの作成は、開発者に次に必要な一手を自ら実行する機会を与えるものでもある。これは、日本の開発者が世界で技術的イニシアティブを取っていくことで必要なことである。

同時にこの編纂作業は現在における情報蓄積のあり方にも示唆を投げかけるものである。この数年で、様々なゲーム開発に高い意識を持つデジタルゲーム関連の機関によって、ゲーム開発技術の可能な範囲の共有化、オープン化を促進するカンファレンス、セミナー、講演、ワークショップなどが行われて来た。そういった発表の際に問題となるのは、

① 開発者自身が自身の技術の重要性を判断するのが難しいこと
② 企業が開発技術をどの程度オープンにするか指標がないこと

の二点である。前者は、よく見られるケースで、日本では過少に自身の仕事を評価する事例が多く、なかなか公の場で発表する機会を得る確信が得られない。それは、指標となる技術の情報がないからで、自身の仕事の価値を正確に判定するためには、これまでのゲーム開発技術の歴史が必要なのである。②も基本的に同様である。これまで、ゲーム産業全体が共に形成してお互いに恩恵を受けて来た技術的潮流がないために、技術を公開するということそのものに対する判断を難しくしているのである。

このように、技術資料の欠如と技術的潮流の欠如は、ゲーム産業の慣性となって、悪循環を産み出している。本報告書のゲーム開発技術編纂の仕事は、この二つの問題を改善しようとする試みである。そして、継続して、ゲーム開発技術に目を向け評価し、整理をくり返すことで、雑然とした混沌から本来のゲーム開発技術の道筋を見出し、活性化した技術的潮流を導くことは、技術分野のみならず、ゲーム産業全体の活性化につながるはずである。

表 3.1-01　ゲーム開発技術ロードマップ執筆者一覧

章	開発技術分野	執筆者（敬称略）
3.2.1	ゲームデザイン	井上明人（国際大学 GLOCOM 研究員）
3.2.2	3D CG 製作	川島 基展（東京工科大学 片柳研究所　クリエイティブ・ラボ）
3.2.3	サウンド製作	細江　慎治（株式会社スーパースィープ）
3.2.4	アニメーション製作	金久保　哲也（株式会社バンダイナムコゲームス）
3.2.5	プログラミング AI	三宅　陽一郎（株式会社フロム・ソフトウェア）
3.2.6	プログラミング グラフィックス描画	宮澤 篤・大久保 明（株式会社バンダイナムコゲームス）
3.2.7	プログラミング 物理・衝突判定	長谷川　晶一（電気通信大学）
3.2.8	タスクシステム技術	田村　祐樹（株式会社ネバーランドカンパニー） 大野　功二（オープランニング） 黒須一雄（株式会社モバイル&ゲームスタジオ） ※第 4 章に黒須氏へのインタビューを掲載
3.2.9	プログラミング スクリプト	小久保　啓三（専門学校 HAL 東京）
3.2.10	ネットワーク通信	佐藤カフジ（フリーライター）
3.2.11	開発方法論	長久　勝（ハイパーコンテンツ(株)/早稲田大学メディアネットワークセンタ）

3.2　カテゴリー別の技術ロードマップ

3.2.1　技術としてのゲームデザイン

井上明人
国際大学 GLOCOM 研究員

本稿は、技術としてのゲームデザインの簡略な紹介することを目的としている。ゲームデザインという概念は、それ自体が未だ明確な範囲を持たない概念であるが、ゲーム開発の現場の業務として言えば、主にディレクターやプランナーとされる職種の人々が独自に担うと考えられている業務が「ゲームデザイン」と呼ばれる領域となる。

だが、前提としてゲームデザインとは狭義の意味で「技術」ではない。そのため、技術史としての記述を開始する前に、どこまでが「技術」の領域でありうるのか、そして、そうではないのか。この概念上の問題を確認しておきたい。

(1)　技術としてのゲームデザインの領域の定義：何を扱い、何を扱わないか

「技術」という語についての定義は、ギリシャ語の語源から、ハイデガーの技術論に至るまで、多種多様なものが提出されており、厳密な定義を試みようと思うと、きわめて困難な類の概念の一種である。ここでは、哲学的に厳密な議論は行わないが、本稿に必要な範囲で、この概念の指し示す範囲について述べたい。

(a)　手段と目的

人間の行為を「目的と手段」という形で分けたとき、手段にあたるものが技術である、としばしば位置付けられる。前提として「目的」が与えられているため、目的の実現度合いや、その効率性に応じて「技術」はその良し悪しを評価されるという性質を持つ。定義上、評価基準を内包したものである。

しかし「ゲームデザイン」とは、そもそもゲームをどのようなものとして設計するのか、という目的設定に関わることが多い。「何を実現したいか」ということを考えるための手立ては、＜手段＞という意味での技術ではなくなる。目的は、評価基準を用意する。評価基準そのものを、評価することは批評としては可能だが、書き手の主体性によって書かれないタイプのテキストにおいては原理的に困難である。そのため、「ゲームデザイン」を所与の目的を実現するための技術発展の歴史として描くことは、そもそも限界がある。

だが、より厳密に言うならば、手段と目的という区分は、その区分自体の自明性がはっきりと区分けできない。手段は目的に従うばかりでなく、目的は手段に影響される。たと

えば、「面白いゲーム」の面白さについての感覚のバリエーションは、一律に規定することは難しくとも、実質的にはだいたいの範囲が存在している。成功したゲームデザインが＜何であるか＞という評価基準について述べることは批評的な行為とならざるを得ないが、成功したと＜考えてられているであろう＞ゲームデザインについて述べることは、可能である。優れていると考えられているゲームのエッセンスを模倣するための、プログラムの技術にもグラフィックの技術にも還元されないような残余、これをゲームデザインの技術と呼ぶならば、これは「技術」として位置付けることが可能な範囲と捉えることができる。

一方で、既存の発想とは全く違ったことを実現しようということを目指すようなもの――すなわち、目的設定に関わるような議論――も「ゲームデザイン」に関わる範囲として扱われる。だが、これは技術として記述することは不可能な部類に入る。評価基準そのものに関する議論は、技術論の範疇ではない。よって、本稿では、こうした水準での「ゲームデザイン」に関わる問題は、過去の傾向の変遷として記述するにとどめ、発展史としては扱わない。

(b) 全体と部分

手段とは、全体として達成されることの部分であると位置付けることができる。

だが「デザイン」という概念には、部分について考えることではなく全体の関係性そのものの配置こそがデザインである、という見方がある。ある特定の要素を、削るか、削らないかということの決定は、全体とのバランスの中ではじめて意味を持つ。最終的に成立するゲームの全体性は、様々なトレードオフの中のうちの一部である。それゆえ、ゲームデザインについての技術とは、ゲーム全体の中に占める要素ごとの改良に関する問題ではない。トレードオフの構造そのものを理解する、ということがゲームデザインを考える上で重要な位置を占める。

これは、先ほどの目的と手段という区分けに対応させて言えば、目的の選択そのものの良し悪しについて語ることではなく、目的の選択をする上でどのような前提を理解すべきか、についてだと位置付けることができる。

本稿では、1. ゲームデザインの発想の変遷（目的）2. ゲームデザインの技術と考えられている領域（手段）　3. ゲームデザインにおけるトレードオフ（部分と全体性）を順に扱う。

(2) ゲームデザインの発想の変遷

上述のように、ゲームデザインは、その概念の内に明確な評価基準を持たないため、単に発展史として記述することはできない。だが、その前提を置いた上で、あえてゲームデ

ザインを「歴史」として記述するヒントとして、ゲームデザインの類型論を手がかりにしたい。

ゲームデザインの類型論としては、分類数の大きなものから数えて、12 分類、8 分類、4 分類など、ゲームの面白さについて様々な説明モデルを示した類型がある。主要な例については、表 3.2-01 に示した。

表 3.2-01　ゲームデザインの類型論

一元論的説明	学習、適応モデル（ラフ・コスター、井上明人など） フロー体験（チクセントミハイ） 非日常、インタラクション、コミュニケーション、身体、能動性
二分類（二元論）	ルドゥス/パイディア（カイヨワ） ゲーム/遊び（ミードなど） ルール/フィクション（Jesper Juul など） 能動/受動、ルール/目的
四分類	アゴン（競争）、アレア（運）、ミミクリ(模擬)、イリンクス（眩暈）（カイヨワ） Socilizer/Killer/Explorer/Achiver(リチャード・バートル)
五分類	視覚 / 聴覚 / 触覚 / 味覚 / 嗅覚
八分類	驚き/ファンタジー/物語/挑戦/コミュニケーション/発見/表現/余暇 （GDC2008、Game Design Workshop handout より）
一二分類	Timmy[Power,Social,Diversity,Adrenalin]、 Johnny[Combo,Offbeat,Deck,Uber]、Spike[Innovator、Tuner、 Analyst、Nut&Bolt] (Mark Rosewater)

このうち、社会心理学者のチクセントミハイによるフロー体験や、フランスの哲学者／社会学者のロジェ・カイヨワによる分類であるアゴン（競争）、アレア（運）、ミミクリ（模擬）、イリンクス（眩暈）などといった分類が日本では知名度が高い。一方、英語圏では、ルドゥス（きっちりとした遊び）／パイディア（枠の緩い遊び）といった枠組みが参照されることが多いなど、同じ研究の中でも参照される領域に違いが見られることが多々ある。

ただ、こうした種類の議論を見るときに注意するべきことの一つとして、これらが「面白さ」の類型論なのか、それとも「ゲーム」や「遊び」という現象の類型論であるのかは区別しておく必要がある。一元論的説明や、二元論（二分類）の説明はどちらかと言えば、面白さについての類型論であるというよりは、「ゲーム」や「遊び」という人間行動を説明し、考えるための学問的なモデルをベースとしているため、議論のモデル自体となるテ

キストを読み、ある程度まで理解しなければ、それほど深みがない。一方で、四分類や、八分類は発想法のヒントという水準で適用可能な、「面白さ」の類型論である。

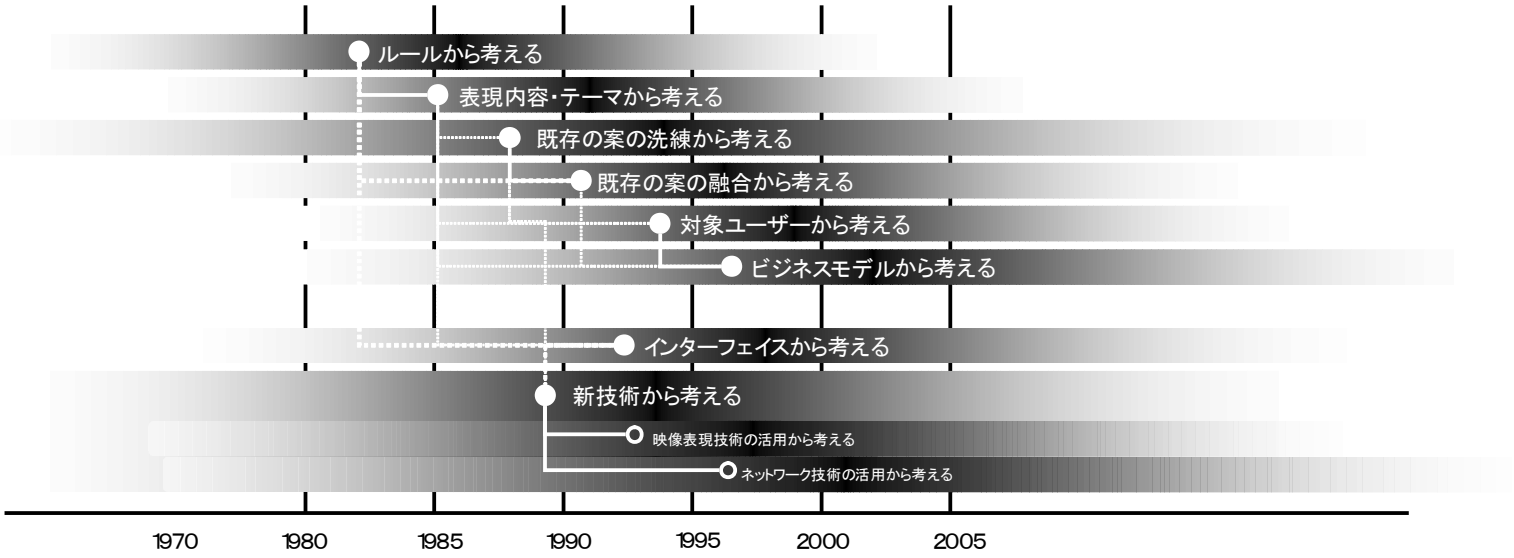


図 3.2-01　ゲームデザインの発想法の変遷史

ゲームデザインを行うための発想方法から、ゲームデザインについての分類を行ったものに Chris Bateman ら(2005)による分類があるが、この分類を参考にゲームデザインの発想法の変遷について整理すると、おおまかに

図のような変遷史を描くことができるだろう。いくつかの時期を大きく分けて説明しよう。

(a)　ゼロからの構築：ルール、表現内容から考える

まず、コンピューター・ゲームの登場から 80 年代中盤のファミコン・ブーム頃の時期までは、コンピューター・ゲームというメディアの中でいかに新しいルールを持ったゲームを設計するか、あるいは、新しい表現を行うか、といった発想法が多く見られる。例えば、その一つに『パックマン』の岩谷徹[1]の提唱している「動詞から発想する」という発想法が見られる。岩谷氏によれば、パックマンは「食べる」という動詞を元に閃きがはじまった、と言い動詞の数だけゲームのバリエーションがありうるのではないか、という説を唱えている。

また、70 年代～80 年代前半は、ゲームジャンルの基礎となった RPG、STG、SLG、ADV など各種のゲームシステムの大半が出そろってきた時期である。現在のゲームシス

1　岩谷徹『パックマンのゲーム学入門』(エンターブレイン, 2005)

テムのほとんどは、この時期に原型となるものが提出されている。

(b) 既にあるものを編集し直す：洗練、融合から考える

原型となるゲームシステムが構築された次の時期に来たのは、原型の洗練や融合である。形が整いつつあった RPG にさらに新しいシステムを付け加えたり、ゲームシステムと物語上の展開をうまくかみ合わせたり、といった形でのゲームデザイン手法が発達した。これは、ゼロベースからの新しい案を出すことに限界が見え始めた 80 年代後半から 90 年代かけて主軸となる方法論となってきた。

原型となるゲームシステムがある程度まで固定化されていれば、ユーザー側の期待も煽りやすくマーケット規模もある程度までは読みやすい。その上、ゲームの基本設計の大枠が決まっていれば開発サイドでも、開発規模や開発工程といった点について経験を蓄積しやすいというメリットもある。ゲームソフトのシリーズ化といった現象はシステムの洗練という発想をベースにして開発がはじまる代表的な事例である。

ゲームシステム間の融合については、例えば『サクラ大戦』などでは恋愛 ADV に戦闘 SLG の要素が加えられるなどといった形で、複数のゲームシステムを一つのパッケージに収められている。こうした融合事例では、複数のシステムが一つに収められているだけでなくシステム間の相互作用や、相互の矛盾などについても、いかにすればよりユーザーの支持が集められるか、がよく考えられてきた。

また、全く新しいゲームシステムを構想するのではなく、既存のシステムを基礎にしながら表現内容を変化させることなどによって新しい体験をもたらすといった試みも、90 年代以降のゲームデザインにおいては大きく注目を浴びている方法の一つである。例えば『絶体絶命都市』などは、ほとんど既存の 3D アクションゲームの方法しか用いていないが、舞台設定に阪神大震災を彷彿とさせるような世界観を設定したことで、新たな局面を切り開いた作品として評価された。

いずれにせよコンピューター・ゲームの産業初期と比べると、ゼロから新しいゲームシステムを作るという発想は非常に困難な試みとなってきている。

(c) 技術要因、ビジネス的要因等からの発想

さて、最後に技術要因や、ビジネス要因からゲームデザインを発想するといった経路がある。技術の改良的な発展は、表現の幅が拡がるという側面があるだけでなく一定の閾値を超えることで、全く新しいゲームデザインそのものを可能にするような側面も持っている。例えば、CPU/GPU 等の処理能力の向上は、より高精細な映像表現を可能にすると同時に、ある程度の高度な AI の挙動を前提としなければ作れないゲームデザインを可能にしたり、三次元の映像表現でしかありえないゲームデザインを可能にしている。『塊魂』などは、その好例だろう。こうした、技術要素の変化は、ゲームという表現がアナログか

らコンピューターになったことではじめて可能になった表現を生んだような、根本的な技術革新でもありうる。たとえば、Wii のようなインターフェースの水準からの作り変えは、ゲームデザイン単独での新しい発想や、新しい搦め手の手段を講じることが難しくなったところで、ゲームデザインをブーストさせるための有効な方法として機能している。

以上のような三つの経路は、ある時期からいきなり支配的なものになったわけではなく、ゲーム開発史のどの時期においても常に存在している。だが、大まかに言えば、ゲームデザイン単体で、ゲームのルールや表現をゼロベースから新しいものを考えるということは、年を追うごとに難しくなってきている。そのため、いかに新技術などと結びついた形での発想を考えていくか、ということはゲームデザイン分野においてもより重要性を増しつつあると言える。

(3) ゲームデザインの技術的領域

先述したように明確な技術の評価基準が成立しにくい領域であるため、これを「技術」と呼ぶべきかどうかにも、論争があるが、ここではゲームデザインとされていることの中でも、比較的合意が取れそうな事項を、いくつか紹介してゆく。

大きく分けると、1. ゲームとして成立させるための最低限の方法論　2. ユーザーエクスペリエンスを考える方法　3. 実製作において必要となる方法　の三つになる。

(a) ゲームとして成立させるための最低限の方法論

まず、ゲームをゲームとして成立させるための要件は、おおむね次の四つ程度の要素に還元されて語られてきた

(A) 固定されたルール：何をしたらどうなるか、についての決まり。
(B) 目標：プレイヤーが目指すべき到達点や、方向性を提示する。
(C) 結果のバリエーションが保たれること：毎回、同じ結果になってはいけない。
(D) インタラクションが成立していること：実質的に、何もやることがない、とプレイヤーに感じさせない。

（C+D を融合させた形で、意志決定のジレンマを生じさせることが要件の一つとして数えられる場合もある）

実製作の上で考えるべき要素としてはほぼこの四要素を満たせば、ほとんどの場合は大丈夫であろうと考えられる。

ただし、これらの非常に基本的な要素を作り変えてしまおうという野心的なゲームや、概念の中間領域に属するようなゲームも数多く存在する。例えば、『SimCity』には明確な目標は提示されないし、MMORPG のようなジャンルでは、ルールをどのように機能させるかがユーザーの自由で、ある程度まで変えられるものもある。こうした点について全て満足に議論するための「ゲーム」を機能させるための最低限の要素定義は、一律には難しいが、先に挙げた四条件よりもさらに包括的な定義を行うのであれば、たとえば、以下のような定義が可能である。

(A) 認識可能で機能するルール
(B) 目標への動機付け（内発的動機付け／外発的動機付けのどちらでもよい）
(C) 随伴性の認知を伴うインタラクション（自分の行為によって何かが変わったということが認識可能）

この他にも、意志決定という要素に着目して概念化を行う定義や、プレイヤーの行為のバリエーションのダイナミクスとゲームのバリエーションをもとにモデル構築を行う MDA モデルなど、ゲームが成立するための最低限の要素定義については多種多様なモデル構築がすすめられている状態である。この点について、より踏み込んで議論しているものとしては、Jesper Juul『Half Real』(2003、MIT Press)、Eric Zimmerman & Salen『Rules of Play』(2003、MIT Press)などが知られている。日本語文献では、井上明人「遊びとゲームをめぐる試論」（『未来心理 vol.13』所収、モバイル未来社会研究所、2008）などがある。何が「ゲーム」として機能するための最低限の要件でありうるのかに関する学術的な議論において、まだ決着は付けられていない。

(b) ユーザーエクスペリエンスを考える方法

ゲームデザインの独自の技法として洗練が重ねられてきた領域の一つは、プレイヤーがプレイ全体を通して、どういった経験をするのか、という観点からゲーム全体調整していく技法だろう。

とりわけ①レベルデザイン、②テストプレイ、③インターフェース・デザイン、④マーケティング、といった領域は、ゲームデザインといった概念そのものを離れて独自の技術領域として括られる傾向も強い。マーケティングや、インターフェース・デザインが専門領域として機能していることは、言うまでもないが、レベルデザインについても、レベルデザイン専用のツールがこの 10 年程度の間に大きく発展したことによりレベルデザインのみを取り扱った専門の書籍などが登場してきた。（例えば、Phil Co『レベルデザイナーになる本』ボーンデジタル,2006 など）

① レベルデザイン

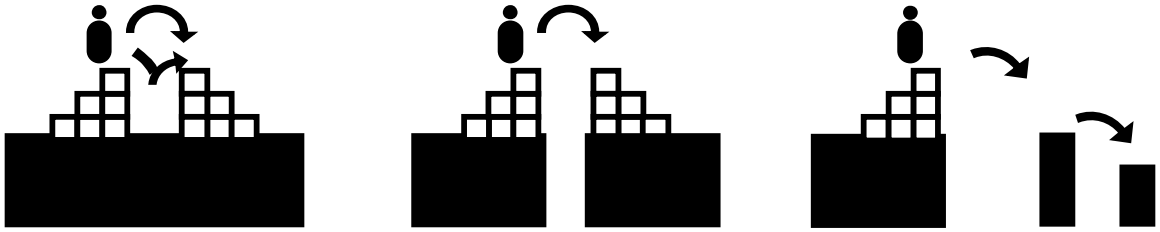


図 a　　図 b　　図 C

図 3.2-02　レベルデザインの事例

レベルデザインを説明するための事例としては、上記のような『スーパーマリオブラザーズ』のマップ構成がしばしば取り上げられる。『スーパーマリオブラザーズ』でのジャンプを使わせるシーンは図 a、図 b、図 c のような順序で配置されている。すなわち、ジャンプを行うという動作を楽しませるための仕掛けとして、a.アクションを覚えさせるシーン、b.アクションを使わせるシーン、c.アクションを応用・究めるシーン、という形で段階的な学習を行わせている。こうしたマップ構成によるプレイヤー経験を開発者側で調整する行為がレベルデザインと呼ばれる。

レベルデザインは、『スーパーマリオブラザーズ』のような面クリア型のゲームの中での段階的学習だけでない。FPS などの場合には、a.隠れる場所、b.多人数の銃撃戦の舞台となる踊り場、c.一対一で向き合う狭い廊下、などといった経験のバリエーションによってマップを作り込むという要素が重要になる。2D シューティングでは a.敵の攻撃が激しい所、b.敵の攻撃が緩いところ、といったマップが交互に入れ替わることでプレイヤーは、気が抜けるかと思ったらいきなり緊張し、緊張しているかと思ったら気を抜いてよくなるなど、ゲームプレイが常に刺激的に機能するための方法論が、蓄積されている。

表 3.2-02 に、こうしたレベルデザインの主要な手法および、レベルデザインに深く関係する要素をまとめた。

表 3.2-02　レベルデザインの主要な手法および要素

段階的刺激	・ 段階的学習：敵や障害物のレベルが順に高くなる。一段階前のスキルの積み重ねで、クリアできるように設計する。 ・ ヘルプシステムを用意する：プレイヤーがわからなくなった時に参照可能なシステムの準備 ・ オープニングデモによるゲームの内容解説 ・ レベルアップの速度調整：前半に加速度的に強くなるか、後半で加速度的に強くなるか、一定の速度で強くなるか
刺激の間隔調整	・ 緩急を 30 秒～1 分程度の間隔で付ける：アクションゲームの場合 ・ レベルアップの際に、強くなったことを実感させる仕組み：RPG などの場合 ・ プレイヤーにストレスを意図的に溜めさせて、ほどよいところで課題を解決できる程度のゲームバランス調整（ギリギリでのクリアー）
刺激のバリエーション	・ 銃撃戦の場所のバリエーション（踊り場、廊下、隠れ場）、銃のバリエーション（散弾銃、ピストル、ライフル）： FPS の場合 ・ 敵キャラクターの攻撃方法のバリエーション（直線に進む敵、ルートを巡回する敵、特定のポイントを監視する敵） ・ 難易度選択、エリア／ルート選択など （関連）効果音のバリエーション （関連）必要となるアクションのバリエーション（ジャンプ、パンチ、キック）
その他	・ ゲーム初期のうちに、最終到達ポイントや全体像を提示する ・ ゲームマップの全体認識を、段階的に大きく変更を加える

なお、この他にも、一見、弾幕が激しく難易度の高そうなエリアだと思わるが実質的にはアタリ判定が緩く攻略可能なエリアの作成することによる錯覚効果や、クリア方法が、他のエリアと同様になっていることでバリエーションを付けつつも基本的なプレイにおいては迷わせることがないようにする類似性を用いた認知コスト低減など、多種多様なレベルデザイン手法がある。

② テストプレイ

また、現在、手法の洗練が大きく求められている領域としてα版やβ版のテストプレイをどのような形で行っていくかということが挙げられる。開発に十分な期間やコストをかけられなかったために、開発当事者によるテストプレイのみを行った結果、ユーザーエクスペリエンスを大きく損なうということは頻繁に生じる。具体的には、はじめてプレイするユーザーがどこで躓きやすいかといったことがわからなくなり、過度に難しすぎるゲームができてしまったり、逆に簡単すぎるゲームをつくってしまったりといったことである。

この問題は、非常によく知られてはいるが、どの程度までテストプレイに手間をかけられるかどうかは、実際の開発期間や開発コストとのトレードオフとの関係にもあり、どの程度効率的なテストプレイを行えるか、は開発における一つの胆となる。たとえば、作り込んだ CG が「ウリ」の部分となるような大規模な RPG の開発では、開発プロジェクトの終盤になるまで、ゲームの面白さが見えにくい。そのため、ゲーム開発の初期からテストプレイをどの段階でどのように行っているかを、計画的に設計していく必要が生じる。

主要な方法について表 3.2-03 にまとめた。

ゲームの開発規模が大規模なものになればなるほど、ゲームの個別の要素のテストプレイが、どこまで全体としての経験と整合性を持つかといったことが重要になってくる。たとえば、『スーパーマリオブラザーズ』の開発で、宮本茂は当初から BGM によってメモリが食われることを見越し、プログラム開発時にも関係のない BGM を組み込んで開発をすすめるなどという形で、可能な限り最終形をイメージしながらの開発を行っている。

また、一部の格闘ゲームの開発などでは、一般ユーザーに加えて非常にスキルの高いプレイヤーによる評価も重要になるため、有名なゲームプレイヤーをテストプレイヤーとして採用するといった形で開発が行われている。

表 3.2-03　テストプレイの主要な方法

手法		コスト	備考
人	テストプレイチームの編成	大	アルバイトなどを雇ってのテストプレイチームの編成、テストプレイ専門の組織へのアウトソースなど。プロジェクトの当初から開発予算に組み込む必要がある。 別部門として組織されてしまうために、開発プロセスの中に踏みこめなくなることもある。
	他部署の社員によるテストプレイ	中	他部署の人間に一時的協力してもらう。社内調整能力の高い開発者にとっては、簡単な手法。
	開発当事者によるテストプレイ	小	開発当事者によるテストプレイ。最も簡単な手法だが、視点に欠けが生じやすい。
ツール	アイトラッキングシステム／脳波測定など	大	最も先端的な手法。ただしシステムそのものを用意するために必要なコストも高く、データを解釈するための手間も必要になる
	ビデオ撮影 キーロガー ヒートマップ	中	同時にディスプレイを並べて複数のテストプレイを再生したり、複数の開発者が同時に一つのテストプレイ動画を評価できるという利点がある。
	テストプレイヤーからの報告（アンケート／自由記述）	中	レポートによる報告。自由記述の場合は、テストプレイヤーがレポートをまとめる能力によってレポートの質が変わるという問題はあるが、テストプレイを見ているだけではわからない問題を洗い出せる可能性がある。
	後ろからの観察	小	簡単な手法であるが、テストプレイヤーが躓いているところなどを見て、その場で何がコミュニケーションを取ることができる。手軽に、何度にできるという利点も大きい。

③　インターフェース・デザイン

インターフェース・デザインについては、インターフェース研究の長い歴史が存在するが、コンピューター・ゲームのインターフェースに関しては、ウェブサイトのインターフェースや、日常生活における道具の工業デザインと共通する点と、そうでない点がある。

まず、共通する点としては、あたりまえだが、使いやすいインターフェースを設計することが求められる。こうしたインターフェース設計の細かいノウハウについては、インタ

ーフェースの使われる状況や、インターフェースの特性によって無数の要素が挙げられる。こうした点については、ドナルド・ノーマン『誰のためのデザイン?』(1990、新曜社)などを参照されたい。なるべく自然物のメタファーなどが流用できるようなインターフェース設計になっているかどうか、といった点はゲームでもしばしば重視される。

もう一方で、どこまで既存のインターフェースとして出回っているものを利用するかどうかという点については、ゲームと工業デザインなどでは判断が異なる。たとえば、アクションゲームを作る場合には、いかに新しく楽しいインターフェースを作るかといった点がゲームの重要な要素となる場合が多い。ただし、工業デザインなどでは、新しいインターフェースを採用することは、ユーザーにとっては認知コストが上昇するため、特に必然性がなければ一般的なインターフェースと異なる仕様を設計しても受け入れられないことが多い。

ゲームのインターフェースについて論じたものとしては、サイトウ アキヒロ,小野 憲史『ニンテンドーDS が売れる理由―ゲームニクスでインターフェースが変わる』(秀和システム、2007)などがある。

④ マーケティング

実際に、ユーザーエクスペリエンスを考える上では、広告などを含めたマーケティングもかなり重要な要素として機能する。例えば、アクション RPG を、RPG と誤解させるような広告を行ったために目標とするユーザーセグメントと、実際の購買層とに齟齬が生じて、ゲームの評価が低くなるといった事例は多い。

たとえば、『ファミ通』などゲーム・ジャーナリズムによる評価が高いにも関わらず、インターネット上のユーザー評価では評価が低いといったソフトは少なくない。こうしたソフトでは、開発現場とマーケティング部門との間での調整が取れていないことが多い。

社内のガバナンスの問題でもあるが、海外への販売戦略・ローカライズなどが行われる際には、一般的に社内のコミュニケーションコストも増大するため、マーケティング上の問題も発生しやすい。

ゲームデザインとマーケティングの関わりについて論じた書籍としては、Chris Bateman, Richard Boon “21st Century Game Design” (Charles River Media, 2005)などがある。

(c) 実製作において必要となる方法

① ビジネススキル、プロジェクト管理能力

ディレクターや、プランナーと呼ばれる業種の実際の業務は、基本的には個人製作ではなく集団製作であるため、ゲームデザイン単独のスキルというよりは、ビジネススキル一般に属する能力が必要となる。具体的には、コミュニケーション能力や、わかりやすいド

キュメントを書く能力といった要素だが、こうしたノウハウについてはゲームデザイン独自の技術という側面は弱いため本稿では詳細しないが前田圭士『ゲームデザイナーの仕事』（ソフトバンククリエイティブ,2008）などで詳しく紹介されている。

② 開発工程に関する理解

作業全体の上流工程となる企画書の作成などは、より下流の工程（プログラム、データ作成、バランス調整）の作業に大きな影響を与える。下流工程の作業に入っている瞬間に、上流工程に差し戻して作業するとなると、大きく開発コストを失うことになる。
この点は、ソフトウェア工学などの研究領域であり、上流工程から下流工程にむかって流れ作業で作成していくウォーターフォール型の開発の他にも、アジャイル型開発の体勢など様々な開発工程が模索されている。

③ 関連ドキュメントの作成：仕様書や企画書の作成

企画書については、1 ページ～3 ページ程度で作成し、短くてわかりやすく、実際に社内で通る企画を書くテクニックが求められる。

仕様書については、プロジェクト管理の手法とも結びつくため、プロジェクトの内容二応じて、仕様書ベースではなく、ウェブ上の情報共有ツールやプロジェクト・マネージメントのための開発工程管理のソフトウェアなどによって一部機能が代替されるなど、近年では仕様書に類するドキュメント作成にはバリエーションが増加しつつある。

この点についても前掲の前田圭士『ゲームデザイナーの仕事』や、山本貴光、馬場 保仁『ゲームの教科書』（ちくまプリマー新書、2008）などが詳しい。

④ 扱う対象についての知識収集／モデル化の技法

ゲームの中で何を表現するかということがゲームデザインを発想する経路の一つであることを述べた。これには、a.先述の岩谷が提案するような「動詞」からの着想という抽象度の高い表現の他に、 b.サッカーや競馬などのより具体的な題材を表現対象とする場合が挙げられる。

抽象度の高い概念を表現対象とする場合にはインターフェースの練り込みなどが重要になると考えられるが、サッカーや競馬などの、具体的な題材を扱う場合には当該対象についての知識収集はむろん不可欠となる。さらに重要なのは、誰の視点からどのように現象を抽象化し、ゲームシステムとしてくみ上げるかというモデル化の能力となる。

こうしたモデル化の方法論等については、それほど一般的な議論を行っているものはあまり見ないが、『SimCity』などで知られるウィル・ライトなどは、ネットワーク論や、社会システム分析、経営システム論、進化ゲーム理論など社会科学の知見を現象のモデル化のために応用している。今後、シリアス・ゲーム研究などでもこうしたモデル化の技法についての研究は進展してゆくものと考えられる。

⑷ゲームデザインにおけるトレードオフ

ゲームデザインにおいて、より微妙な問題は、限りのある方法論の中で、複数の要素のバランスの微妙さをどのように感じ取れるようになれるか、といった感覚の問題がある。まず、一般的なトレードオフの言葉で表現すれば、

- 特定のユーザーセグメントに特化するか、万人向けにするか（あるいは、どのユーザーセグメントを狙うのか）
- 新しい技法に挑戦するか、すでに確立され洗練された技法を用いるか
- 一点の要素に注力するか、多くの要素の作り込みに注力するか
- 垂直統合型の開発を行うのか、水平分離型の開発を行うのか
- 大規模なプロジェクトにするか、小規模なプロジェクトにするか

といった要素が考えられるが、これをゲームデザインの問題に置き換えると表 3.2-04 のようにまとめられる。

表 3.2-04　ゲームデザインのトレードオフ

一般的なトレードオフ	ゲームデザインの場合のトレードオフ（例）
特定のユーザーセグメントに特化するか、万人向けにするか（あるいは、どのユーザーセグメントを狙うのか)	・ 題材のリアルなシミュレーションを重視するか、なんとなくの「それっぽさ」を重視するか ・ Liner か Sandbox か(一本道のゲームか、自由度の高いゲームか。) ・ 懇切丁寧にレスポンスのわかりやすいゲームを作るか、ゲームの仕組みをプレイヤーが一つずつ理解していくようなゲームを作るか ・ コミュニケーション重視か、対戦重視か（オンラインゲーム） ・ 成長要素を入れるか、入れないか
新しい技法に挑戦するか、すでに確立され洗練された技法を用いるか	・ 一般的なインターフェースを採用するか、新しいインターフェースを作るか ・ （原作のゲーム化などで）原作に忠実に作るか、既存のゲームシステムの上で作りやすいものを作るか
一点の要素に注力するか、多くの要素の作り込みに注力するか	・ 複合的なモード（学園／戦闘／恋愛）のゲームを作るか、単一のモードのゲームを作るか ・ 複合的なゲームスケール（戦略級／戦術級／戦闘級）を作るか、単一のゲームスケールのゲームを作るか ・ やりこみ要素の充実したものを作るか、シンプルな内容のゲームを作るか ・ 大規模なプロジェクトにするか、小規模なプロジェクトにするか
垂直統合型の開発を行うのか、水平分離型の開発を行うのか	・ モジュール型の開発アーキテクチャ（ミドルウェア）を用いるか、インテグラル型の開発を考えるか。 ・ アジャイル型開発か、ウォーターフォール型開発か
その他	・ 一人称視点をメインにするか、三人称視点をメインにするか ・ プレイヤー・キャラクターの位置付けをどうするか ・ カット・シーンを多く入れるか、少なくするか

(a)トレードオフの例

こうしたトレードオフの構造は、あるところまでは両立しないが、ある場合には単純な対立構造でもないような微妙な関わりを持っている。ここでは、シミュレーションと、物語に関するトレードオフという二つの事例について紹介する。

① シミュレーションにおけるトレードオフ

これらのトレードオフ関係は非常に込み入っているが、敢えてこれらのトレードオフ関係をより整理した形での把握を例示するならば、おそらく要素間の関係性をネットワーク図にして表現するという方法が考えられる。一例として、戦闘機のゲームを作る際に、リアルなフライトシミュレータを作るか、わかりやすい楽しさのある戦闘機ゲームを作るか、といったトレードオフにおいて、当然関係すると思われる要素をいくつか抜き出したものが図 3.2-03 である。

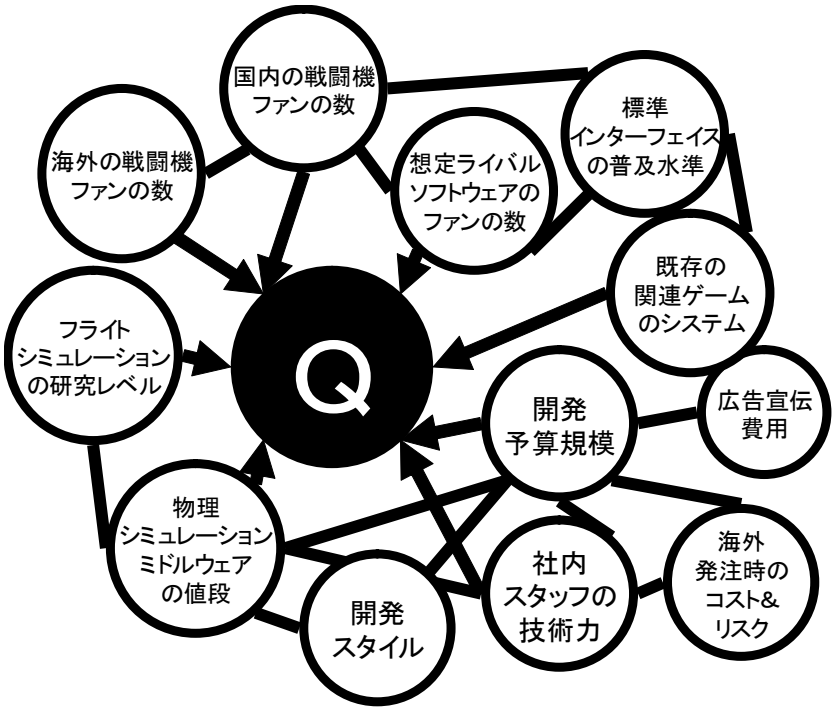


図 3.2-03　要素間の関係性

例えば、リアル志向のシミュレーションソフトは、リアルすぎて難易度が高くなり、多くの人にとっては面白くなくなる場合が多いとされる。そのため、物理的にはあり得ない動きではあっても、プレイヤーにとって「それっぽい」と感じられる操作のできるゲームを作ったほうがゲーム作りとしては正解である、というのがスタンダードな議論としては存在する。だが、一方で、題材となっているモチーフに対してコアなファンが数百万人規模で存在するようなゲームの場合には、そのモチーフをかなりリアルに作り込んでも、特定ファン層に向けて販売することが可能なのであれば、一般向けよりもややリアル志向のシミュレーション重視のゲームが売り上げを得ることは可能になる。アメリカン・フットボールや、バスケットボールのゲームなどでは、アメリカと日本とでは大きく売り上げの異なるソフトが存在するが、これはそうした背景に起因している。

② 物語におけるトレードオフ

また、物語表現とゲームの間にまたがる関係性の複雑さなどは、根が深い問題である。一般には、一本道の物語 RPG と、自由度の高い Sandbox タイプのゲーム作りに対立がある、という認識があるがこの対立項は、実際には遙かに複雑な構成を持っている。

日本語で「物語」と言った場合、小説、マンガ、映画などのようなメディアに記録された起承転結のあるもののことを一般に指している場合が多いが、物語研究ではこうした意味での物語は"Story"と呼ばれ、Story が紡ぎ出されることは Story Telling や Narrative という概念によって区別される。日本語では、物語 vs ゲームという構図での対立図式がどうしても多くなってしまうが、ゲームは Story のメディアとは対立的になりやすいが、ゲームと Story Telling という区分は必ずしも対立的な要素とはならない。例えば、コンピューターRPG の元となった、TRPG は Story Telling システムとしての役割を備えている。そして、Story や Story Telling の要素となる事件 Event もまた、これらの概念とは区別される。Story の定義の一つとして、複数の事件 A、B が時間的な因果によって把握されたものを Story と呼ぶといった Story 定義も、物語論では一般的な定義の一つである。非日常的な経験であることを Story の要件に加えるべきかどうか、といった点もこの概念に関する論争では数多く見られる。

つまり、簡単に整理すれば、

a. 開発者が準備した Story をいかにしてプレイヤーに体験させるかという問題
b. 開発者が準備した Event をプレイヤーに体験させられるか、という問題
c. プレイヤーがゲームの中で何かの Story Telling を経験するか、という問題

はそれぞれ、別々の水準の問題であり、区別して論じることができる。Henry Jenkins[2]など、海外のゲーム研究の議論の中ではこうした物語論の議論の蓄積の上に、空間と物語の相互関係などを整理する議論を提出している。一方、日本でも前田圭士[3]など、選択肢に関する分類など、物語とゲームの関係性に関する議論は一部で対立し、一部で区別が不可能になるような微妙な関係性を有している。

(b) トレードオフをどう処理するか

市場の状況、技術動向、開発に使える社内リソース／社外リソース、関連ゲームシステムなどの微妙な部分のできの良し悪しなどに関する判断には、こうした形で、多数の要素が複雑に絡む。そのため、トレードオフに対してどれだけ敏感に総合的な判断が可能か、

[2] Henry Jenkins "Game Design as Narrative Architecture"
http://web.mit.edu/cms/People/henry3/games&narrative.html
[3] 前田圭士『ゲームシナリオライターの仕事』ソフトバンククリエイティブ、2006

は「センス」などという観点で片付けられることが多い。実際に、全てを理屈のみで、記述的に理解させようと思うと、膨大なドキュメントを必要とするか、分析的に図 3.2-0 のようなネットワーク図を描き、個々の条件要素を洗い出しての記述になるため、パターンとして一般化するよりも「センス」の水準で考えたほうが効率的になる場合が多い。ただし、既存のゲーム業界の文法等の「センス」をベースにした発想法では、どうしても過去の経験則に偏る傾向がある。新しい試みを考えるときにはこれが障害として機能することも少なくない。

こうしたトレードオフの問題の微妙さを解決するために、複雑性が高い状況を処理する方法としては下記のような手法から、問題解決を考える手段がある。

① 問題の相互依存関係から、ボトルネックになる部分を洗い出すようなネットワークの分析を行う

分析のために必要な時間や抽象的な概念化の能力を必要とするため、必ずしも簡単な方法ではないが説得的、かつその後のプロジェクトにも再利用可能な分析となるかと考えられる。

② 実際にモックアップを作成し、面白さの片鱗を感じられるかどうか。アウトプット優先の検証プロセス

モックアップを作る手間が簡単であれば効率的な手法となる。実機で動くα版を作るという段階もあるが、紙芝居やアナログゲームで簡単なものを作るといった方法も存在する。

③ ブレインストーミングなどの玉石混合でのアイデア出し

KJ 法など、発想法を用いて行う手法も有効。

以上、ゲームデザイン分野は、技術という発想が必ずしも素直になじまない領域である。そのため、技術にならない領域をいかに「技術」という視点から構成できるか、は今後もゲーム業界にとって重要な課題であり続けるだろう。

3.2.2　素材作成　CG(3D)

川島 基展
東京工科大学 片柳研究所クリエイティブ・ラボ

(1)　3DCG 技術の歴史

図 3.2-04 は、リアルタイム 3DCG 技術のロードマップを示したものである。

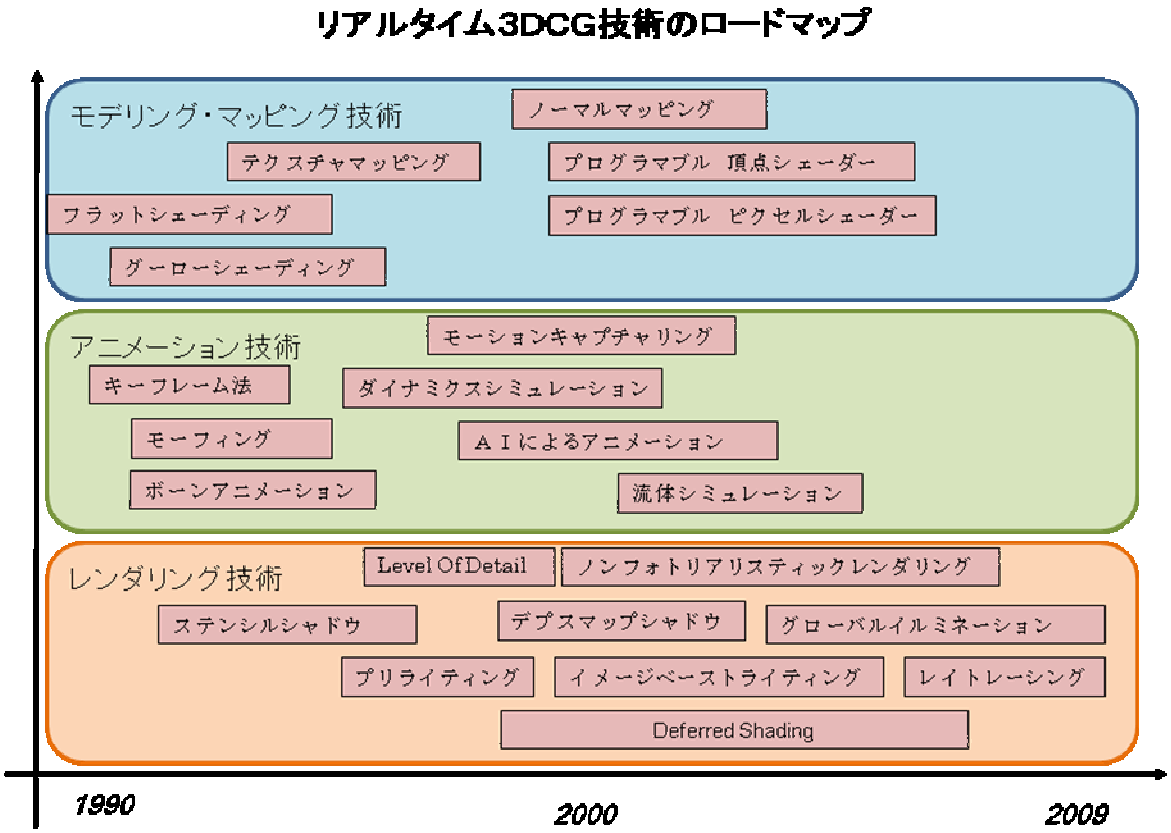


図 3.2-04　リアルタイム 3DCG 技術のロードマップ

(2)　3DCG の各要素技術

図 3.2-04 のロードマップに沿って、各要素技術について解説する。

(a)　モデリング・マッピング技術

①　モデリング

3 次元空間の中に、キャラクターやメカ、背景などの形状を表現する工程を、モデリングという。モデリング工程では、点・線・面の 3 要素を使って、立体形状を表現する。面の表現には、3 つ以上の点を結んだ多角形である**ポリゴン**の集合体を用いるのが一般的である。図 3.2-05 は、ポリゴンによって、単純な形状を表現した例。多面体の直線的なライ

ンだけでなく、球体の曲面も、複数の 3 角形ポリゴンを用いることによって表現している。

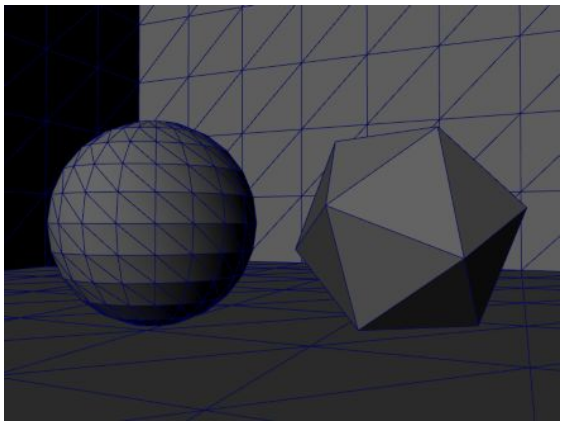


図 3.2-05　ポリゴンによるモデリングの例

ポリゴンによるモデルは、平面の集合体で構成するため、かつては、キャラクターや有機的な曲面を表現するのが困難であった。現在は、ポリゴンをパラメーター化し、NURBS やベジエなどの曲面によって制御することで、滑らかな面を簡単に作成できるようになった。また、図 3.2-06 のように、**サブディビジョンサーフェイス**と呼ばれる細分割技術を用いることで、粗いポリゴンモデルから、滑らかで複雑な曲面を作成することができる。

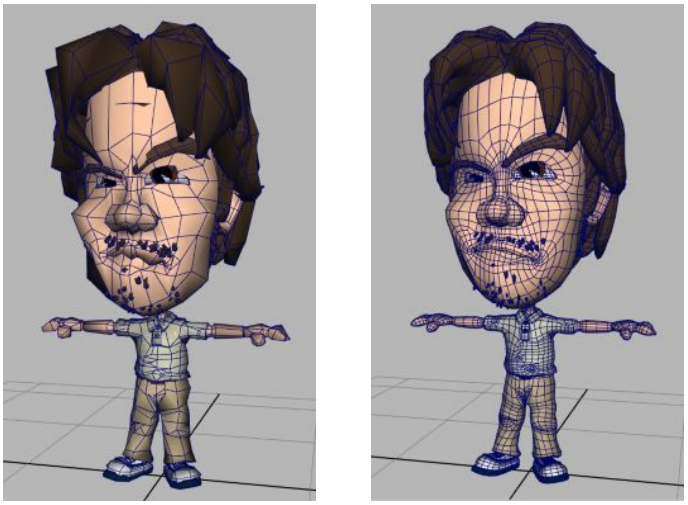


図 3.2-06　サブディビジョンサーフェイスによるキャラクター・モデリングの例

左が細分割前、右が細分割後

ただし、ゲームにおいてリアルタイムグラフィックス表現を行うためには、可能な限りポリゴンのデータ量を減らして描画コストを抑える必要があるため、最終的にこれらを通常のポリゴンデータに変換し、最適化する必要がある。近年のハードウェアの進歩により、

リアルタイムで描画可能なポリゴンの数が増しており、より高精細なモデルによる表現が可能になった。

モデリングのための操作は、ソフトウェアによって多少の違いがあるが、基本的には、次のようなオペレーションを併用する。

- 点・線・面の移動・回転・スケール
- スイープ操作
  - 面・線・点の押し出し
  - 断面形状の回転
- CSG（ブーリアン演算）

② 質感設定・マッピング

モデリングされたオブジェクトに対し、次のような質感特性を設定していく。

- 拡散反射光(diffuse)
- 鏡面反射光(specular)
- 環境光(ambient color)
- 透明度(transparency)
- 凹凸変化(bump/normal map)
- 反射率(reflectivity)
- 屈折率(refraction index)

これらの質感特性について、図 3.2-07 のように、画像（テクスチャマップ）によって、細かな変化を表現することが可能である。近年のハードウェアでは、モデルの表面の凹凸変化の情報を法線マップとして与えることにより、少ないポリゴンのデータに対し、高精細な凹凸のディティールを与えることができる。

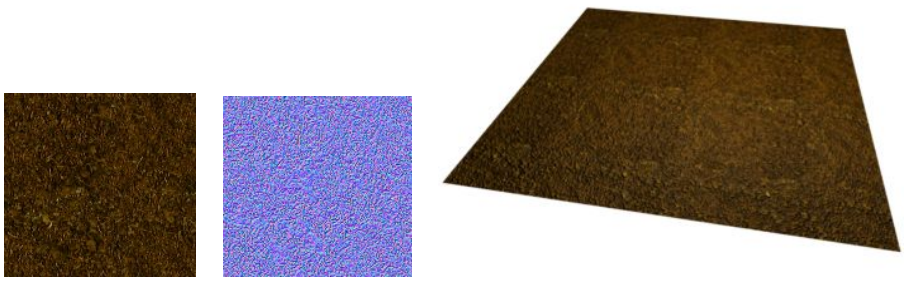


図 3.2-07　拡散反射光マップ（左）、凹凸変化のための法線マップ（中）、レンダー結果（右）

また、近年では、プログラマブルな頂点シェーダー、ピクセルシェーダーを適用するこ

とにより、標準的なシェーディング技術では難しい高度な質感や、そのアニメーション表現を行うことができるようになった。反射や屈折現象についても、GPU などのハードウェア処理を活用することで、表現することができる。

(b) アニメーション技術

3DCG でのアニメーション設定は、次のような手法を組み合わせて行う。

① キーフレーム法

キーフレーム法は、3DCG アニメーションの最も一般的な設定手法で、今日でも主流となっているアニメーション設定である。まず、一連のアニメーションの中の重要なポーズであるキーフレームを設定し、その後、キーフレーム間の補間を調整することによって、ポーズと部位の軌跡を仕上げる。

図 3.2-08　キーフレーム法によるキャラクター・アニメーションの設定例

② ダイナミクスシミュレーション

デジタル技術を使う上での大きなメリットの 1 つは、コンピューターが得意とするシミュレーションを活用できることである。ひらひらした衣服や髪の毛、水の流れや雲、爆発など、いろいろな現象を物理計算を用いてリアルに表現することを、ダイナミクスシミュレーションという。ゲームコンテンツにおいても、ハードウェアの処理能力の向上により、ダイナミクスシミュレーションをリアルタイムで行い、表現することが可能になった。

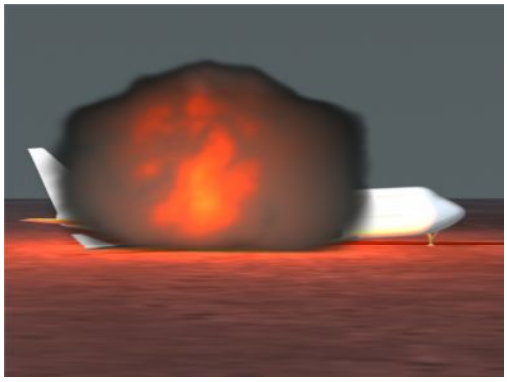


図 3.2-09　ダイナミクスシミュレーションによる爆発表現の例

③　サンプリング

アニメーションを手作業やシミュレーションで作成するのではなく、さまざまな外部入力デバイスによってサンプリングする手法がある。ゲームコンテンツ制作において用いるサンプリングの代表的な手段は、俳優の演技を取得可能な**モーションキャプチャ**である。

システムの方式には、ギブスのようなセンサーを身につける**機械式**や、磁場を利用する**磁気式**など、さまざまなものがあるが、3DCG アニメーション用途では、俳優の演技の邪魔にならない、図 3.2-10 のような**光学式**が最も一般的である。モーションキャプチャは、リアリスティックな演技を使って効率的にキャラクタアニメーションを制作することのできる手段であり、とくに多くのキャラクタアニメーションデータが必要になるゲームコンテンツ制作においては、今後ますますキー技術として注目されていく技術である。

図 3.2-10　光学式モーションキャプチャの例

モーションキャプチャを使う場合は、モーション撮影後のデータをアニメーションとして編集し、仕上げる作業もアニメータが行う。

(c)　ライティング・レンダリング技術

①　ライティング

ゲーム・コンテンツにおける CG グラフィックス表現は、手描きや実写撮影などの、さまざまな要素を組み合わせて行うものである。キャラクターや背景の陰影づけには、実写撮影と同じく、ライティングを施す。

キーライト、フィルライト、バックライトのような演出的なライティングに加えて、日光や地面の照り返しなどの自然界の光源や、爆発の閃光や魔法効果など、特殊効果用の光源設定なども、ライティング担当者の仕事である。

代表的なゲームエンジンにおいては、次のようなライティング手法を用いることができる。

- ポイントライト（点光源）
  - 1 点から全方向を照射
- ディレクショナルライト（平行光源）
  - 太陽光など、指向性のある光源
- スポットライト
  - 照射範囲を絞ることができる
- エリアライト（面光源）
  - 蛍光灯など、面積をもつ光源
- ボリュームライト（体積光源）
  - 提灯や光の玉など、体積をもつ光源

図 3.2-11　屋内照明（左）と、屋外照明（右）の例

②　レンダリング

構築したシーンを、2D の画像として出力する工程を、レンダリングという。レンダリングの計算アルゴリズムは、写実性を追求する**フォトリアリスティック・レンダリング**と、絵で描くようなタッチで表現する**ノンフォトリアリスティック・レンダリング**の 2

つに大別され、それぞれ、次のような計算アルゴリズムを組み合わせて使用する。たとえば、手描きのアニメーションのようなタッチの画像を得るためには、トゥーンレンダリングが有効である。

いずれについても、近年のゲームエンジンにおいては、高度なシェーダーやポストプロセスを利用することにより、リアルタイムでこれらの表現を実現することが可能になった。また、Deferred Shading のように、プリレンダリングにおけるコンポジットに類似した手法により、高度な画像処理を高速におこなうことが可能となり、それらを活用した事例も多く見られるようになった。

- フォトリアリスティック・レンダリング
  - スキャンライン法
  - レイトレーシング法
  - グローバルイルミネーション
  - イメージベーストライティング

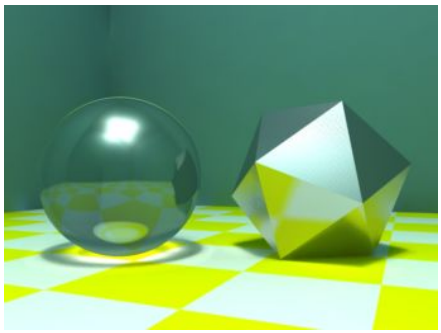
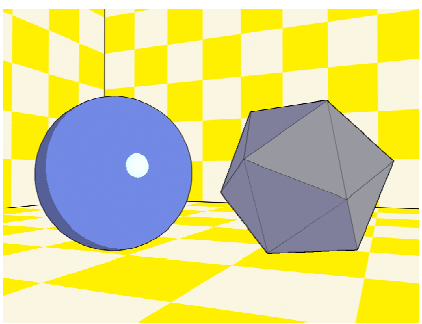


図 3.2-12　フォトリアリスティック・レンダリングの例

- ノンフォトリアリスティック・レンダリング
  - トゥーンレンダリング
  - 手描き・水彩調レンダリング



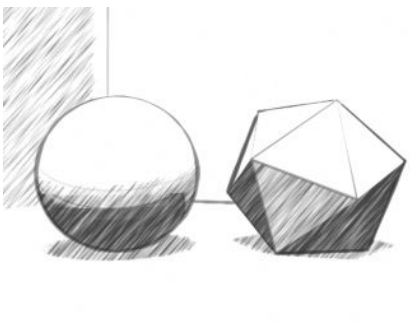


図 3.2-13　ノンフォトリアリスティック・レンダリングの例

左がトゥーンシェーディング、右が手描き調レンダリング

参考文献：
［1］栗原恒弥、安生健一：　“3DCG アニメーション”，技術評論社（2003）
［2］デジタル映像表現編集委員会：　“デジタル映像表現―CG による映像制作―”，財団法人　画像情報教育振興協会（CG-ARTS 協会）（2006）

3.2.3　素材作成　サウンド

細江　慎治
株式会社スーパースィープ

(1)　1980 年初頭のゲーム機創世の頃から現代そして未来へ

音場はモノラルからステレオを経て多チャンネルサラウンドへ、音源も単純な PSG からサンプリング、そしてストリームへと進化してきた。初期 PSG 時代はあまりの制限の多さに、音楽家よりソフトウェアエンジニアが音を作る事も少なくなかったが、よりよい音楽や SE（サウンド・エフェクト）を作る為に、音楽自体を作る作曲家と実機用のデータに落とし込むエンジニアで分業をする会社も出てきた。

同時発音数 3 声、4 声の中でいかに楽曲を成り立たせるか。作曲家には小編成で成り立つ音楽というスキルを要求し、またデータ化には ROM 容量の制限や、良い音を出す為にアセンブラレベルでのデータ化が要求された。この例は 1 つの極端な例で、実際には音楽や音を作る専門の人置くこともなく、音楽をかじった程度の誰かが「でっちあげていた」ケースが多かった。

世にファミコン（任天堂ファミリーコンピューター）の開発がこなれたあたりで、分業化は徐々に進んで、ハードの制限がありながらも作曲家のスキル以上にエンジニアやプログラマーの腕 1 つで想定していた音により近づける事が出来るようになった。

1990 年前後でファミコンには拡張音源が積まれるケースが増え、PC エンジン（NEC）等の新たなハードが発売され、音源自体のパワーアップも図られ、ある程度は作曲家のイメージに近い音楽が作れるようになってきた。また、MIDI（Musical Instrument Digital Interface）したツールや、SMF（Standard MIDI File）をそのまま使えるようなドライバーも徐々に開発され、テキストエディタでしか制作する事が出来なかった従来のスタイルにかわって、作曲家でもゲームの音を作れるように進化を遂げてきた。

その後 PSG はサンプリング音源へと進化したが、また新たなサンプリング技術による差が生まれたが、その後サンプリング音源の要である ROM 容量や内蔵 RAM の増加と共に徐々に緩和されてきた。また CD-ROMROM（PC エンジンの拡張版）から始まった

CD メディアを記憶媒体としたゲーム機の登場により、部分的に音楽は一般音楽と全く同様の作りが可能となったが、インタラクティブにコントロール出来る程のパフォーマンスやノウハウの蓄積がなく、単純に BGM を流せる程度にとどまった。

ハードの進化はさらに進み、CD メディア等の光学ディスクからの読み込みが高速化、また内蔵メモリの増加のおかげでゲームデータと音楽データの同時読み込みが出来るようになり、それを支援するミドルウェアが多数出てきた。ここまでくるとゲームに使用する音楽は一般の音楽家が普通に作る事が出来るようになったがそのシステムをどう使うかというノウハウは音楽家の領域ではなく、ソフトウェアを作る側の領域だった。

現代において、さらに進化したか？ゲームにおいての音楽の進化はここでほぼ終了してしまう。映画音楽に関しても、サラウンドまで使った音楽というのは稀で、基本的にステレオで制作されて終わる。マルチチャンネルは SE にのみ使われ、より SE を効果的に演出する為にもそれ以上のチャンネルで音楽を制作する事は滅多に無い。ただ若干ではあるが実験的に音楽もマルチチャンネル制作される事があるが定番化はしていない。

サンプリング（音声等）の進化は PSG の時代にも存在したが、当時の総容量自体が数 K バイト～数十 K バイト程度だった為、K バイト以下のオーダーで作る事が多かった。音源自体のバッファ容量のサイズ制限もあるが、CD-ROM が使われるようになるまでは総容量次第で、とても自由に使える容量ではなかったが、現代に於いてはシリコンメディアも大容量化してきたので、ネックは音源の RAM 容量に移り変わってきた。（図 3.2-14 参照）

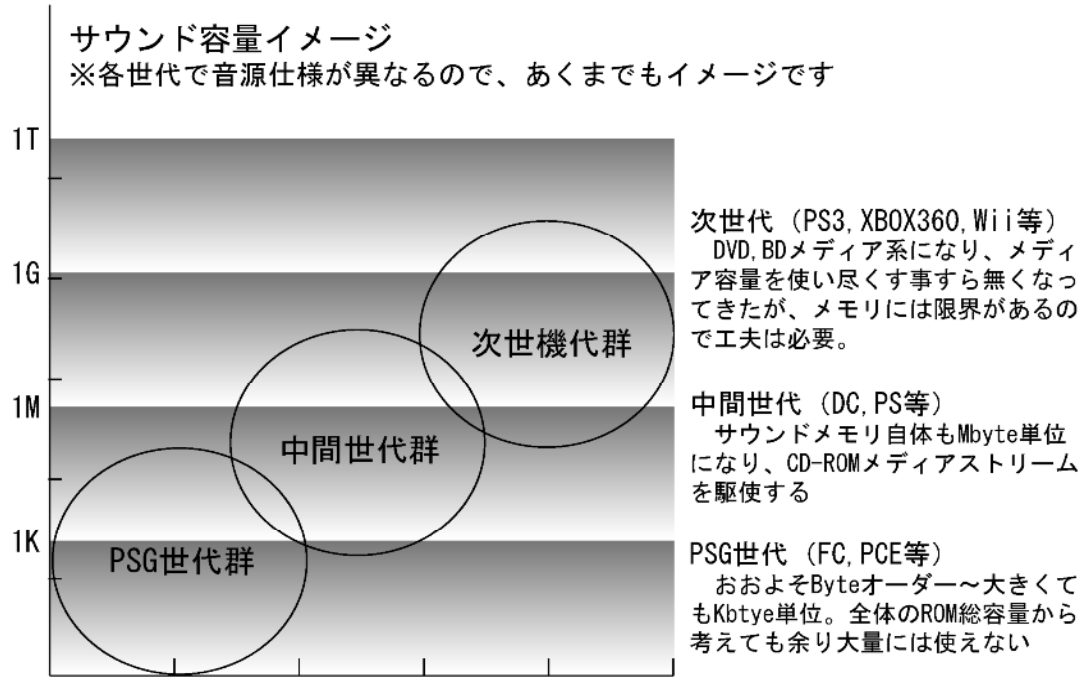


図 3.2-14　サウンド容量イメージ

(2)　新たな作業

メディア容量の拡大した事によってそれまでのゲームとは明らかに変わったところは、音声が多用されるようになった事と一般音楽と変わらない生楽器を使った音楽等が代表的なところだろう。今まで 1 つのゲームで 1 人、2 人でサウンド周りを制作していたのに、突如人手不足になる。もちろん時間さえあれば延べのマンパワーで制作も可能だが、基本的には色んな作業が増えて。音声を多用する為には、声優のコーディネートからレコーディング編集も必要になり、またそれをアウトソーシングするなら折衝能力が必要に。さらには 3D の概念をもった作業が必要になってきた。また生楽器を使ったレコーディングをするにも別のスキルが必要で、これを全てサウンドというセクションに含めるとしたら、1 つの作品に 10 名弱の人数が必要とも言える。もっとも、ハイエンドゲームとカジュアルゲームでは必要な要素自体が違うので、必須という訳ではない。（図 3.2-15：分業参照）

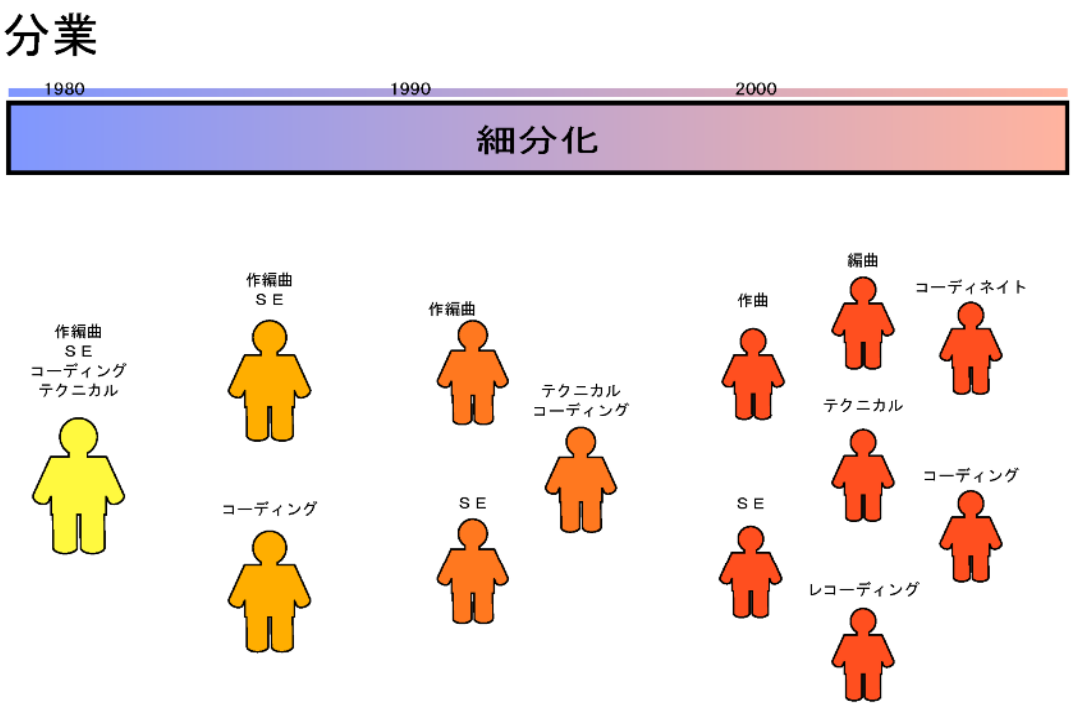


図 3.2-15　分業

- 作曲　音楽の根幹を作る
- 編曲　生演奏の為のアレンジや譜面（場合によってはアレンジャーの他にオーケストレーション専用のアレンジャーが必要）
- SE　サウンド・エフェクトの制作
- コーディング　実機に SE 等を仕込む
- サウンドテクニカルディレクター　ソフトウェア担当との間をとりもつ
- レコーディングエンジニア　声や楽器演奏を収録する
- コーディネーター　声優のアサイン、演奏者のアサイン、スタジオのアサイン等を行う

ゲームの音楽の行程はほぼ映画を作るのと似た構造にゲーム的なインタラクティブな部分を含めた人員構成が必要になってきた。物量の肥大化、分業化によるサウンド制作の方向性を定める為に音響監督的存在が必要になってくる。もちろん既にこのような構造になっている会社は海外では多数あるもっとも面倒になってくるのは、3D 的考えや必要性、アプローチ方法のノウハウになってくる。

(3) 多チャンネル化

モノラルからステレオ、5.1 チャンネル、7.1 チャンネルと多チャンネルが進んでいる。それと同時にスピーカー2 つのステレオ再生環境での疑似 3D 技術出力が進化してきた。多チャンネルか疑似 3D ではエンコードの方法だけの差ともいえるのであまり考慮しなくても良い。一般的な多チャンネル再生環境には多くの問題があり、リスニングポイントが非常に狭い事、設置環境を選んでしまう事や、配線自体の煩雑さ、設備の費用を考えると簡単にエンドユーザーの多数派にはなり得ない。疑似 3D なのか、3D なのかはデコードの問題なので、より高品位のデコーダーが作られれば良い。

3D に関しての多くの事は既にミドルウェア等のツールによりほぼ実現しており、あまり意識しなくても利用出来るようになってはいるが、サウンドというセクションだけでは実現出来ない事も増えてきている為にプログラマー、プランナー等の認知度を上げる必要があり、サウンドデザイナーとサウンドエンジニアとしてのコンビネーションは続くことになる。

また制作プロデューサーにも多少は理解して頂かないと、その為のコスト計算がされないままプロジェクトが進行してしまう事も考えられる。また現状のサウンド環境を含め理解している音響をまとめる人が必要になってくるだろう。

3D 映像世界では基本的に論理的な構造を基本とし管理上破綻しないようにスケールが保たれているが、音に関してはそうもいかず、1 メートルの距離で普通に聞こえる音量が、50 メートルはなれたら聞こえないという状況でも聞こえて欲しい場合がある。映像で例えるなら歪んだ世界、それが多々必要とされる音響演出を提案する為に色んな手を入れないと実現不可能な懸案が多数出てくるだろう。

(4) まとめ

視覚的な表現や効果に比べ、聴覚的な表現や効果はあまり重要視されず、ハード設計の時点から後回しになりがちだったり、ソフト予算配分的にも十分には与えられない事が多く実際にその効果も気がつかれなかったりする。

ただ、全てのゲームに於いて全てのサウンドがベストな状態でなければならないという訳でもないため、カジュアルなゲームのスタンスからハイエンドゲームのスタンスまで、それぞれの着地点を見極める為の人材や、その他のセクションと円滑にコミュニケーションをとる立場の人材が必須になってくるであろう。

3.2.4　素材作成　アニメーション

金久保　哲也
株式会社バンダイナムコゲームス

(1)　年代別ビデオゲーム アニメーションの変遷

ビデオゲームにおける 3DCG アニメーション表現はハードと共に進化しているとも言える。ここではハードの進化に沿って、ビデオゲームにおける 3DCG アニメーションの表現とそれらを実現するための技術の変遷を解説する

(a)　1980 年後半～1990 年前半　業務用ビデオゲームにおける 3DCG アニメーション表現

この年代の業務用ビデオゲームにおける 3DCG 表現は、ポリゴンの表示はフラットシェーディングの単色で、表示できるポリゴン数も限られていた。扱われるジャンルはレースゲームやシューティングゲームなどで、モデルのアニメーションはプログラムによってコントロールされていた。従来の 2D 表現と比較するとアニメーション、造形の緻密さで表現力は劣っていた。2D 表現が必要な部分は、従来のスプライトの手法を組み合わせる方法がとられていた。また、DCC ツールでレンダリングした画像をスプライトや背景に使用するといった方法で表現力の弱い部分を補っていた。

(b)　1990 年後半　家庭用ハードの登場

業務用ビデオゲームが高性能化した事により、より多くのポリゴンが表示できるようになり、テクスチャマッピング機能が実装された事で、細かなグラフィック表現が出来るようになった。アニメーション表現へリソースが割けるようになったのもこの時代からである。アニメーションの方法として、従来のプログラムによるコントロールに加えて、アーティストが DCC ツールでアニメーションデータを作成し、これを実機で再生する方法が加わった。また、ポリゴン表現が可能な家庭用ハードの登場で、多くのゲームデベロッパーが 3D ゲームの開発に着手したのもこの時代からである。これによって、ビデオゲームにおける 3DCG アニメーション表現が急速に実践されていくことになる。

(c)　2000 年前半　高性能な家庭用ハードの登場、ゲーム開発の大規模化傾向

家庭用ハード市場の勢いは衰える事無く、より高性能な家庭用ハードが登場する。ここに至っては家庭用市場と業務用市場の勢いは逆転し、ビデオゲーム開発は家庭用向けに注力される傾向が強くなった。業務用ビデオゲームと比較して家庭用ビデオゲームは、プレイ時間が長く、グラフィック、アニメーションなど大量のコンテンツが必要とされ、ゲー

ム開発の大規模化の要因となった。

また、ハードの高性能化によって、パーティクルや物理シミュレーションといった、それまで実現できなかった表現が可能となった。これらは煙や炎、物体の破壊表現などに使用されるようになった。

(d) 2000 年後半　HD 対応ハードの登場

2000 年後半に登場した第 7 世代ゲーム機では、解像度が HD に対応した事によってグラフィックのクオリティを大幅に向上させた。アニメーション表現には、より細かな演技が求められるようになった。2000 年前半から引き続き、作成するコンテンツ量は増加傾向にあることも加わり、大規模開発化の傾向が更に強まったと言える。一方で、それまでのハイエンド指向一辺倒だった方向性から、従来に無い新たな入力機器を主軸としたハードや携帯型ゲーム機のシェア拡大など、ゲーム開発の対象が多様化している。開発するハードに応じて、それまでに確立してきたアニメーション表現や作成方法も継続して使用されることになる

モーションキャプチャにおいては、従来の体の動きをキャプチャするだけでなく、顔や指の動きを同時に収録できるまで精度が向上した。リアルタイムでの物理演算が可能となり、映像的な演出からゲーム性に関わる部分まで様々な要素に使用されるようになり、今後より一層活用される場面が増えていく事が予想される。

(e) ビデオゲームにおけるアニメーション表現の今後

ビデオゲーム機の高性能化、HD 映像への移行によって、より細かなアニメーション表現が求められるようになっている。アニメーション作成技術の今後のトレンドとしては、シミュレーションを利用した不雑なアニメーションやモーションキャプチャで収録した顔や手なども含まれる大量のデータをアーティストがストレス無く、イメージ通りに操作、調整することが出来るアニメーション環境が求められる事が予想される。

また、近年注目されているプロシージャル技術やゲーム AI 技術はアニメーションと深く関連する技術として見逃せない。 北米を中心に支持されているオープンワールド型と呼ばれるゲームタイトルでは、世界そのものを細かいディティールで表現し、プレイヤーが行える行動に広い自由度を持たせて、世界観そのものを楽しむという遊び方を提示している。オープンワールド型のタイトルではキャラクターに多様性が求められる。従来の定型化したアニメーション表現だけでは多様性に限界があり、キャラクターの状態によってアニメーションを動的に調整するように技術が必要である。 また、AI によって表現されるキャラクターの知覚をどう視覚化するか、という点で従来以上の高度なアニメーションの制御技術が必要になる。こうした実機上でのアニメーション制御を考慮したアニメーション作成がアーティストに求められるだろう。

(2) キーフレームアニメーション

キーフレームアニメーションは 3DCG アニメーションにおいて、もっとも基本となるアニメーションテクニックである。一連の動きの中で特徴となるポーズをアーティストがモデルを変形、パラメーターを変更して作成する。これをキーフレームと言う。時間軸に沿って並べられたキーフレームとキーフレームの間は、コンピューターによって補間される。補間方法はアーティストがファンクションカーブを変更する事によって、キーフレーム間の挙動やスピードを調整することが出来る。

アーティストが DCC ツールを操作して作成するところから、「手付けアニメーション」とも呼ばれている。

キーフレームアニメーションは作成した DCC ツール上でしか再現することが出来ない。そこで、完成したキーフレームアニメーションは、データ出力時に連続したフレーム毎のデータに変換されて、ビデオゲーム機で再現される。1 部の例外を除いてゲーム・アニメーション作成において、どの作成技術、方法を使用した場合でも、ビデオゲーム機へ実装されるデータはフレーム毎のデータに変換されたものである。

(a) 移動、回転、拡大縮小のキーフレームアニメーション

これらの基本的なパラメーターのアニメーションは、プログラムによって制御される事が多いが、演出的な要素が強く、複雑な動きを表現する場合、アーティストがキーフレームアニメーションで作成する場合がある。直接、各モデルのパラメーターにアニメーションを設定する以外に、後述する DCC ツールの機能を組み合わせて、間接的にコントロールする場合があるが、最終的にはフレーム毎のオブジェクトのアニメーションとして出力される。

(b) モデル形状変形のキーフレームアニメーション

モデルの形状変形は、キーフレームとして同一のトポロジーで形状の異なるモデルを用意する。これらのモデルをブレンドする事によって、アニメーションを表現する。ブレンドのパラメータをフレーム毎に出力して、ビデオゲーム機でアニメーションを再現する。この手法はビデオゲーム機の計算コストが高いため、リソースを割く事が難しい時代には、フレーム毎に変形したモデルデータを出力して、これをフレーム毎に差し替える事によってアニメーションを表現した。この手法は短いフレーム数でサイクルするアニメーションで使用された。

(c) 頂点カラー、輝度情報のキーフレームアニメーション

初期の 3D ゲームでは、ハードの制約から固定されたライティング表現しか行うことが出来なかった。 環境のライティングの変化を表現するために、同一形状で頂点のカラーや輝度が異なるモデルデータを差し替える方法が使用された。この方法では、アーティストはライティングに対応した頂点カラーや輝度に変更したモデルを用意して、差し替えはプログラムによって制御された。

(3) ボーンアニメーション

ボーンアニメーションはキャラクタアニメーションに特化したアニメーション機能といえる。骨に見立てたボーンオブジェクトで階層構造を作成して、間接に見立てて回転させるアニメーションをボーンアニメーションという。関連する技術として、この各ボーンオブジェクトとキャラクタモデルの各部分の関連付けを行い、ボーンオブジェクトを回転することで、キャラクタモデルを変形させるスキン変形がある。

ボーンアニメーションにはアーティストによって作成されるアニメーションとプログラムによってコントロールされるアニメーションがある。

ボーンアニメーションの作成を支援する技術として、インバースキネマティクスとコンストレイントアニメーションがあるが、これらの技術を使用してアニメーションを作成した場合、ビデオゲーム機でアニメーション再現するためには、各ボーンオブジェクトのフレーム毎の回転データに変換して出力する必要がある。

(a) 変遷

ボーンアニメーションおよびそれに関連する技術は 1990 年後半の時点で DCC ツールに実装されており、キャラクター・アニメーションの作成の効率化を目的としてビデオゲーム開発でも使用されるようになった。インバース・キネマティクスと組み合わせて、アーティストがボーンオブジェクトの階層構造を考慮する事無く、キャラクターの四肢の動きを直感的に作成する事ができるようになった。この時代、ビデオゲーム機でスキン変形が表現できなかったため、関節毎に分解したリジットなオブジェクトに関節のアニメーションを割り当てて、キャラクタアニメーションを表現していた。

2000 年前半になり、ビデオゲーム機の高性能化によってスキン変形が可能になった。間接に継ぎ目の無い自然な造形でのキャラクター・アニメーションが表現できるようになったのに加えて、それまでリジットなオブジェクトでは表現できなかった、髪の毛や布、筋肉の盛り上がりなど、細かなアニメーションを表現できるようになった。顔のモデルにボーンオブジェクトを設定して、フェイシャルアニメーションを行った事例もある。

2000 年後半、第 7 世代のビデーゲーム機では HD 映像に対応するために、より細かな

表現でボーンアニメーションが使用されるようになった。 映像制作においては、筋肉モデルによる形状変形の表現など、 ボーンアニメーションに代わる新たなアニメーション技術が取り入れられてきているが、 第 7 世代の高性能ハードをもってしても、これらのアニメーション技術を実現する事が難しい。

(b) インバーキネマティクス

通常のボーンアニメーションでは関節の階層構造に従って親から子へ各関節の回転を指定していく。しかし、この方法では手や足の動きを制御するのが難しい。対して、インバースキネマティクスは関節の末端位置を指定する事で、その状態に至る各関節の回転を自動的に計算するため、足踏みをする、手で物をとるといったアニメーション表現を直感的にコントロールすることが出来る。

インバース・キネマティクスの機能はビデオゲーム機では 1 部の例外を除いて実装されていない。各 DCC ツールでのインバースキネマティクの挙動が微妙に異なり、同じ挙動をビデオゲーム機上で再現する事が難しい。DCC ツール上でインバース・キネマティクスで作成されたアニメーションは、各関節の回転データに変換されてビデオゲーム機で再現される。

プログラムによるインバースキネマティクスのコントロールの例として、地形の高低差に合わせて、足の位置を調整する表現があげられる。こうした処理を行う場合でも、アーティストがインバースキネマティクを利用して作成したアニメーションデータを各関節の回転データとして出力している場合が多い。

(c) コンストレイントアニメーション

インバースキネマティクスと同様にアーティストのボーンアニメーション作成の支援機能として活用されているものにコンストレイントアニメーションがあげられる。これは、ボーンオブジェクトの階層構造に依存しない、移動、回転を行う事ができる機能である。コントローラーとして、オブジェクトや Null ノードを用意して、ボーンオブジェクトをこれらに拘束する。類似する機能として、キーフレームとコントローラーを関連付けて、コントローラーを動かすだけで複雑なアニメーションを制御する機能もある。インバースキネマティクやコンストレイントアニメーションを組み合わせ、アーティストが直感的にキャラクターをコントロールできる環境を構築する事をリギングと言う。

(d) スキン変形

ボーンモデルとキャラクタモデルの各身体部分の関連付けを行い、ボーンオブジェクトを回転して、キャラクタモデルを変形する手法をスキン変形という。2009 年現在、ゲ

ーム・アニメーションにおいてキャラクターのアニメーションを表現する方法として最も多用されている手法である。顔の表情変化や布の表現などにも使用されている。これらのアニメーション表現には筋肉の伸縮を基に変形、またはクロスシミュレーションといった、より目的に即した手法があるが、ビデオゲーム機でこれらを再現する事が難しいため、代用の表現方法としてスキン変形が使用される事が多い。なお、2009 年の時点でクロスシミュレーションによる布の表現を行った事例もいくつかある。

(e) フルボディ IK

インバース・キネマティクスはあらかじめ設定したルートと末端間の関節の回転を計算する事しかできない。対して、フルボディ IK は、ルートと末端となる関節を自由に変更することが出来る。例えば、人が物をつかもうとする場合、まず、つかむ物に手を延ばし、それでも届かない場合は上半身を曲げて、手が届くようにしようとする。一般的な腕のインバース・キネマティクスの設定では、このような表現を行う場合、最初の腕を延ばす動きは、手の末端を移動してインバース・キネマティクスでコントロールし、上半身を曲げる動きは、別途設定した上半身のコントローラーをタイミングをとりながらコントロールすることになるが、フルボディ IK では腰のジョイントをルートに設定して、腕の末端のジョイントをコントロールするだけである。フルボディ IK もインバース・キネマティクス同様にプログラムによるコントロールも可能である。

(4) テクスチャアニメーション

テクスチャアニメーションは文字通り、テクスチャを動かす事によって表現されるアニメーションである。アーティストはペイントソフトで手描きや、実写素材を編集するなどしてテクスチャデータを作成する。複雑な表現では、DCC ツールを使用してレンダリングして作成する場合もある。

テクスチャデータはテクスチャの解像度を大きくしたり枚数を多くしたりする事で、グラフィック表現が緻密になるが、ビデオゲーム機が高性能化してきた 2009 年現在でも、アーティストの望むだけのリソースを割くことは難しく、限られたリソースを有効活用するために、アーティストが様々な工夫を凝らしているのが実情である。

テクスチャアニメーションはスクリプトでコントロールされる場合が多い。例えば、小さな炎のテクスチャデータをモノクロで作成して、異なる大きさや移動スピード、色の変化などの組み合わせをスクリプトでコントロールする事で、複数のバリエーションを少ないリソースで表現することが出来る。こうした手法によるアニメーション表現は「エフェクト」と呼ばれている。エフェクトはテクスチャの作成とスクリプト作成という、アートとプログラミングの 2 つのスキルが求められ、ゲーム開発においては 1 つの職種とし

て確立している。

(a) 変遷

テクスチャアニメーションは 1990 年後半、ビデオゲーム機でテクスチャマッピングの機能がサポートされてから 2009 年の現在まで、様々な表現に使用されている。

初期のビデオゲーム機ではテクスチャの UV 座標を変化させる UV スクロールは座標の数値変化をプログラムでコントロールする事で実現するため、少ないリソースでアニメーションを表現する事ができる有効な手段であった。電光表示板や川の流れなど動きの方向性がはっきりしたアニメーション表現に使用された。

2000 年前半になり、ビデオゲーム機の高性能化によって、1 枚のテクスチャを UV スクロールするだけではなく、テクスチャデータをフレーム毎に差し替えてアニメーションを表現する事も可能になる。

パーティクルを併用した爆発や炎の表現など、ゲームの世界に臨場感を加える有効な表現手法として、活用されるようになった。

2000 年後半、第 7 世代のビデオゲーム機においては、ディスプレースメントマップの機能を利用して、形状のディティールをアニメーション表現する事も可能となった。炎や煙、木漏れ日といった、ボリュームレンダリングの表現は 2009 年現在でも、ビデオゲーム機でレンダリング表現する事が困難であるため、パーティクルアニメーションとテクスチャアニメーションを併用した表現方法が利用されている。

(b) UV スクロール

テクスチャの UV 座標を動的に変化させる事で、オブジェクトに貼付けたテクスチャがスクロールする。形状変化のアニメーションやイメージの加算、減算合成と組み合わせる事で、 川の流れや木漏れ日などの表現に使用される。UV スクロールは 1 枚のテクスチャを U 方向、V 方向に移動する表現しかできないので、それ以上に複雑なテクスチャのアニメーションを行う場合は、後述する連番テクスチャ、ムービーテクスチャを使用する。

(c) 連番テクスチャ、ムービーテクスチャ

複数のテクスチャをフレーム毎に差し替えたり、テクスチャデータにムービーファイルを使用したりする事によって表現されるアニメーション。 爆発や煙、炎など刻々と状態が変化する現象を表現する場合に使用される。また、ゲーム中の状況を小さなイメージでレンダリングして、これをテクスチャ素材としてゲーム画面に中継モニター的な表現を行うと行った、プログラムによるテクスチャアニメーションの事例もある。

(d) ディスプレースメントマップのアニメーション

ディスプレースメントマップはテクスチャマップの明暗によってオブジェクト表面の法線情報を変化させる手法で、モデリングでは表現が難しい細かなディティールを表現するのに適している。この機能を利用したアニメーション表現として、キャラクターの顔の局部的な変形に伴う皺の表現があげられる。アーティストは皺の出ていない平常の状態と皺が最も出ている状態のディスプレースメントマップを作成する。プログラムコントロールでこれを対応する顔のボーンアニメーションの変化量に対応させてブレンドすることで表現される。同様な方法で、インタラクティブに水面に発生する波紋を表現した事例もある。

(5) 数理アニメーション

数理アニメーションとは計算式によって表現されるアニメーションである。広義に解釈すると、プログラムで表現されているアニメーションは全て数理アニメーションと言えるが、ここでは、アーティストが数式やパラメータをコントロールする事によって表現されたアニメーションに限定して解説する。

(a) 変遷

2000 年前半にアーティストがコントロールする数理アニメーションの環境が登場する。ビデオゲーム機が高性能化する事で、プログラムで数理アニメーションを作成し、煙や炎の表現は技術面では実現可能となったが「絵作り」という面でアーティストがコントロールに関与する必要性があった。そのため、アーティストがスクリプトでパラメータをコントロールするパイプラインが構築された。

2000 年後半、第 7 世代のビデオゲーム機では、オブジェクトの破壊表現にリジットダイナミクスが使用されるようになった。また、映像表現の手法以外に、リジットダイナミクスや流体シミュレーションを使用したパズルゲームが開発されており、今後はゲームデザインの要素に活用される可能性も考えられる。

(b) パーティクル

パーティクルは煙や炎などエフェクト表現の手法として使用されている。粒子に対して重力や質量、加速度などのパラメータを設定して、アニメーションをコントロールする。パーティクル自体は古くから DCC ツールの機能としてサポートされていたが、ビデオゲーム機上で表現が可能になったのは、2000 年に入って、第 6 世代のハードからである。それ以前では、DCC ツールで表現されたパーティクルをフレーム毎の移動と回転のアニ

メーションやオブジェクトとして出力して使用した事例がある。現在アーティストはテクスチャ素材を DCC ツールで作成して、粒子のコントロールはスクリプトでコントロールする場合が殆どである。映像制作ではパーティクルとボリュームレンダリングによって煙や炎をリアリスティックに表現しているが、ビデオゲーム機で同様の手法を行うにはリソースを多く割くことから、テクスチャアニメーションと併用した表現が利用されている。

(c) リジットダイナミクス

リジットダイナミクスはオブジェクトの破壊表現に使用される事が多い。破壊するオブジェクトをアニメーションさせるパーツに分割して、外力や質量、摩擦、衝突時のパラメータを設定して、アニメーションをコントロールする。オブジェクトにコリジョン（衝突判定）を設定して、オブジェクト同士が衝突してはじかれたり、形状を考慮して地面を転がったりなど、自然で複雑なアニメーションを自動生成することが出来る。

アーティストは DCC ツールでリジットダイナミクスの表現を行うパーツ単位にオブジェクトを分割して、パーツのレイアウトを行う。シミュレーションに関連するコントロールはスクリプトで行う。ビデオゲーム機でリジットダイナミクスは実現されているが、大規模、大量のオブジェクトをシミュレートするといった、スケールの大きな表現を行う事が難しい。この場合、DCC ツール上でリジットダイナミクスを行い、シミュレーション結果をパーツ毎のアニメーションデータとして出力している。

将来的には全てのリジットダイナミクスによる表現が、ビデオゲーム機で再現できるようになる事が予想されるが、例えば破壊の表現を行う場合、アーティストがあらかじめ破壊可能な部位を、CDD ツールで作成しておく必要がある。こうした DCC ツール上での事前のデータ作成のプロセスの効率化、最終的には動的に破壊パーツを自動生成するような技術が課題となるだろう。

(6) モーションキャプチャ

モーションキャプチャとは実空間における運動情報を数値化する技術、人体や動物の主要な関節部分にセンサーをつけて、 運動データをコンピューターに取り込む技術である。ビデオゲーム開発においては、主に人型のキャラクターのアニメーションを効率的に大量に作成するために活用されている。

モーションキャプチャを利用する事で、アニメーション作成がなくなると思われがちだが、モーションキャプチャで収録したデータは、キャラクタアニメーションの素材でしかありえない。ビデオゲームはプレイヤーのインタラクションによって映像が変化するものであり、このインタラクションに違和感無く追従するアニメーションデータが必要となる。総じてビデオゲームにおける動きは誇張されたもので、収録したデータを誇張表現す

る必要がある。

ビデオゲームのアニメーション作成におけるモーションキャプチャの利点は、複雑な人体の動きを参照しながらも、ゲーム・アニメーションとしての調整ができるという点と言える。

(a) 変遷

1990 年前半からモーションキャプチャは、ビデオゲーム開発で利用されるようになった。3 次元のキャラクターのアニメーションを手付けで作成することは時間のかかる作業で、モーションキャプチャによる作業の効率化は効果が大きかったと言える。磁気式や機械式など様々な方式のモーションキャプチャシステムが登場したが、現在ではデータ収録の自由度や安定性から光学式モーションキャプチャが使用されている場合が殆どである。

2000 年前半以降、ビデオゲーム開発の中心が家庭用市場に移行すると、モーションキャプチャの需要が増加した。家庭用ゲームでは状況説明やストーリー進行の役割を果たすカット・シーンが多く取り入れられているためである。セットを組んでの収録や複数人数の同時収録、ワイヤーアックションなど、複雑な状況でのモーションキャプチャ収録にも対応する必要が発生した。

2000 年後半、第 7 世代のゲーム機リリース以降では、より細かな演技をモーションキャプチャで収録するニーズが高まった。HD 画像では従来のキャラクター表現では十分とは言えず、より細かなアニメーション表現が求められるようになったのである。また、従来のプリレンダリングによるカット・シーンの作成が HD 画像に移行した事でコストが高くなった事で、リアルタイムによるカット・シーンへ移行する傾向が強くなった。モーションキャプチャの利用方法は、映像制作における利用方法と殆ど変わらない状況になりつつある。

(b) フェイシャルキャプチャ

モーションキャプチャ技術の進化でマーカーが小型化した事により、小さなマーカーを顔に付けて、顔の動きをモーションキャプチャで収録することが出来るようになった。これをフェイシャルキャプチャという。ビデオゲームでフェイシャルキャプチャが活用されるようになったのは、2000 年以降である。技術的には既に確立しているが、今後の課題として収録した顔のアニメーションデータを効率的に編集できる環境の構築が重要となる。

顔の動きは体や手の動きと異なり、筋肉の伸縮と各部位同士の連動した動きによって成り立っている。フェイシャルキャプチャのデータは各部位のコントロールポイントが個別に動いている。ちょっとした動きの調整でも、複数のコントロールポイントを連動した動きになるように調整しなければならない。顔の動きと同じ挙動で各コントロールポイン

トが干渉しながら動くフェイシャルアニメーションシステムが必要なのである。

(c) パフォーマンスキャプチャ

全身、フェイシャルに加えて指の動きを一度に取り込む方法をパフォーマンスキャプチャという。この方法はフル CG の映像制作において活用さている。前述したフェイシャルキャプチャの問題として、体と顔の演技を別々に収録するためにタイミングがずれてしまい、これを調整する作業が難しく時間のかかる作業であるという事があげられる。アーティストからは連動した全身の動きを 1 回でデータ収録したいというニーズが強く、今後はカット・シーンを中心にパフォーマンスキャプチャを活用されることが予想される。

(d) リップシンク

リップシンクとは、収録したキャラクタボイスを音声解析して、各音素のブレンド率で音声と同期した口のアニメーションデータを作成する方法である。役者の演技からデータを収録する方法として、モーションキャプチャと並んで多く活用されている手法として、ここで紹介する。

モデルデータをブレンドする方法とボーンアニメーションによる方法がある。モデルデータをブレンドする場合は、各音素に対応した口の形状のモデルを用意する。ボーンアニメーションの場合は、各音素に対応したボーンのキーフレームポーズを用意して、これをブレンドする。音声解析からブレンドまで一連のアニメーション作成はオフラインで行われ、ブレンド率やボーンのアニメーションを出力してビデオゲーム機で再現する。
ブレンドする音素の数が多いほど、自由度の高い編集環境で、自然な口の演技を作成することが出来るが、リソースを多く割かなければならない。モデルデータをブレンドする方法はリソース量が多くなるため、自由度の高いリップシンクアニメーションが必要な場合、ボーンアニメーションによる方法を使用する場合が多い。

3.2.5　プログラミング　AI

三宅　陽一郎
株式会社フロム・ソフトウェア

(1) はじめに

デジタルゲーム AI の資料が充実して来るのは、主に米の GDC において 2000 年以降であり、それ以前は、Web 記事や雑誌インタビューなど、断片的な記事が存在するに過ぎない。また、特に日本においては情報の公開は、この数年の CEDEC や IGDA 日本（国際ゲーム開発者協会　日本支部）の活動を通して活性化しているが、それまでの長い期間に渡って技術情報公開についてゲーム業界が閉鎖的であったために、利用できる公式の文書も少ない。また企業内の開発者でさえ、自社か自身が担当したタイトル以外把握できない、或いは、開発文書を作成し保存するという文化を確立して来なかった日本の開発の現場では、そのタイトルに従事した開発者でさえ、細かいパラメーターの全体を開発後には追えないという事態になっていることも決して珍しいことではない。このような状況において、ゲーム AI の歴史を編纂するということは非常に困難な仕事であり、実は、これはゲーム AI のみならず、あらゆるゲーム開発技術において同様の状況であることを、最初に言及しておかなければならない。つまり、ゲーム開発技術分野全体に渡って情報の整備と歴史の掘り起こしが必要なのである。日本デジタルゲーム学会(DiGRA JAPAN)やデジタルコンテンツ協会の本報告書によって、ベテラン開発者へのヒアリングや講演会の実施などの活動が行われ始めているものの、まだまだ情報が不足しているのが現状である。また、学術的にゲーム開発技術の歴史を調査しようとする運動もまだ少なく、また産業側にも情報公開に対する壁が存在するために難しいケースが多い。つまり、ゲーム開発技術の編纂と公開は、ゲーム産業自体が立ち上がって解決せねばならない課題である。

まず本報告書は、公開された数少ない情報から、ゲーム AI の歴史を再構成した報告書であることを了解頂きたい。そういった意味で、ゲーム AI の歴史、いや、他の技術分野の歴史は、過去のゲーム開発の調査をくり返しながら、毎年、更新されるべきであって、また、更新されるように毎年新しい成果と調査を行わねばならない。それは他の歴史研究と同様である。同時に、現在においても、日本のゲーム AI の情報は非常に少ないのであるから、時を経れば、同様の問題が、現代という時代に対して「情報が少ない」という事態が発生するのは間違いない。ゲーム開発技術の情報は、「公開か、非公開か」、という稚拙な議論ではなく、「何を公開し、何を秘匿とするか」、という知識運用（ナレッジ・マネージメント）の問題意識が必要である。ある程度の情報は公開し業界全体で共有することで、ゲーム産業全体としての技術レベルを上げ、お互いがお互いに恩恵に預かることが出来るはずである。

(2) ゲーム AI とは何か？

デジタルゲーム AI を正確に定義することは難しい問題である。それは、一般に人工知能という学問においてさえ、知性の定義が難しいのと同様である。そこでまず、ここで 2 つの例を提示してから、ゲーム AI の定義を行うことにする。

敵が自分に向かって弾を撃って来る 2D シューティング・ゲームを考えることにする。そういった敵は、単に軌道と弾を撃つタイミングの関数が埋め込まれているだけであるが、「敵が弾を撃ってくる」という事実によって、プレイヤーの心理は、その敵を「敵意ある AI」として認識することになる。もし、ステージさえ小さければ、簡単な動作プログラムでも十分にそういった感情をプレイヤーに喚起するに十分であり、このような機能を持つ敵をゲームでは AI と見做す。つまり、デジタルゲームにおける AI は、ユーザーがゲームプレイ体験を通じて知性と感じることが出来れば、それを AI と呼ぶ。これはデジタルゲームがエンターテインメントであることを反映している。ここから更に、高尚な AI であるか、そうでないかは、積み重ねる動作によって変化することになる。

また、近年の 3D 空間の FPS である HALO（Bungie）[1]を考えてみる。HALO は、3D 空間で銃を持って敵を迎撃するゲームである。そういった空間における敵 AI は、2D の簡単なゲーム以上に複雑になったステージに対して、パス検索、状態遷移マシン、知識表現など AI の技術を駆使してステージの状況を理解しながら行動する必要がある。ただ、そういった現代的なゲームになっても、最終的な目標は変ることなく、プレイヤーに己を知的な存在に見せかけるというところにある。また、さらに、発展した目標としては、そういった知性を通じて、戦場全体に渡る AI の群れの知性を演出する、という目標もある。そこで、デジタルゲーム AI を、以下のように定義する。

デジタルゲームにおける AI とは、ユーザーがそこに知性を感じさせる存在のことを言う。

「そこに知性を感じさせる存在」は、*illusion of intelligence*（知性の幻影）と呼ばれることもあり、デジタルゲームにおける AI は、実装技術によってではなく、極めて主観的な体験として定義される。このような定義は、「一般的な人工知能技術をゲームに埋め込んだものを AI と呼ぶのか」、或いは、「中身はただの条件分岐のプログラムだが、外見はキャラクターの形をしているオブジェクトを AI と呼ぶのか」、「台詞を喋れば AI なのか」、という問題に触れることなしにゲーム AI を定義付ける実効的な定義である。

一方で、この定義はゲーム AI が技術にこだわらないということを意味しない。「Illusion of intelligence （知性の幻影）」の感覚をユーザーに与えるために、学術的な AI 技術を適用することも一つの方法であれば、単なるプログラムを演出によって知性と見せかけることも一つの方法である。前者は、主に欧米において、リアリティを求めるゲ

ーム開発と、学術的な AI 技術をマッチさせることで発展を遂げて来た方向である。欧米のゲームは、そもそも学術的なコンピューター・サイエンスから発祥したこともあり、ゲーム AI は、一般の人工知能技術を吸収して、リアリティのある AI が探求されて来たのである。一方、日本においては、主に、ゲームそのものがリアリティではなく物語であり、特殊な世界であり、ファンタジーであり、技術である前にメディアであり、シチュエーションによって知性と見えればよい、という方向の AI を主に探求して来た。そのため、技術としては、欧米に非常な遅れを取っているものの、日本独自のノウハウを、日本独自のコンテンツに結び付けて育んで来た。しかし、そういったノウハウも公開されることは少なく、また、海外などを含めて普遍的な意味では理解しにくいため、国際的に存在感を持っているとは言えない。結果として欧米と日本技術の差が残ったと言える。

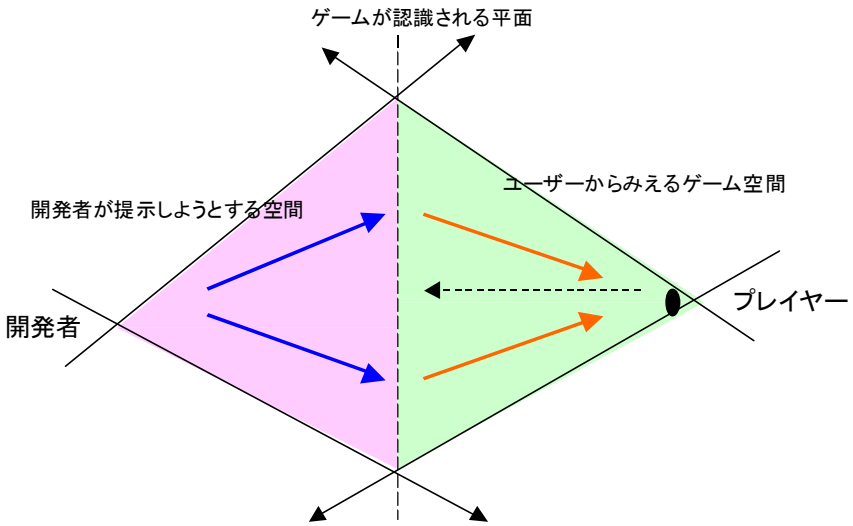


図 3.2-16　デジタルゲームにおけるプレイヤーと開発者の関係図

左から開発者が展開するゲームと、右からプレイヤーが知覚する世界が、中心で交差する。中央の境界は開発者がプレイヤーに見せることを前提にしている平面であり、プレイヤーが実際に認識するゲーム平面。ゲーム開発者が AI を作るというときには、実は AI そのものではなく、プレイヤーの体験を AI を含んでどう構成するか、ということを第一に考えている。その点がエンターテインメントとしての AI であり、研究で AI そのものを作る、ということと本質的に異なる点である。

開発者とプレイヤーはゲームを通して対峙する関係にある。開発者はゲームの仕組みまで含めて改善し知り尽くしているが、プレイヤーが受け取るのは、プレイヤーに感知される部分のみであり、その向こうの内部ダイナミクスは明かされることはない。そういったプレイヤーがゲームを認識する平面において、AI は活動し、ゲーム世界を背景として、プレイヤーはそれを知性と感じるのである。つまり、デジタルゲームの AI においては、その内部構造自身が AI である理由とは決してならないのである。

また、ユーザーに対して主観的な体験を与えるからと言って、そういった実装が直接に知性を創作するものばかりではない。例えば、ユーザーのスキルを解析して、ゲームレベルをコントロールするなど、長時間プレイして初めて気付く AI もゲームには埋め込まれていることもある。

さらに、上記の定義は、群集制御、プランニング、推論機構など、単に知能をシミュレーションする人工知能技術を知っているからと言って、デジタルゲーム AI を知っているとは言えないことも意味している。如何なる技術であれ何らかの意味で、ゲーム内で知的な効果を生み出さない限り、デジタルゲーム AI とは言われることはない。

例えば、群集制御で制御した鳥を登場させるとする。それは、一般に知られているように俯瞰視点から見れば確かに群集に見える[2]。しかし、それを主観的に見た場合、視界の狭いゲームでは鳥が集まって飛んでいるだけで、それが、集団として制御されていることを実感することは難しい。俯瞰的にシミュレーションを見ることと、主観的体験としてゲームを構築することは、全く異なる事象なのである。ここにゲーム AI としての課題が始まるのである。即ち、どうすれば、自分の作っているゲームで群集制御を見せることが出来るのか？　どうすればユーザーの主観的世界にある効果を実現することが出来るのか？ という課題である。

このように、ゲーム内の効果によってしか資格付け（qualify）されないゲーム AI の定義は、学術的研究から見れば、かなり偏狭な定義と見えるかもしれない。この、反対の立場として、一般に人工知能技術と言われるものを導入すれば、それはゲーム AI である、という定義が考えられる。しかし、よくよく考えてみれば、人工知能という学問自体が非常に総合的名学問であって、一つ一つの技術は単なるアルゴリズムや最適化手法である場合が多い（遺伝的アルゴリズムやニューラルネットワークは最適化手法の一つである）。

デジタルゲーム AI が、デジタルゲーム AI として自律した学問として成立するために、現在は、上記のような「実用的な定義」を採用しておくのが都合がよい。デジタルゲーム AI は、決して他の分野に従属した分野でもなく、また、非常に近い分野としてロボット AI の技術をそのまま持って来られるものでもなく（ヒントにはなる）自律的分野であり、また、そうであることを発展と共に示して行く必要がある。デジタルゲーム AI の研究が進めば、より的確な定義がされる日も来るだろう。だが、そのためには多くの研究の積み重ねを待たねばならない。それまでは上記の定義によって、デジタルゲーム AI と他の分野を明確に区別し、デジタルゲーム AI 固有の研究の内容を集積する指標とする必要がある。

以下では、上記の定義に従って、ゲーム AI としてどのような分野があるかを概観しながら、これまでの歴史と発展を描いて行くことにする。また、それぞれの引用した事項については引用先を章の最後にリスト化してある。解説のため簡単化して説明しているので、是非、必要な文献は直接読まれることをお薦めする。

(3) ゲーム AI の分類

ゲーム AI はゲームに対して相対的に形成されるものである。それは人間や動物が、がこの地上という環境に対して相対的に形成されていることと動揺である。そこで、ゲームのジャンルによってゲーム AI を分類するというアプローチがある。AI から見たもっとも大きなゲームジャンルの分類は、ユーザーとゲーム世界の関係から導かれる。即ち、「対話型ゲーム」と「没世界的なゲーム」に分けることが出来る（注：どちらも一般的な用語ではない)。

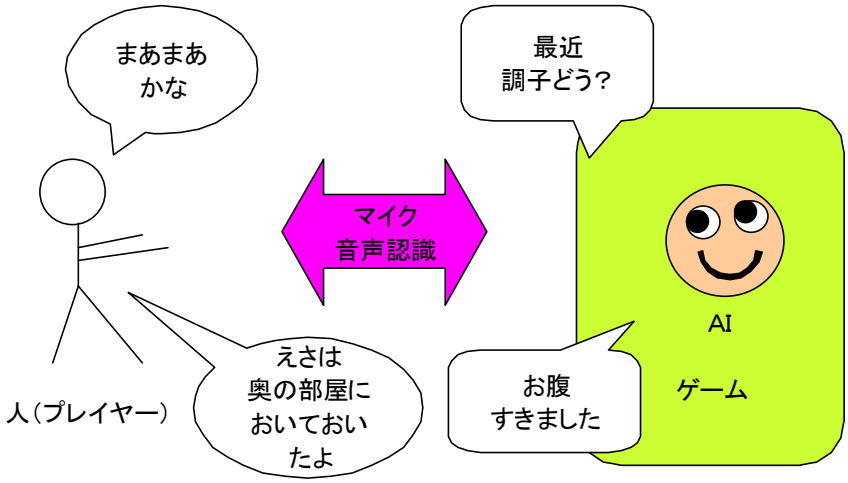


図 3.2-17　対話型ゲーム

プレイヤーはスクリーン内の AI と対峙して会話する。つまり、そこにある「AI」と自分自身が対峙する。

「対話型ゲーム」とは、ゲーム内のキャラクターが、プレイヤーに対して話しかけたり、プレイヤーがコントローラーやマイクを通してキャラクターに指示を出したりするゲームのことである。代表的な例として、「シーマン」（ビバリウム、1999 年）[3]や、「N.U.D.E.@ Natural Ultimate Digital Experiment」（レッド・エンタテインメント、2003 年）[4]、「オペレーターズサイド」（ソニー・コンピュータエンタテインメント、2003 念）[5]が挙げられる。ゲーム世界が、プレイヤーのいる世界から延長された空間であるという設定であり、音声など人と人のコミュニケーションと同じ手段で、プレイヤーが AI に指示を出す。こういったゲームにいて、AI はプレイヤーに直接話しかけ、プレイヤーの声を聞く一個の人格的知性として存在する。ただ、こういったゲームは高度な音声認識技術などを必要とするためか非常に数が少ない。

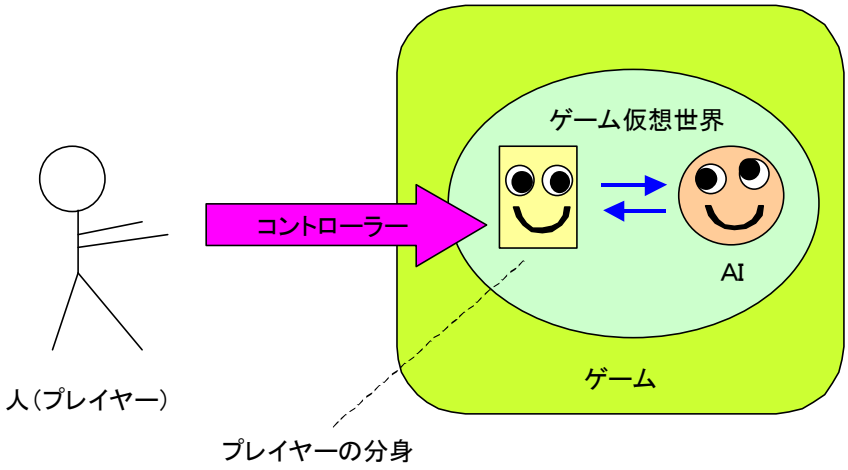


図 3.2-18　没世界的ゲーム

プレイヤーはゲーム世界の中に自らの分身を持ち、その分身を通してゲーム世界内の AI,オブジェクト、環境と相互作用する。

一方、「没世界的なゲーム」というのは上記以外のゲームで、プレイヤーがゲーム世界で、自分のキャラクターを持ち、完全にゲーム内で定義された身体と立場の上でプレイをするゲームである。こういったゲームにおける AI は、二つあり、主にゲーム全体をコントロールする「メタ AI」[6]（はっきりとした定義された言葉はない。筆者は「システム AI」と呼ぶこともある[7]）、とキャラクターを操作する「キャラクターAI」である。キャラクターAI はゲーム AI の全分野の中でも主要な部分を占める分野であり、所謂敵 AI である。メタ AI は歴史も長いがあまり目立つことがなかったが、近年、複雑化するゲームシステムの難易度や面白さを調整する AI として注目を集めている。

次章からキャラクターAI について解説し、その次の章でメタ AI について解説することとする。

(4) キャラクターAI

キャラクターAI を定義する要素は「環境」「身体」「知性」である。これは全く、我々自身である人間の知性と同様である。我々は環境世界の中に息づいており、身体を守り、知性によって行動を決めながら日々を生き延びている。知性は身体によって環境へ影響を及ぼし、環境から情報を取得して意思決定を行う。ただ、キャラクターAI が違うのは、人間が地上の環境に対して形成された存在であるのと同様に、キャラクターAI が「デジタルで造られた仮想空間の中に息づく知性である」という点である。人間が地上に適用するように形成された知性であるのと同様に、キャラクターAI は、仮想世界に対して相対

的に形成された知性なのである。「世界を知覚し、判断し、身体を正確に動かし、環境世界に影響を及ぼし、さらにもう一度その影響を与えた世界を知覚して行動を決定する」、という高度なキャラクターAI は、この 4，5 年になってようやく一つの形としてデジタルゲームの技術として定着した。後述するが、この実装フレームを「エージェント・アーキテクチャ」という。しかし、こういったアーキテクチャの指向へ至るまで、デジタルゲームの黎明期の原始的な AI から 30 年に渡る発展が必要であった。ここでは、その歴史的変遷を解説する。

キャラクターAI の特徴は、それが運動する身体とそれを操作する知性を持ち、身体を通じてゲーム世界に影響を及ぼすところである（接身体を持たない戦略ゲームも、意思決定を通じて環境世界に影響を及ぼすことは同様であるので抽象的な意味では身体であり、また RPG の魔法も身体の延長と捉えることも出来る）。こ 3 つの要素を通して、キャラクターAI の本質を彫り起こすことが、本章の目的である。この解説を読まれる方には、取り上げる例があまりにも偏っていると判断される方もおられるかもしれない。ただ、紙面は限られており時間は有限である。本解説の目的は全てのゲームの AI を取り上げ尽くすことではなく、全ての AI の歴史を包括的に捉える視点を提供することにある。その視点が、上記で挙げた「知性」「身体」「環境」から知能というものを捉える観点であり、実はこの観点は現代の AI では実装フレームそのものとなっている。

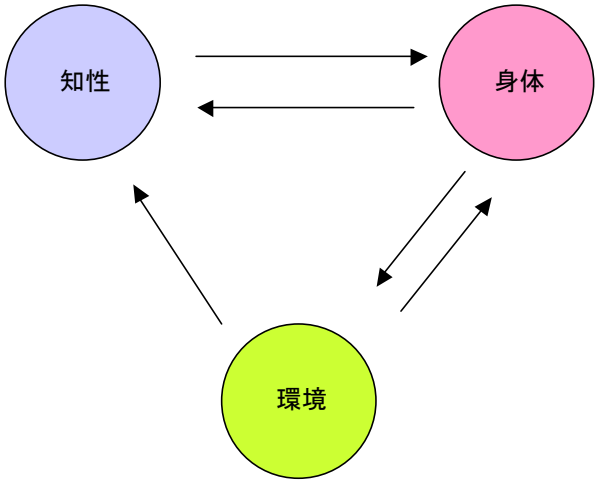


図 3.2-19　キャラクターAI の 3 つの主要な要素

「知性」はキャラクターAI の AI そのもの。身体はキャラクターの身体、直接身体がない戦略ゲームなどでは、自身が持つ手駒全体などを指す。「環境」は、ここではゲーム世界のステージのこと。この三者の相互作用系を創造することが、キャラクターAI を製作することである [8]。

人間という知性体である我々は、知性が身体にフィットしており、身体がどのように環境に影響を及ぼすかをあらかじめ知っており、5 つの感覚によってどのように環境から

情報を収集するかを知っている。人間は長い進化の過程で、「環境」「知性」「身体」の精妙な結び付きの中で、高度な活動を育んで来た。しかし、人工知能を作るということは、そういった普段あたり前だと思っていることが全くない状態から、この三者の間の関係を築いて行くことである。この三者の関係を自明なものとせず、零から出発することが、人工知能を製作する者に、知能というものの深い知見を与える。例えば、普段、我々は知能があって身体を制御していると思っている。そして思考は自由であり、身体はそれに従属しているように思える。しかし、それを逆に考えることも可能だ。身体こそが、我々の思考を制限し規定しているのかもしれない[9]。例えば、我々は常に自分の身体のスケールに合わせて物事を考えるし、進化の歴史を考えてみれば、知性の主たる役割とは身体をうまく動かして環境に適応することであったはずだ。身体というのは、長い年月をかけて、我々の考え方に深い影響を与えて来たはずである。とすれば、キャラクターAI の知能を作るときに、いきなり抽象的な理論から入るよりも、まずキャラクターの身体動作の特徴を抽出したデータを構築し、用意されたアニメーションの特性を理解し、そのような身体パラメータを前提に思考を形成して行くという方法もあるはずである。実際、キャラクターAI は、今、そういった方向に進みつつあるのである。また、身体と環境のインタラクションから知性を構築して行く方法は、人工知能の有名な研究でサブサンプション構造と言われている[10]。これも、身体から知性を形作っていく方法である。

以下、「身体－知性－環境」という視点から、それぞれの時代の AI を捉えて行く。

（注 1）ゲーム AI という分野は、新しい手法が導入されたからと言って、これまでの手法が完全に打ち捨てられるわけではない。新しい手法が重ねられる一方で、ゲームの大きさやキャラクターの種類によって、従来の手法が引き継がれている。例えば、有限状態マシンによる AI の実装方法がトレンドになっても、以前の条件分岐による実装はそのまま残っている。そういった視点から、以後に紹介する 3 つの時期は、それぞれの手法の始まりの時期を示しているのであって、それ以前の方法が廃棄されたわけではない。もし、現在の全ての AI の実装を見ることが出来たら、それこそ、歴史的な実装の全てを博覧することが出来るだろう。ゲーム AI の系統発生の歴史は現時点の個体 AI の発生に深く投影されているのである。

（注 2）AI の実装の情報は、最初に述べたように、2000 年前後まで、表に現れることは少なかった。出来れば、全てのゲーム AI のコードを見て、ゲーム AI の「本当の歴史」なるものを捉えてみたいと思うが、それは出来ない。結局、表に出ている有名なタイトルの AI 技術という点をつないで「ゲーム AI の歴史」を再構築する、というのが、現段階における精一杯のことである。また、CG と違って AI は、プレイしてみないと AI の良さはわからない。また、かなりやり込まないとわからない AI もある。つまり、ゲーム AI でよい仕事をしても、なかなかアピールされる機会は少ない。一方で、80 年代は、人工

知能自体が世間で持てはやされた時代であり、ゲームの宣伝上、普通の AI に、「なんとか AI」という名前をつけて誇大宣伝が為された。現代でも、その名残はある。ところが、そういった AI がまかり通るほどに、AI は抽象的である。ゲーム AI のよい仕事は、須らく評価されるべきである。そのためには、ユーザーや開発者が、きちんと AI の良さを感じ、AI というものに対する感覚を持たねばならない。そして、それは同時に、デジタルゲームの良さを味わう感覚を一層高めることでもあるのである。

(a) 第一期 単純なパターン AI

① くり返しパターンの AI

デジタルゲームにおいて、1970 年代後半から始まる最初のキャラクターAI の作り方は、ある行動パターン（ビヘビア）をくり返す、という手法であった。つまり、プレイヤーの位置や行動に関係なく、決められたパターンをくり返す AI であった。例えば、右に 2 歩、弾を撃ち、左に 2 歩、弾を撃つ、或いは、シューティング・ゲームであれば、回転しながら弾を撃ちフェードアウト、などである。つまり、こういったキャラクターは、AI というより、ゲームの舞台装置の一部として動作していた。RPG などでも、街に入れば、決められたパターンの動きと台詞しか言わないキャラクターが右往左往しているのを見ることが出来るだろう。

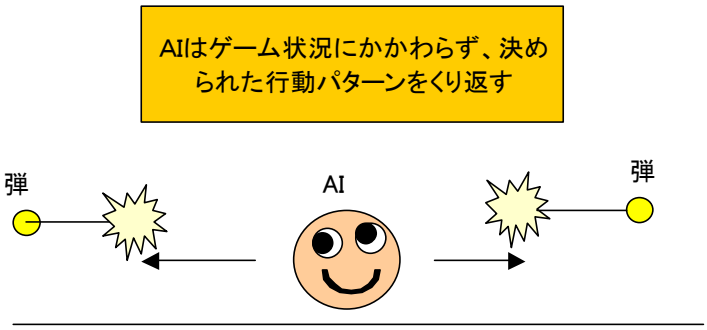


図 3.2-20 単純なワンパターン AI (例：左右に移動して弾を撃つ)

例えば、スペース・インベーダーのインベーダーは決まったパターンで上から下へ降りて来る。

しかし、パターンをくり返す AI でも、数が多ければ、プレッシャーになるし、またマップが狭ければ、ランダムなシューティング・ゲームであっても、プレイヤーに当たる確率が非常に高い。当時のゲームデザイナーは、このような単純な AI を「ゲーム内の地形と合わせて AI をどのような配置に置くか」によってレベルデザインを決定して行ったのである。逆にプレイヤーから見れば、こういった時代においては、まずは AI を観察し、パターンを予測し、最適な行動をイメージしながら自機をコントロールしてプレイする、というサイクルがゲーム攻略そのものであった。

こういったAIは環境を知覚することなしに、単なるパターン動作をくり返しているのであるから、AIにとって環境は存在せず、知性から身体への指令がくり返されるだけである。

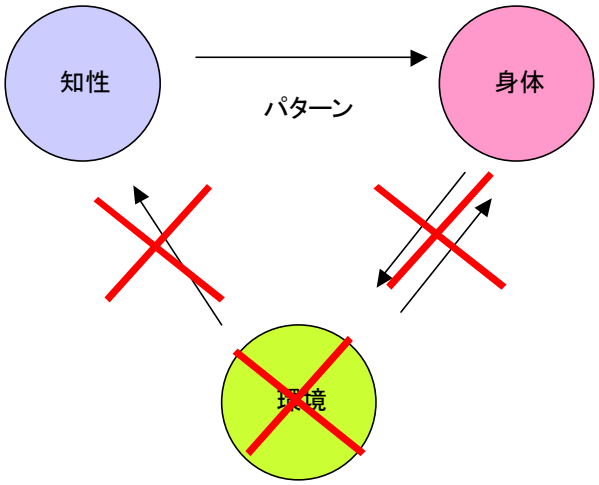


図 3.2-21　第一期における「知性」「身体」「環境」の相関図

環境との相互作用はない。

② インタラクティブなAI

次に、インタラクティブ性を持ったAIが出現する。同一のキャラクターが数パターンの行動パターンを持ち行動する。例えば、プレイヤーキャラクターと同じライン状に並べば「発見」したこととして、プレイヤーキャラクター近付いて来る、狙い撃つなどである。プレイヤーから見れば、最初はすぐに見破ることが出来る程度のロジックに過ぎなかったのであるが、次第に戦略ゲームなどの複雑なAIまで、様々なロジックパターンを持ったAIが実装された。実装としては、例えばアセンブラの条件分岐を用いたロジックであり、プレイヤーの位置、状態などを変数として条件分岐し、各分岐に対応する行動パターンが指定されている。今度は、プレイヤーにとっては、AIの分岐条件とそれに対応する行動パターンを読み取って行動することがゲーム攻略そのものとなった。「魔界村」（カプコン、1985年）の一面のボスや、「プリンス・オブ・ペルシャ」（Brøderbund、1989年）のボスキャラクターなどの読めそうで読めないAIの動きは、ユーザーに強いインパクトを残している。

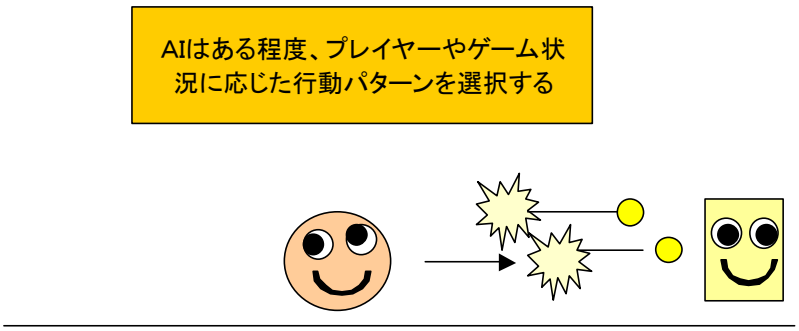


図 3.2-22　条件分岐によって行動パターンが変化する AI

ゲーム状況、主に初期においてはプレイヤーの状態に応じて行動を変化させる AI。例えば、プレイヤーを発見すれば追って来る、そうでなければ巡回する、などの AI である。

また、このような AI の場合は、環境から感覚を擬似した環境変数（プレイヤーの状態）を受け取って反応する。これは、AI がゲームの環境世界から情報を知覚する最初の段階である。

キャラクター・アニメーションとキャラクター移動について述べておく。

詳細な点であるが、キャラクター・アニメーションには動作が保証されている。知性はその条件に応じて身体に命令を出す。身体は、決められた動作を行う。しかし、この動作は多くの場合、環境に関らず実行できることが保障されている場合が多い。或いは、環境に関らず（実行できるかできないかに関らず）実行される場合が多い。例えば、弓を引いて矢を討つ動作を考える。その動作が 2D スプライト・アニメーションであった場合、周囲がどのような状況であれ実行される。3D 空間において弓を引く動作が壁に引っ掛かるかどうかを判定するようになるのは、ずっと先のことである。その意味で、身体動作から見た環境はインタラクション性を持たず最初一方向のものであった。

その一方、位置の移動に関しては、キャラクター同士の衝突や環境・オブジェクトとの衝突、などがあり、インタラクション性の高いものであった。

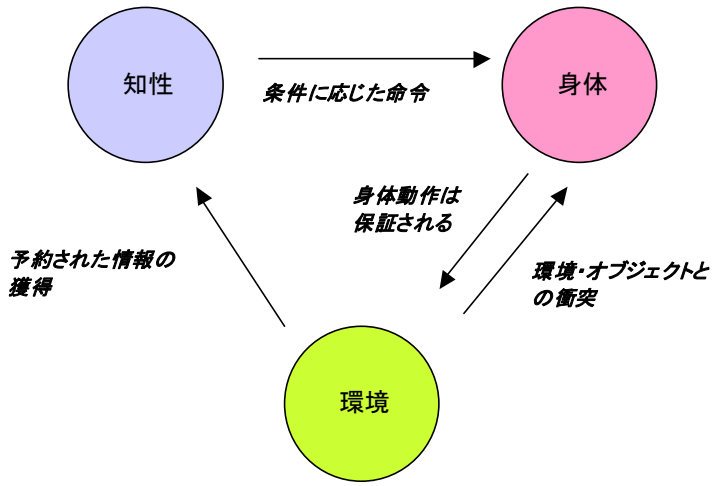


図 3.2-23　インタラクションを（条件分岐）加えた第一期における「知性」「身体」「環境」の相関図

また、こういった時期においても、ステージの地形とキャラクターAI はセットにして考えられており、最小限の AI の動作から最大限の効果を出そうとする努力がゲームデザイナーによって行われた。例えば、シューティング・ゲームで、非常に狭い通路を用意すれば、単純な AI でさえプレイヤーにとっては非常な脅威となる。早期に撃破しなければ撃墜される。また曲がり角を多くして AI を待ち伏せさせるのも同様である。つまり、ステージと AI はあらかじめ共犯関係にある一つの舞台装置として協調動作することで、単純な AI をプレイヤーに対して何倍も知的に見せたのである。例えば、1980 年という初期に作られたにもかかわらず、秀逸な AI とされる「パックマン」（ナムコ）も、迷路という特性の上にキャラクターごとの追跡ロジックが実装された。ステージとロジックのよい組み合わせを実現した例であった。

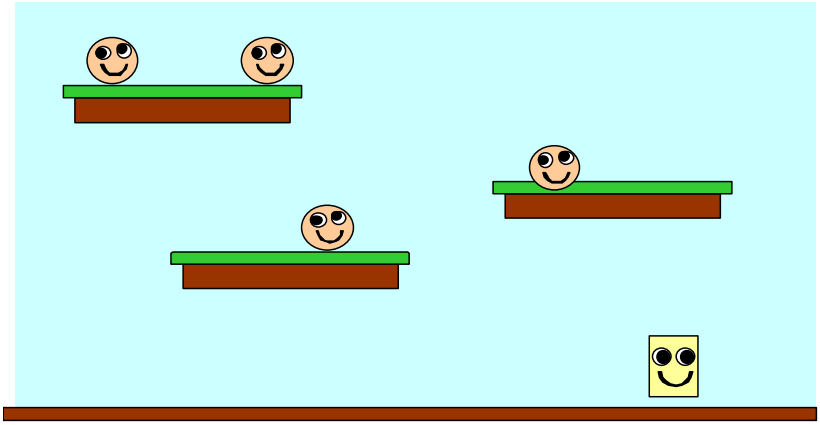


図 3.2-24　単純な動作の AI，ある程度分岐して行動する AI と 2D ステージ

ステージの地形と AI の配置によってゲーム性を作り出した。こういったゲーム作りは、現在も脈々と受け継がれている。

(b) 第二期　構造化される AI

第二期は 1990 年～2000 年に至る約 10 年間のキャラクターAI の変遷である。この時期は、3D 化、C,C++言語の導入など技術全体が急速に変化を遂げたために、AI もまた混沌とした変化、「なにか出来そうでどうやっていいかわからない」雰囲気を持っていた。学術では 90 年までで AI の流行が一端途切れて（第 5 世代コンピュータの失墜)、90 年代は寧ろ、そういった技術をスピンオフしてデジタルゲーム AI の分野で何が出来そうな雰囲気があったのかもしれない。また、同時に、遺伝的アルゴリズムやニューラルネットワークと言った最適化手法が、人口に膾炙する時期でもあり、ゲームプログラマーの射程にそういった技術が入って来た時期でもあった。そんな混沌とした中でも、全体としての大きなうねり「①構造化プログラミングとロジック実装」や、特定の作品で開発された AI「②内部パラメータ変動モデルオブジェクトによる AI 制御による日常系 AI」（The Sims）や「③遺伝的アルゴリズムとニューラルネットワーク」という進歩が見られた。

① 構造化プログラミングとロジック実装

90 年代に入って、コンシューマ機にも C 言語開発環境が入り構造化プログラミングの手法が導入された。構造化プログラミングは、アセンブラ言語に比べて、より大規模で抽象的なロジックを書くのに適した環境であり、また、多くの変数やデータ型を管理しやすくなった。また、この時期は、プレイステーションを始めとする 3D 世代機によって、ゲームステージが 3D 化した時代と重なる。ゲームステージの 3D 化は、ステージ内で AI が処理するべき情報を圧倒的に増加させた。多数のオブジェクト、オブジェクトの物理的運動、ステージの高低、マップの立体的形状などである。また、3 次元では、あるキャラクターが別のキャラクターから見えているか(Line of Sight, LOS)、攻撃可能かどうか(Line of Fire, LOF、火線)、などリアルタイムに AI が判定しなければならない情報が増え、また僅かな位置の変化で判定が変ってしまうというシビアなリアルタイム性を持つ。

こういった 3D 空間におけるあらゆる煩雑さを真正面から処理できる情報処理機能を持つ AI が実現されるのは第 3 期のことであって、第二期の 3D ゲームの黎明期においては、AI に対して取られた処置は次の三点であった。

- AI の行動範囲をある限定した自由に移動できる空間に限定すること。
- ステージを単純化すること（例えば、空の上なら AI は動き放題である)。
- あらかじめ、地形に沿った運動をプログラミングしておく（固定パスを与えておくなど)。

つまり、複雑になった 3D 環境に適応するために、環境を緩和するか、AI を制限するか、というアプローチが取られた。この第二期の初期において、デジタルゲーム AI は必

要とされていたのだが、あまりにもまだ未成熟であったのである。それは、AI 技術の未成熟度の問題と同時に、ハードウェアの制限でもあった。実際、PC ではなくコンシューマ期では、ゲームの仕組みに美麗描画エンジンを積めば、メモリも計算リソースもたいして残っていないというのが PlayStation2 の時代まで続いていた。90 年代の 3D ゲームの AI の多くは、プレイしてみれば、3D 空間を移動できても、何もない空間か、或いは複雑なオブジェクトを使用することもない。明らかに、AI の水準が、ゲームデザインに制限を加えているのがわかるだろう。

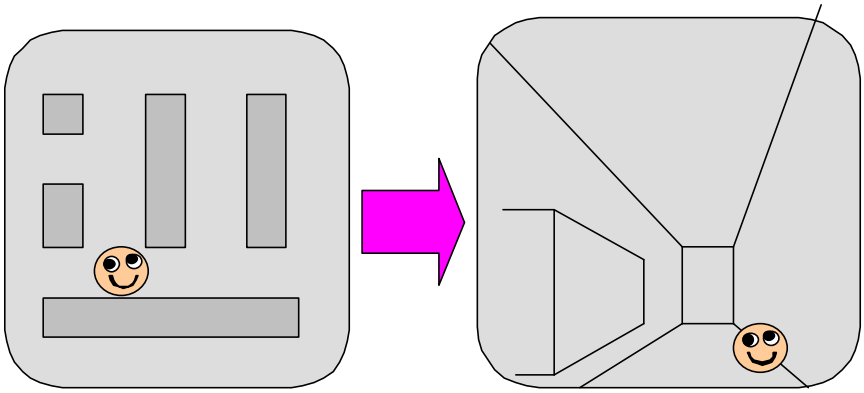


図 3.2-25　2 次元から 3 次元へ

マップの 2 次元から 3 次元化は、俯瞰的なロジックが通用せず、AI がリアルタイムに処理しなければならない情報を圧倒的に増加させた。その中でも、射線（Line of Sight、LOS）、火線(Line of Fire, LOF)判定は、ほんの僅かな位置の変化が true と false を分ける、シビアな変数である。

また C を始めとする構造化プログラミングにおいて、AI は、サブルーチン（独立した関数）であったり、ライブラリであったり、モジュールであったりと、分化されたものの、ゲームルーチンに深く埋め込まれて実装されるケースが多かった。これは、AI が、ゲーム内の環境の動的な変数を解析して行動を決める必要があるために、必然的に、ゲームのメインルーチンとのやりとりを必要とするのに便利な場所に置かれたからである。また、「ロジック＝AI」という狭い固定観念は、多数の条件分岐の下に、深いロジック（条件分岐の深い階層、関数の呼び出しの連続）を形成する傾向を促し、錯綜したコードでデバッグのしにくいコードが生産される傾向を強めた（現代的なデジタルゲーム AI では、AI はある仕組みやシステムで、そこから間接的に知性が導かれる、という観点である）。

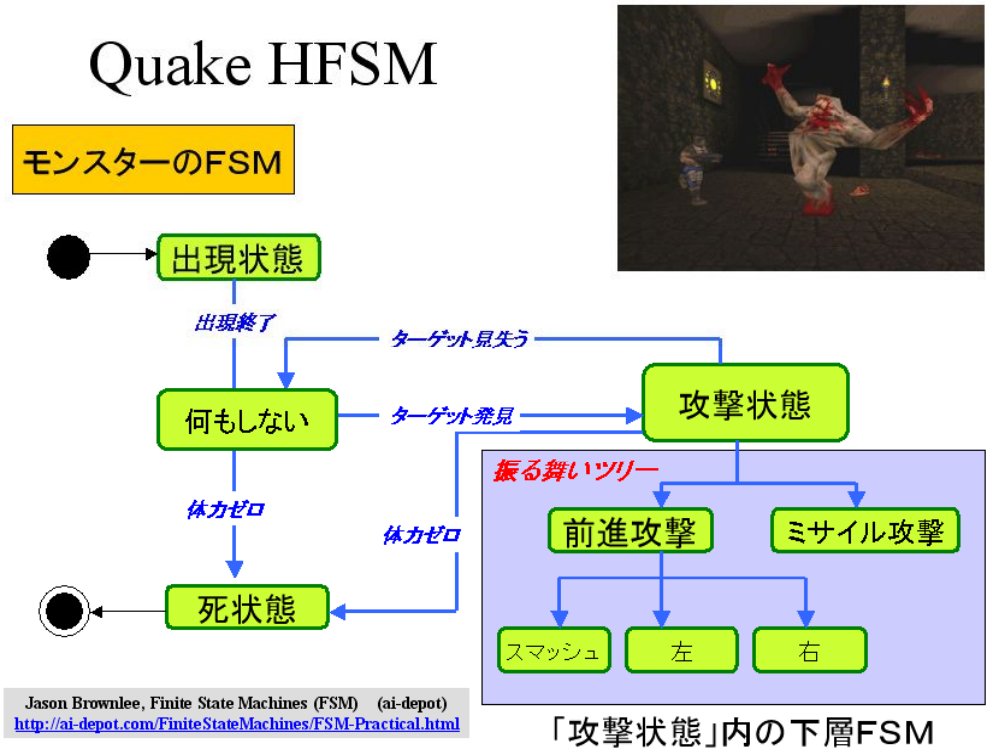


図 3.2-26　Quake における階層型有限状態マシン（Hierarchical Finite State Machine）[11]

こういった中でも単なるロジックコードを超えて、AI を制御するシステムを構築しようとする動きが出て来る。それは、主に PC ゲームにおいてである。有限状態マシン（Finite State Machine）や、評価値システム、そして、進化したスクリプトシステムなどによって、コードをできるだけ簡潔に保つという方向と共に、AI 技術が導入され始めたのであった。これは、C 言語のように、様々なダイナミクスを表現できる言語になって、見通しがよくなった恩恵でもあった。こういった簡単な AI 技術の導入とその経験は、次世代 AI の土壌となった。例えば、PC ゲームの Quake（id Software,1996 年）などでは、階層型の有限状態マシンが（HFSM）が多用されている。

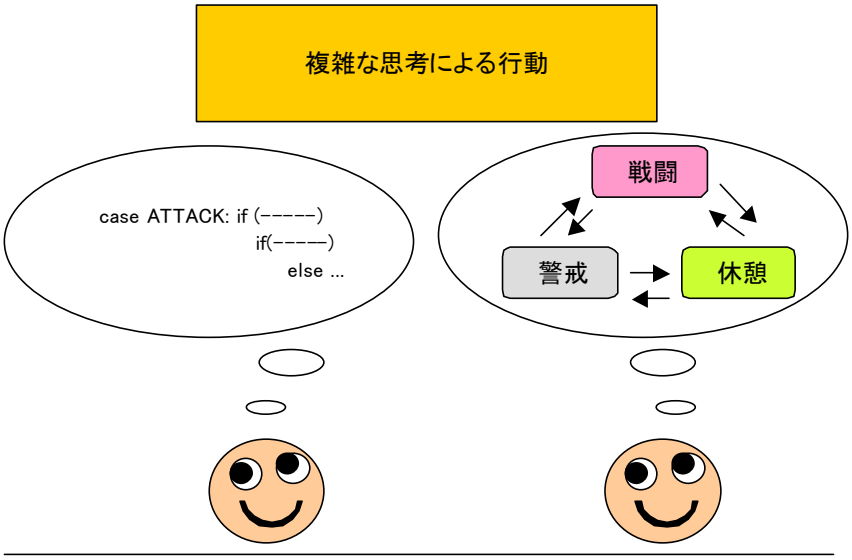


図 3.2-27　論理思考の実装と有限状態機械の導入

この時期の AI から身体にボーンの構造が入り、このボーンの稼動は計算によって行なわれ、身体と環境との衝突判定が、汎用的なプロセスとして実装されることになった。即ち、キャラクターのアニメーションは環境によってより強く制限されることになる。

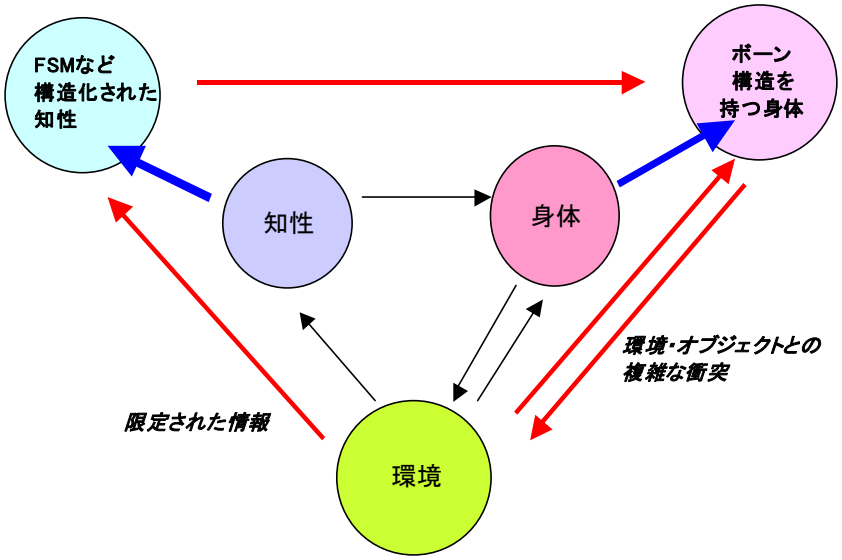


図 3.2-28　構造化する身体と構造化する知性

身体と知性お互いの進化と共にお互いが進化する共進化的なモデルが理想的である。そもそも生物の進化は脳ではなく、身体の進化に適応するために脳が進化したとも考えられる。

②　「内部パラメータ変動モデル」と「オブジェクトによる AI 制御」による日常系 AI

AI と言えばプレイヤーキャラクターを攻撃する AI が殆どであるが、「The Sims」（Maxis, 2000 年）を始めとする日常生活をプレイヤーが楽しむゲームでは、「家で過ご

し街を歩き、他の NPC と会えば会話する」という「日常系の AI」が出現した[12]。一般的に「日常系 AI」では、「単にプレイヤーを攻撃すれば敵と見なしてくれる攻撃型の AI」と異なり、「まるで人間のように」振舞う必要があるため難易度が高い。「The Sims」の全てのシステムは、ある日、ふいに Will Wright が書いて持って来た 300 行程度のプログラムに始まったと言われており、この C 言語コードは公開されている[13]。そこには AI を含め、「The Sims」を動かす主要な原理が全て記述されていた。以下に簡単に、「The Sims」の AI の原理を解説する。

最初の仕掛けはゲーム内のオブジェクトに行う。各オブジェクトには、そのオブジェクトで AI が行うべきアニメーション、サウンドエフェクト、ロジックが仕込まれている。つまり、AI はそのオブジェクトを選んだ時点で、どのように制御されるかが、ある程度決まる。

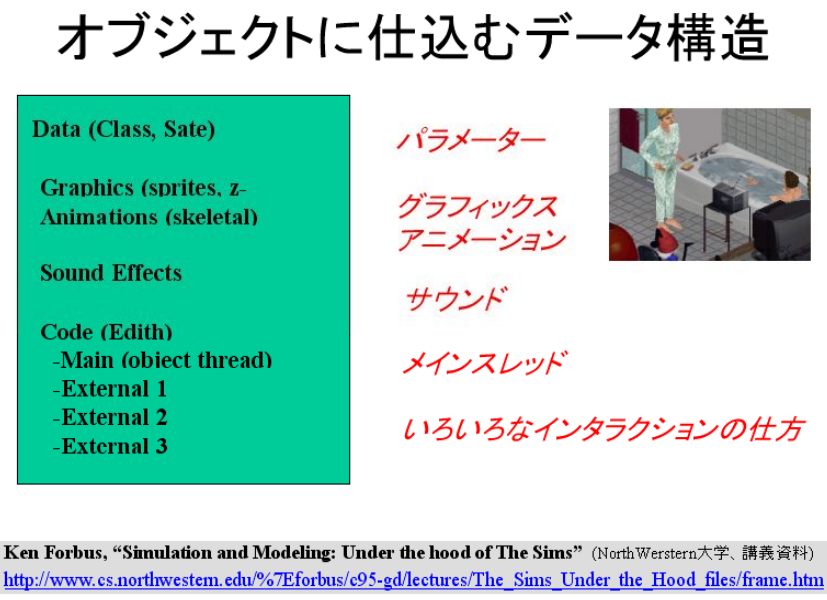


図 3.2-29　『The Sims』において各オブジェクトに対して仕込まれているデータ[14]

このような日常系の AI では、こういった「オブジェクトが AI を制御する」方法を取る場合が多い。

ただし、これだけでは、ただオブジェクト一つに対して一つの行動を行うだけである。これに加えて、AI が活動を条件分岐つきツリーで構築する GUI ツール「Edith」が準備されており、ゲームデザイナーはこれを用いた一連の動作ツリーをカスタマイズする[15]。簡単な YES / NO 分岐だけであるが、条件に応じて一連の動作ロジックを構築できる。

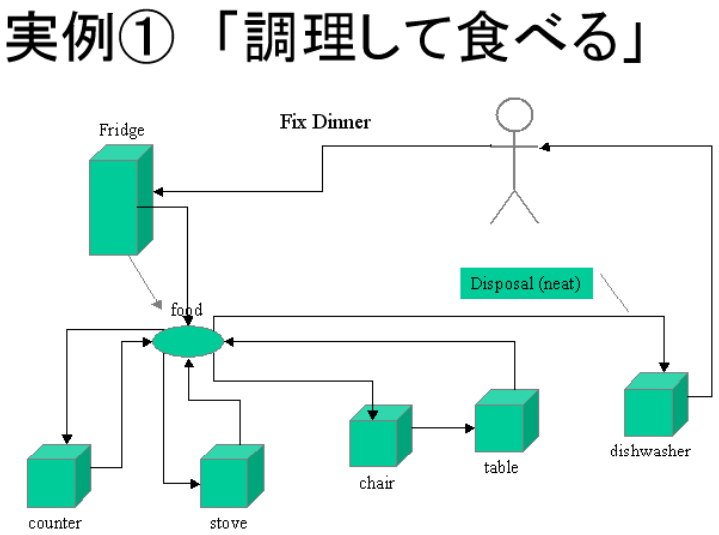


図 3.2-30　『The Sims』における「AI が食事をする」一連の動作の解説図

各オブジェクトには先に述べた動作アニメーションなどのデータが埋め込まれている[14]。「調理して食べる」という行動には、「冷蔵庫を見る」「材料をピックアップする」「材料に応じた調理をする」「食べる」「皿を洗う」という一連の動作が必要である。

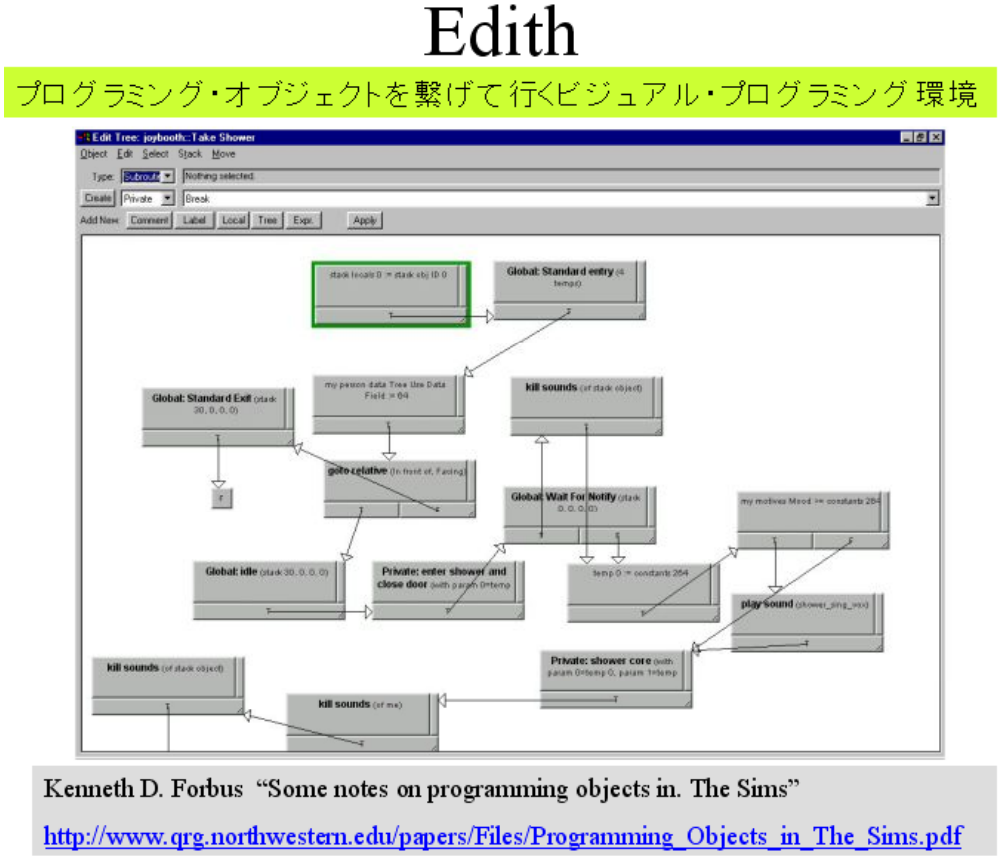


図 3.2-31　AI 及びゲームシークエンスを定義する「Edith」[15]

次に、AI が一体どのような行動を取るか、という判断がなければならないが、日常的なシチュエーションが無限にあるゲームで、ロジックの実装は固定した行動となりやすく適していない。

そこで、AI の内部状態を表現する変動するパラメータを用意し、その内部パラメータを使って周囲にあるオブジェクトを評価する。お腹が空いている時は、冷蔵庫の評価値は上がり、寂しいときは友人の評価値が上がる。トイレに行きたいときはトイレの評価値が上がる。しかし、一体、欲求が満たされれば、評価値は下がる。例えば、お腹がある程度ふくれると、冷蔵庫の評価値は下がる。そして、周囲のオブジェクトの中から、Mood（内部状態のパラメータにウエイトをかけて足した値）を最大化するオブジェクトを選択する。

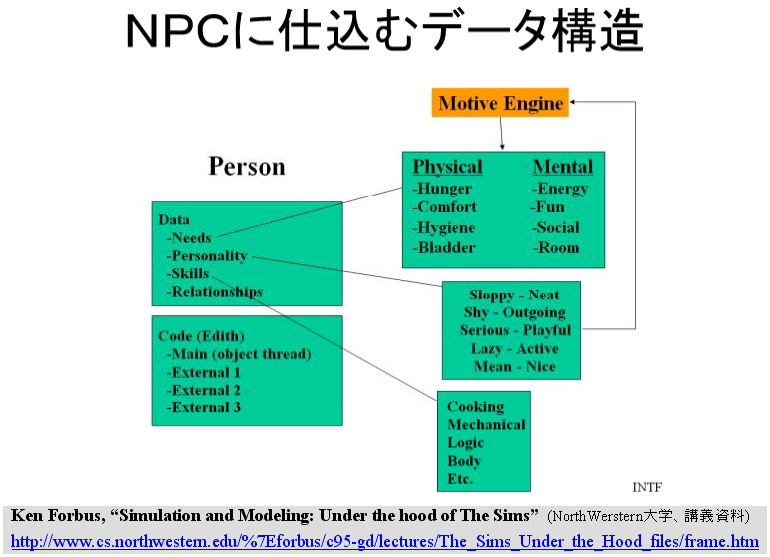


図 3.2-32　『The Sims』において AI に対して仕込まれているデータ[14]

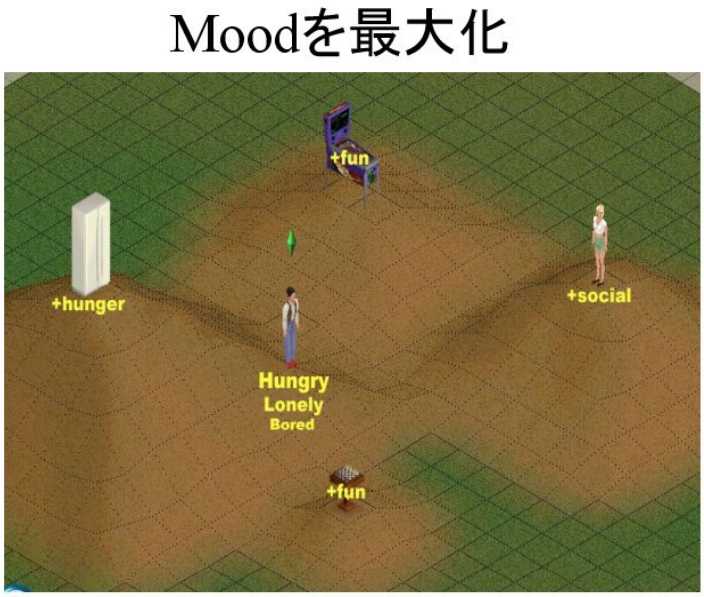


図 3.2-33　『The Sims』の AI の説明図

各オブジェクトには AI 制御するルーチンやアニメーションのデータが用意されており、オブジェクトが AI を制御する。複数あるオブジェクトの中から、Mood 係数（自分の気分）を最大するオブジェクトを最適な行動を選択する[14]。

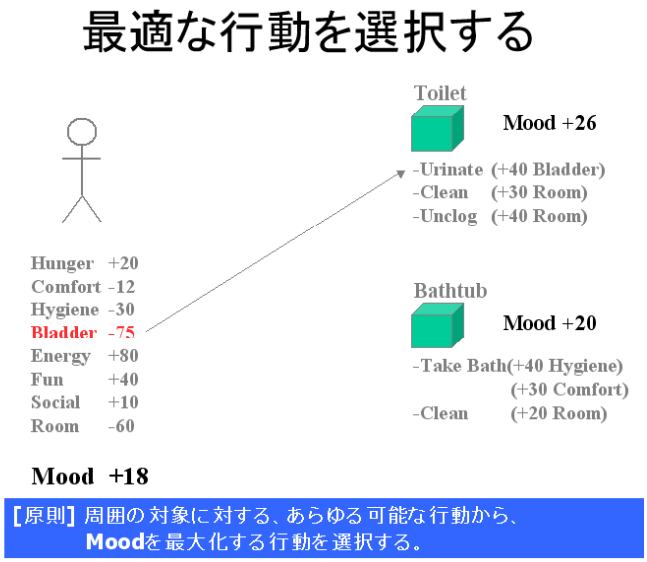


図 3.2-34　『The Sims』の AI の意思決定過程

AI は、様々な内部パラメータを持つ。その値に応じて、各オブジェクトが評価される。例えば、Bladder(尿意)がある（値が低い）ときにはトイレの評価値は高くなる。衛星状態が低い時にはバスタブの評価値は高くなる。各オブジェクトには、動作が複数定義されていて、バスタブであれば「入る」「洗う」があり、動作に応じて回復するパラメータが違う。入れば、衛生状態が上がるし、リラックスする。掃除すれば、部屋の衛生度が上がる[14]。結果として、Mood をどれぐらい上げるかが、選択の基準となる。

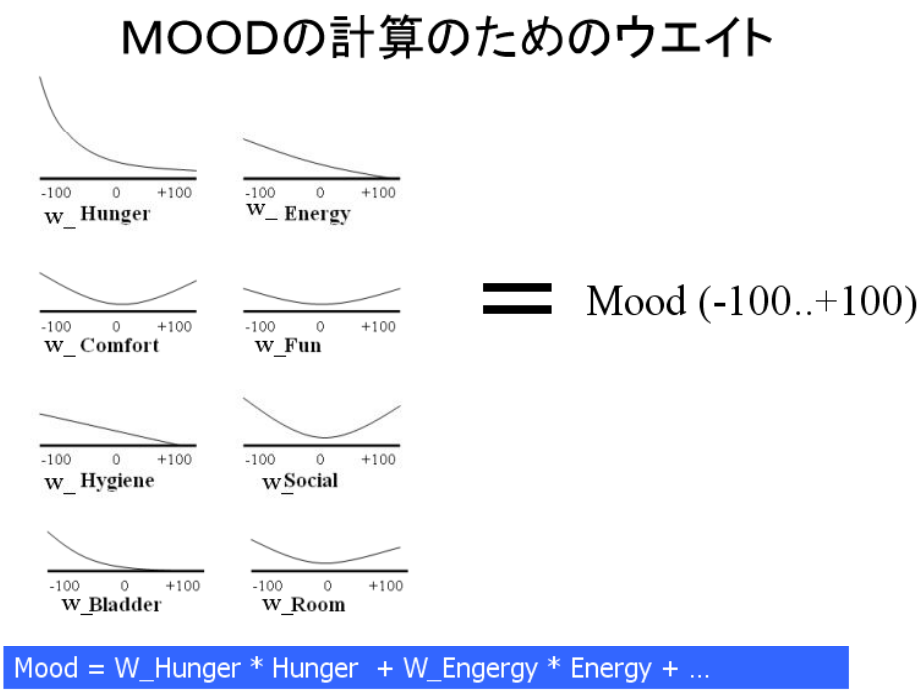


図 3.2-35　Mood 計算のためのウエイト

各内部変数は （–100…100）に制限され、この変数がムードにどう貢献するかを決めるのが各変数に対するウエイトである[14]。

例えば、Hunger というパラメータがある。オブジェクト冷蔵庫へ行けば、この Hunger というパラメータが回復し、Hunger パラメータにかける係数は少し下がる。総合的には、冷蔵庫へ行くことは、ハピネス係数をある程度上げる。例えば+40 とする。或いは、部屋を掃除すれば、Mood は 15 上がる。全てのオブジェクトの行動は、ハピネス係数に変化をもたらすが、現在、部屋なら部屋の中にある全てのオブジェクトの中から Mood の上昇に最も適したオブジェクトを選択し行動するわけである。

そして、一端、ある欲求、例えば内部変数 Hunger が満たされると、冷蔵庫を選択することによるハピネス係数の上昇は小さくなり（なぜなら重み係数が小さくなっているから)、もっと他のこと、洗濯をするとか、友人と話す（Social=社交性パラメータと関係している）と言ったことの方が Mood をよく上昇させるので、そちらを選ぶ。これが、AI の行動原理である。また、こういった内部パラメータは時間的に変化しそれに応じたオブジェクトを選択するので、複雑でかつ合理的な行動選択を行う AI が出来上がるわけである。この原理は『Spore』（Maxis、2008 年）に継承される。

こういった数理的なダイナミクスを用いた AI は、日本が得意としないとことである。米にしても、Will Wright を始めとする優秀なスタッフの数学的感覚があってのことであろう。「The Sims 3」でも、この実装は引き継がれ、パラメータの数も増やされ、ますますリアルティのある AI と発展している[16]。

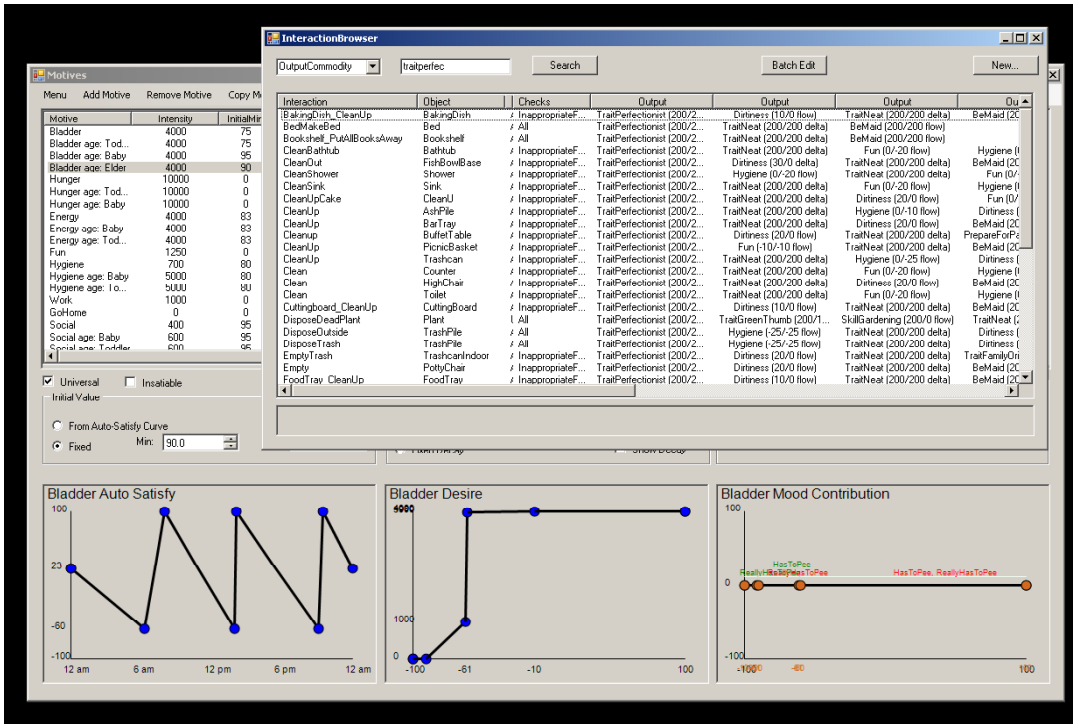


図 3.2-36　『The Sims3』におけるオブジェクト管理と内部変数の変動（下）

GDC2009「Implementation of Personality TrAIts in The Sims 3」より[16]。

③　ニューラルネットワークと遺伝的アルゴリズム

ニューラルネットワーク

ニューラルネットワークの数理モデルは、既に 1960 年代から研究されており、また遺伝的アルゴリズムは 1970 年代から研究されていたが、ゲームプログラミングに C 言語が導入され、簡単に実装出来るようになったのは、90 年代からであった。ただ、デジタルゲームにおいて、何処でどう利用すればよいのかが見えなかった。そして、それは、今もたいして変わっていないが、1995～2000 年にかけて、こういったアルゴリズム的な手法が幾つかのタイトルで応用され、今もなお、デジタルゲーム応用への指針の一つとなっている。

まず、ニューラルネットワークで知られている事例は、以下の 4 つである[17]。

1996 年　BATTLECRUISER: 3000AD (3000AD)
1997 年　がんばれ森川君 2 号　(muumuu)
2000 年　Colin McRae Rally 2.0 (Codemasters)
2001 年　Black & White　(Lionhead Studio)

このうち、『Colin McRae Rally 2.0』 は、車のコースを辿る制御にニューラルネット

ワークを応用している[18]。 『がんばれ森川君 2 号』では、五感を持つピット（AI）が、出会ったアイテムに対して「そのアイテムに対して何をすべきか」「アイテムの印象」（そのアイテムが快いか、不快か）を学習させるために用いている。その実装の詳細については、開発者である森川幸人氏の著作に詳しい[19]。

遺伝的アルゴリズム

遺伝的アルゴリズム（Genetic Algorithm，以下 GA）について、ゲームへ応用するフレームは多くのゲーム AI の教科書に解説されデモも多い[20,21]。しかし、不思議なことに、実際に商業ゲームで実装された例は多く知られていない（少なくとも筆者は殆ど聞いたことがない）。そんな中で、GA を使用した際立った仕事として『アストロノーカ』（muumuu,1998）[22]が挙げられる。『アストロノーカ』は、GA をゲームデザインとどう融合させるか、というお手本のようなゲームであり、GA のアルゴリズムがゲームの面白さと直接結び付いており完成度が非常に高い。また、開発者の森川幸人氏によってアルゴリズムの詳細が解説されている[23,24]。

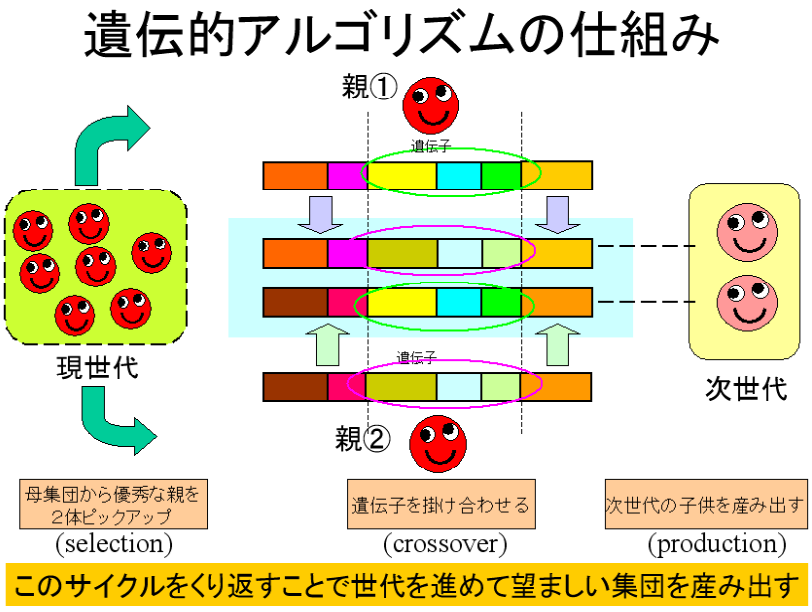


図 3.2-37　キャラクターを進化させるために遺伝的アルゴリズムを用いる

遺伝的アルゴリズムによる個体集団進化のシークエンス

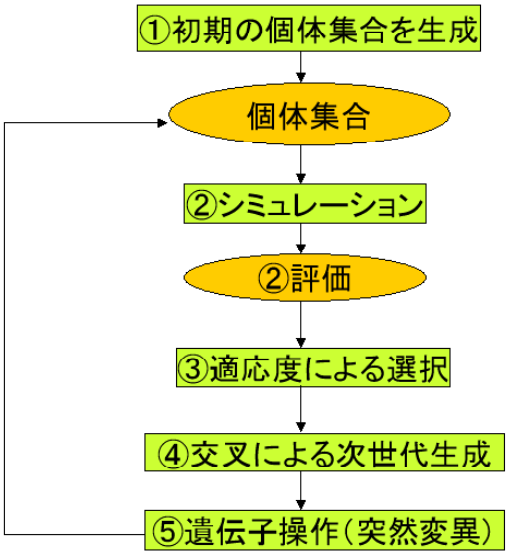


図 3.2-38　遺伝的アルゴリズムの仕組み

ある程度、公平なシミュレーション環境が必要である

遺伝的アルゴリズム×ニューラルネットワーク

ニューラルネットワークを遺伝的アルゴリズムによって発展させる手法が、NEAT(Neuro Evolving of Augmenting Topologies) として知られている[25]。また、オースティンのテキサス大学における研究で「NERO」という 3D ゲームがこの原理を元に開発され公開されている[26]。

⑤ニューラルネットワークの構造が進化させる「NEAT」の技術

Mat Buckland, Chapter 11, AI techniques for game programming, Premier Press, 2002
(実行ファイルとソースコードがCD-ROMにあります)

これまでニューラルネットは、最初に構造を定義した後は変化しなかった。

➡ 動的にニューラルネットの構造を変化させる技術
Neuron Evoluation of Augmenting Topologies (NEAT)

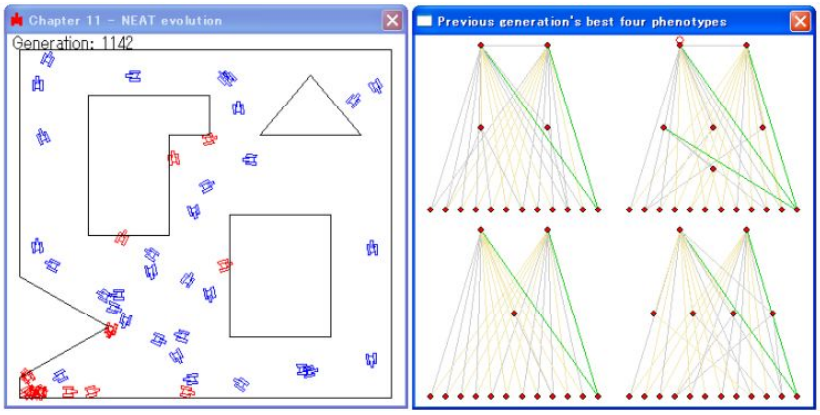


図 3.2-39　「NEAT」の仕組み。動的にニューラルネットワークのトポロジー（形状）が進化して行く[25]。

⑥リアルタイム「NEAT」を使ったシミュレーションゲーム「 NERO 」

Neural Networks Research Group, Department of Computer Sciences, University of Texas at Austin, Neuro-Evolving Robotic Operatives, http://www.nerogame.org/,
(ゲームがあります)

①rtNEAT（リアルタイムNEAT）システムの上で、ユーザーが兵士を訓練する。
②訓練させた兵士同士を対戦させる。

図 3.2-40　テキサス大学の Kenneth O. Stanley 氏による研究成果である「NERO」兵士のニューラルネットが形状を含めて動的に進化して行く[26]。

このようなアルゴリズム的な AI の方向は 1995 年～2000 年にかけて、それほど範例が多くないものの応用を見たが、2000 年以降、ゲーム開発は美麗グラフィックによってユーザーを引き付けるという方向へ流れ、こういった計算量の多いアルゴリズムを用いたゲームは全く見られなくなってしまった。現在 2009 年は、ようやく美麗グラフィックの勢いは落ち着きを見せ（依然、最大の潮流であるが)、もっと他の用途にプロセッサリソースを用いようという流れになっている。しかし、キャラクター制御や意思決定アルゴリズムまで力が注がれているものの、まだ、GA や NN までは探求の手が戻っていないのが現状である。

(c) 第三期　自律化する AI

第二期におけるステージの 3D 化と複雑化は、AI の知性と身体を構造化させると同時に肥大化させ、FSM などの技術が使われ始めた。そして、こういった流れは、これまでステージの一部や自動人形であった AI を、ゲーム内の自律した存在として扱おうとする気運を高めたのであった。

これまでは「知性の構造化」であったが、ここでより広い視点に立って、環境、身体、知性を包括的にモデル化する「アーキテクチャ」という概念が AI に導入されることになる。具体的には、エージェント・アーキテクチャ（実装フレーム）を持った自律した（自分で判断して行動する）AI が開発されたのであった。

特にこの方向の AI は、「DOOM」（id Software、1993 年）[27]のソース公開によって、ゲームジャンルの確立と実装技術のスタートを同時に得た、リアルタイム性を要求する欧米の FPS 分野において、2000 年から急速に発展して行った。また、この方向には米の大学の研究者、またその修士や博士を治めた学生、開発者の貢献が大きかった[7]。
この第三期の技術的内容に関しては、昨年の報告書の第三章で詳細に解説したので[28]、ここでは流れに重点を置いて解説したい。

① エージェント・アーキテクチャ

MIT メディアラボにおける仮想空間内の生物（犬）を実現する、というプロジェクトで育まれた、エージェント・アーキテクチャの一つ「C4 アーキテクチャ」という実装フレームがあった[29] [30]。このアーキテクチャは 2000 年の GDC(Game Developers Conference)においてゲーム開発者に対しても解説され、その後、デジタルゲームのキャラクターAI の設計に大きな影響力を持つこととなった。

このアーキテクチャの最も大きな特徴は、環境と AI の思考を完全に分離し、さらに、思考と身体制御も分離する、という点にあった。即ち、環境と AI の間にはセンサーと呼ばれる関数群が置かれ、この擬似感覚が環境から AI に流れる情報を収集し Working Memory に蓄積され、その情報を基に思考が構築され、その結果の意思決定を受けて身

体制御情報がブラックボードに記述され、その情報を読んだ身体駆動モジュールが身体制御を行う、という一連の流れの示したのであった。これによって、

・ 環境、身体、思考という 3 つの部位が明確に分離すると共に、
・ 情報の流れと記憶の蓄積が明確に定義され、

AI が環境から自律した存在として実装するステージへ移行する端緒を切ったのであった。

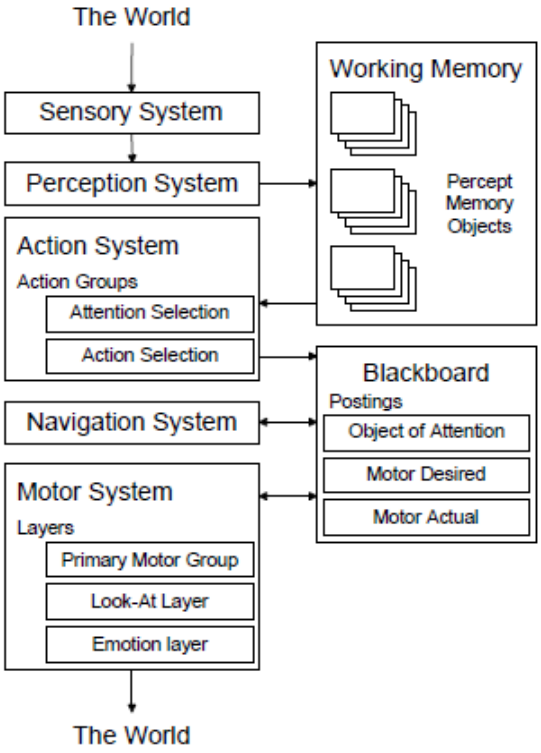


図 3.2-41　C4 アーキテクチャ

MIT Media Laboratory の Synthetic Character Group が提案したエージェント・アーキテクチャ。C4 の C は Cognitive の C、試行錯誤の後の 4 つ目のモデル。特徴はまず全ての記憶が一定の形式で記憶（Working Memory）によって蓄積されるという点、そしてテンポラリーな行動計画はブラックボードに記述される点、学習機能を持つ点などである[29] [30,31]。

C4 アーキテクチャは、以後、改良され、応用され続ける。F.E.A.R. (Monolith Productions)の AI は、この MIT メディアラボに在籍していた大学院生 Jeff Orkin 氏が設計し、C4 アーキテクチャの思考部にアクション・プランニング（一連のアクションプランを思考する）を実装した。プランニングによって AI は、自身の行動の一連のプランを自律的に生成する能力を得ることとなった。この仕事は、ゲーム AI 開発者に大きな影

響を及ぼし現在でもなお、ゲーム AI 開発者に最も多く参照される仕事の一つである[32] [33] [34]。

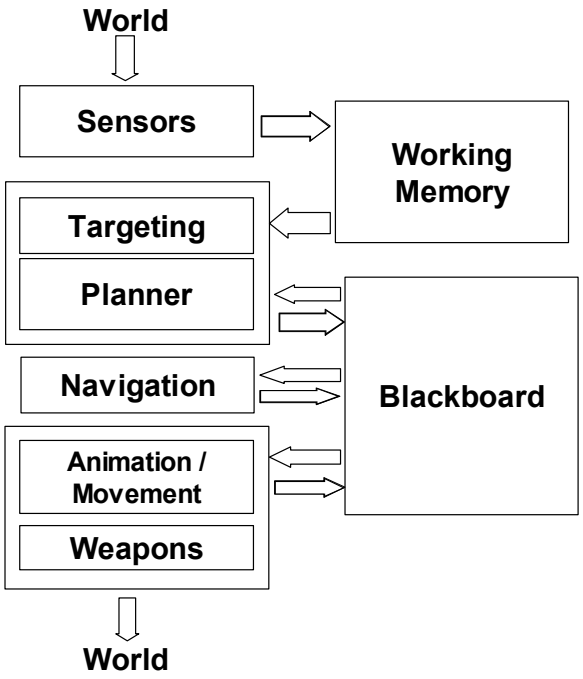


図 3.2-42　F.E.A.R. におけるエージェント・アーキテクチャ

C4 アーキテクチャと同じ構造であるが、プランニングの機能や各要素が F.E.A.R.用に変更されている。開発者の Jeff Orkin は、C4 アーキテクチャを作成したグループと同じ MIT メディア・ラボ出身である[32] [33] [34]。

最初の「HALO」(Bungie)のタイトルは、エージェント・アーキテクチャに沿った設計で、ゲーム世界で起こったイベントに対して適切な反応をし、感情を表現する仕組みになっていた**[35]**。次に、同じく MIT メディアラボから Bungie へ入社した Damian Isla 氏によって、Halo2,3 の階層型 FSM の AI が実装された。これは、第 2 期の FSM を階層型にしたものであり、特にアクションに適用したものはビヘイビア・ツリーと呼ばれる。この手法は実装が単純で、かつ効果が見えやすいため、現在、キャラクターAI の代表的な技術の一つとして広く知られている[36]。これは意思決定部分だけを抜き出しているのでエージェント・アーキテクチャには見えないが、実際は環境と知性と身体が明確に分けられており、エージェント・アーキテクチャとして設計されている。

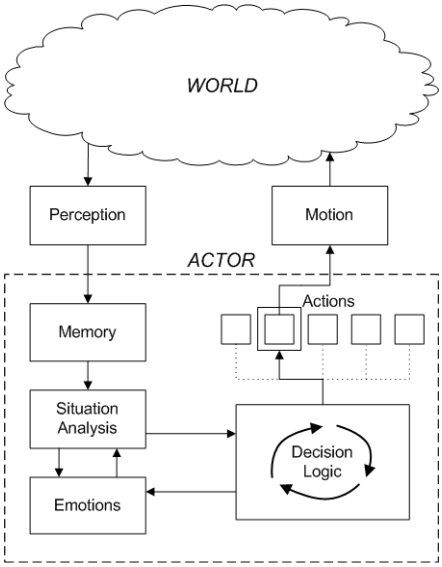


図 3.2-43　『HALO』におけるエージェント・アーキテクチャ

感覚とモーションのレイヤーによって AI の知性と世界が明確に分離されている。アクション選択はスクリプティングできるようになっている。感情は意思決定には関係せず、単にセリフ選択など感情的な表現を行う場合に用いられる[35]。

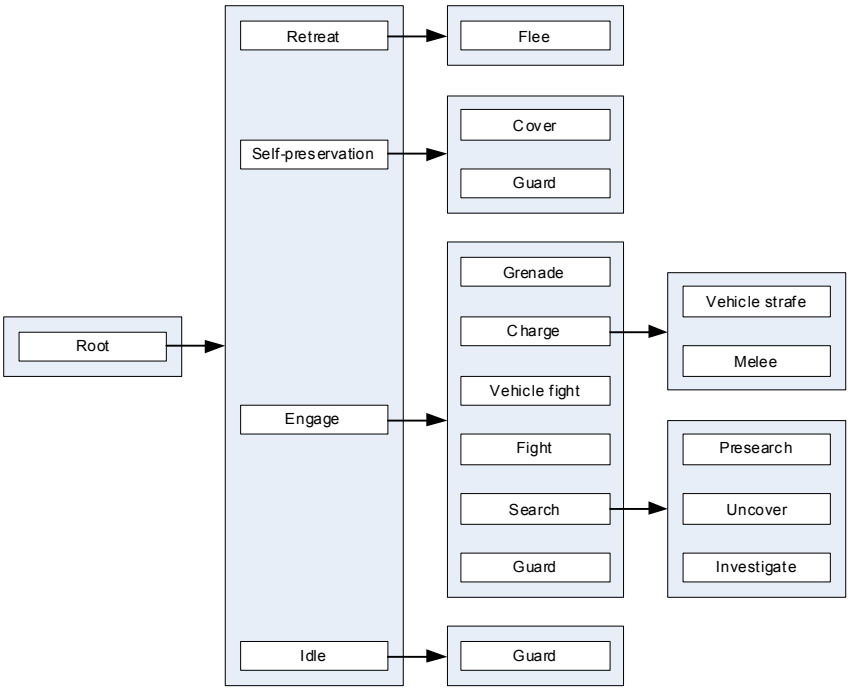


図 3.2-44　『Halo2』における階層型有限状態マシン(HFSM, heuristic FSM)

一方向にしか伸びていないので形式としては非循環グラフ（Directed Acyclic Graph, DAG）である。簡単に、振る舞いグラフ（Behavior Graph）とも呼ばれる。左から右へ状況が「分類」されて「階層化」される[36] [37]。

② 知識表現

また、Halo シリーズで（地味であるが）革新的であったのは、知識表現に重みを置いた点である。知識表現とは、ゲーム内にあるオブジェクトが、AI にとってどのような意味を持つかをデータで表現するものである。例えば、ゲームステージに置かれた車は、単にそのままであればポリゴンの塊に過ぎないが、車の「位置」「どちらの方向に動かすことが出来るか」「運転することが出来る」などの情報をセットにしてオブジェクトに付与しておく。すると、そういった付与された情報によって、AI は、目撃したオブジェクトから抜き出したその情報を基に様々な判断を行うことも出来るのである。そして、周囲にある全てのオブジェクトから、自分がその場で行うことが出来る全ての行動を列挙することで、その中から現状に適した行動を選択する。例えば「車を動かす」「レバーを引く」「ドアを開ける」を抜き出し、この中から、最適な行動を選択する。

ある対象に対して知性が許される行動のことを認知科学では「アフォーダンス」と呼ぶ。アフォーダンスは、普段、我々が意識的に判断しているのではなく、物事を知覚する過程そのものの中に埋め込まれた処理である。こういった無意識な判断を、デジタルゲームでは、オブジェクトに情報を付与することで実現するのである。

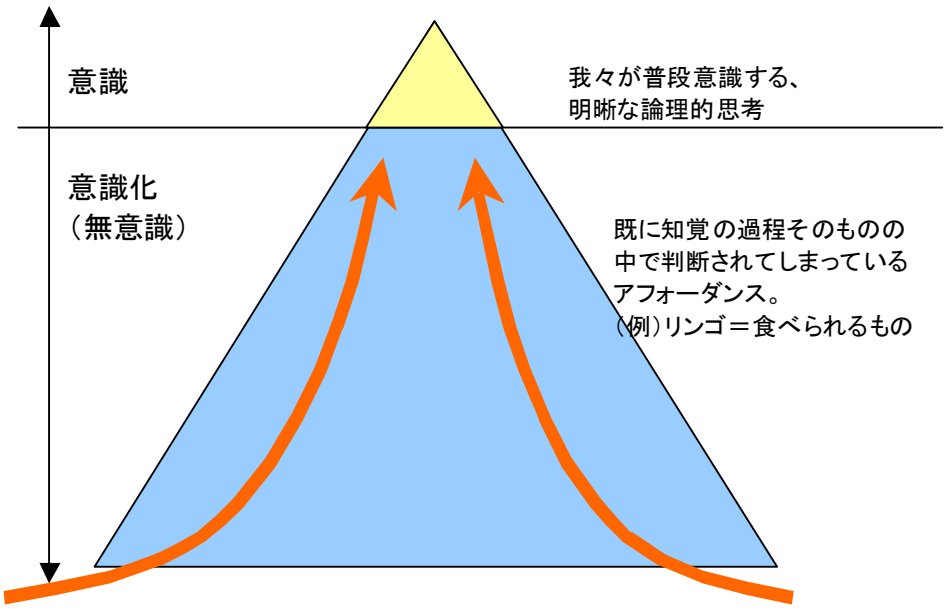


図 3.2-45 事物の意識に先んじて既に世界観の中に判断として含まれているアフォーダンス

アフォーダンスはもともと認知科学の概念であるが、ゲーム AI でも重要な概念である。アフォーダンスはキャラクターAI にとって、そのステージでどういう行動が出来るか、を意味する。

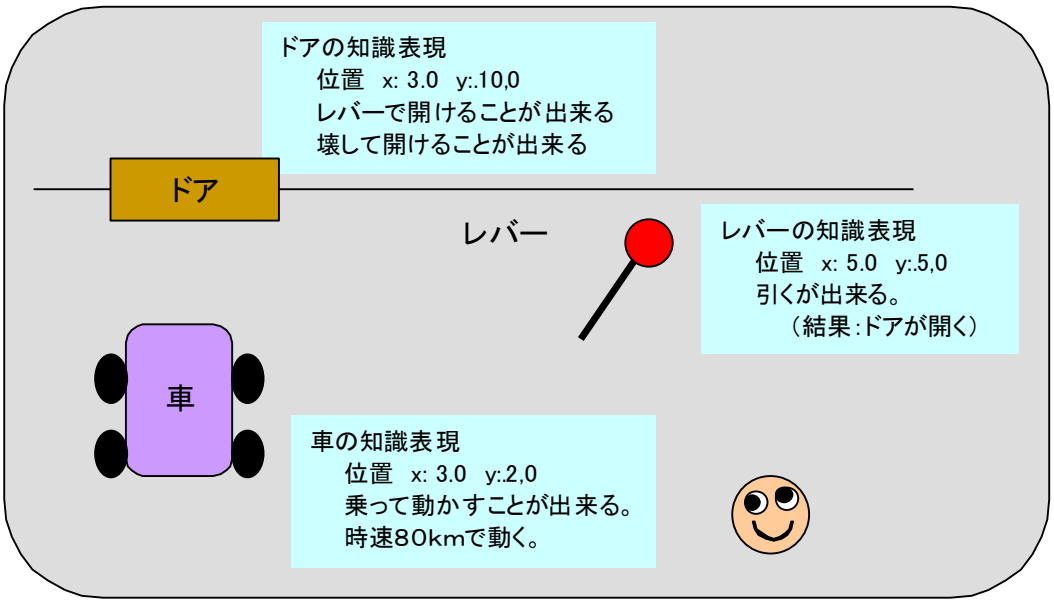


図 3.2-46　ゲームマップにおける知識表現とアフォーダンス

環境内のオブジェクトには、それがキャラクターにとってどのような動作ができるか（＝アフォーダンス）という情報が埋め込んでおくことで、AI はその中から目的に応じた行動を選択することが出来る。

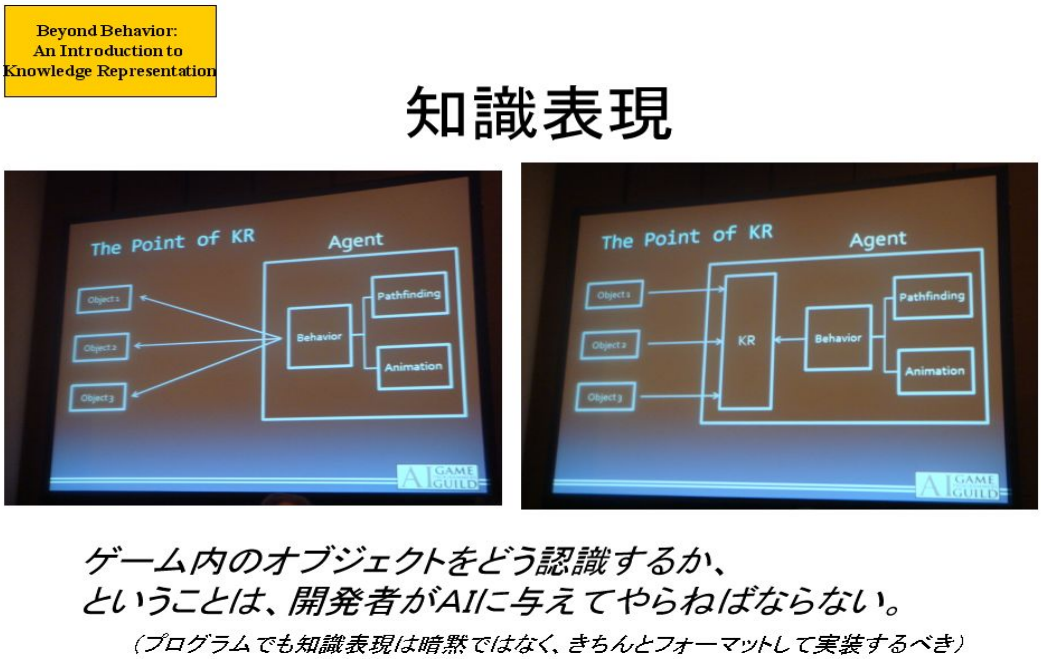
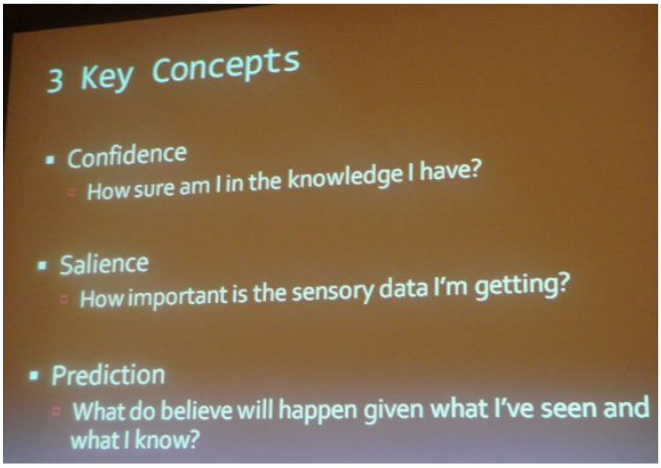


図 3.2-47　知識表現とは何か？

GDC2009 における Damian Isla 氏の講演「Beyond Behavior: An Introduction to Knowledge Representation」より[8]

知識表現で大切なこと



情報の正確性の指標

重要性

認識した事柄から
何が予測できるか？

図 3.2-48　知識表現のポイント

GDC2009 における Damian Isla 氏の講演「Beyond Behavior: An Introduction to Knowledge Representation」より[8]

知識表現

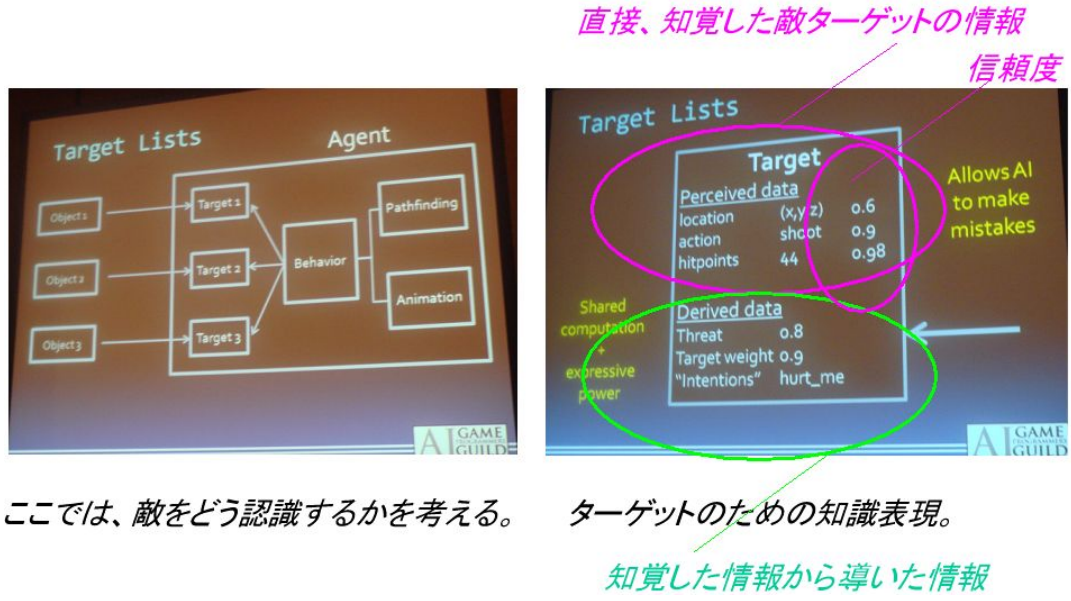


図 3.2-49　知識表現の具体的表現

GDC2009 における Damian Isla 氏の講演「Beyond Behavior: An Introduction to Knowledge Representation」より[8]

③　パス検索システム

2000 年からの 10 年間で、パス検索技術は完全にキャラクターAI の基本として、FPS を始めとする多くのゲームで導入されることとなった。パス検索技術は、地形を覆うように、ウェイポイントと呼ばれるポイントを配置するか、地形に沿ってナビゲーション・メッシュを敷き詰めるデータを準備しランタイムで、任意の点から任意の点のパスを、A* アルゴリズムなどの検索アルゴリズムで動的に検出する技術である。この技術によって、通行可能なキャラクターはあらゆる場所から場所からの移動が可能になり、もはやこれまでのように一定の領域に制限されるということはなくなったのである。また、マップの拡大に応じて、パスデータは階層化（例えば、エリア、部屋、部屋の中のナビゲーション・メッシュ）され、検索も階層化（エリアを要素とする検索、部屋を要素とする検索、ナビゲーション・メッシュを要素とする本来の検索）されることとなった。

パスデータ階層化は計算量を軽減すると共に、戦略性を加味したパス検索技術などを可能にする。大局的なデータは戦略に、細かなパスデータは実際のパスの導出に使用される。現時点では、パス検索は既にデファクトスタンダードであり、これからは、スマートなパス検索（賢いパス検索）が追求されている。

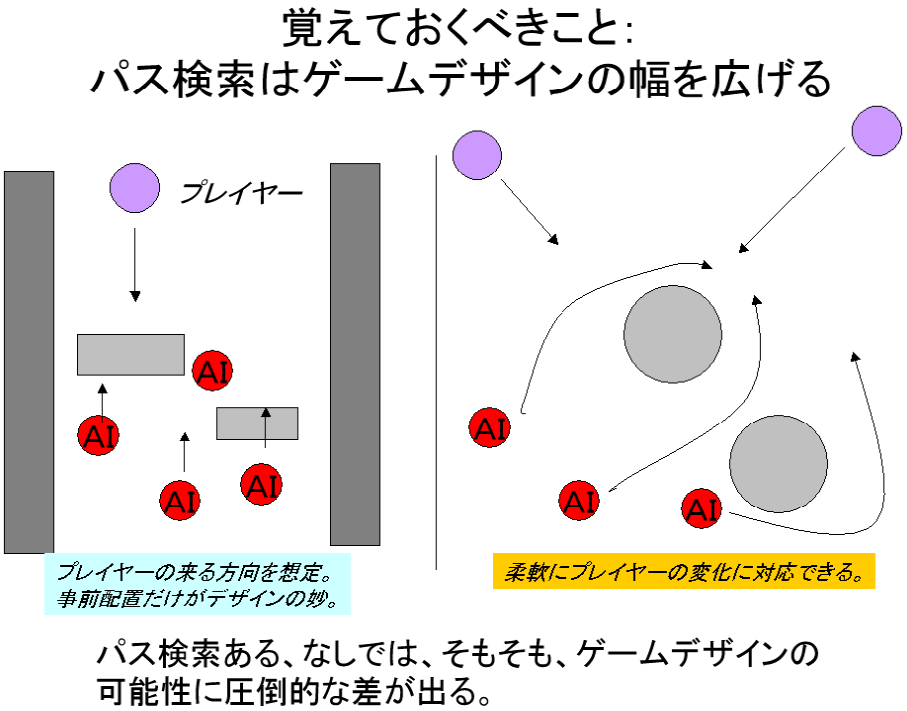


図 3.2-50　パス検索システムが AI に持つ意味

パス検索技術はこれまで局所的に配置されて想定した単純な範囲しか動けなかった AI を複雑なマップ全域を移動可能領域として解き放った[8]。

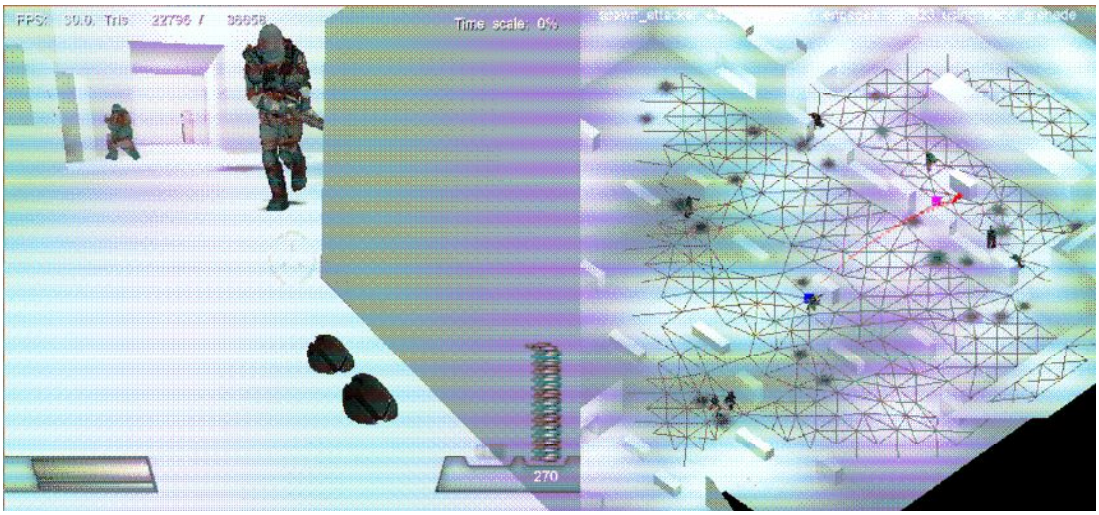


図 3.2-51 『KILLZONE』におけるウェイポイント・マップのテストマップ[38] [39]

KILLZONE はウェイポイントに情報を埋め込むことで、位置に依存する思考に特化した AI を作り上げた。これは『KILLZONE 2』にも引き継がれていくことになる[40]。

また、こういったパスデータをベースとして、位置データの上に、「地形情報や、その場所の明るさ、見通し」など位置に依存した情報を埋め込む方法を「世界表現」(World Representation)という。「世界表現」を最もうまく使ったゲームの一つとして『Killzone』があり、『Killzone』には、各点から 8 方向に対する見通しの距離のデータが埋め込まれオリ、AI はこのデータを基に射線、火線を高速に計算することが出来る[38] [39]。

④ 環境、知性、身体の相互関係の発展

2008 年までに、①「エージェント・アーキテクチャ」②「パス検索技術」は、米のゲーム AI 開発者の中で知識が共有され吸収されて来た。Halo を代表例とする、ビヘイビア・ツリーの方法も広く知られるようになった。③「知識表現」の重要性もまた、指摘され続けている[16]。2008 年度までに、キャラクターAI の主要な技術の基本が出揃った感がある。欧米のゲーム AI の開発には、共通の知識基盤の認識が形成されたのである。

そして、現在では、このフレームの中で各要素と、各要素間の関係が精緻化され、次の段階「より人間らしい AI」へ向けて移行しようとしている。以下に上記の①～③を踏まえて、これまで説明して来た全体像をまとめながら、そういった詳細へ踏み入ることとする。

環境 まず、ゲーム世界の環境は 2D から 3D になり複雑化すると同時に、AI は、ステージにあるオブジェクト・道具を使用することを要求されるようになった。そのため、まず、環境を AI が解釈できるデータに一度変換しておく必要があった。オブジェクトに

は、そのオブジェクトの特性と、そのオブジェクトに対して AI が取れるアクションを付加した情報が作成された。これが「知識表現」の技術であった。また、AI がマップ全体を移動できるためには、まずマップを AI が検索できるグラフデータ、具体的には、ウェイポイントか、ナビゲーション・メッシュデータが地形に沿って作成された。これを、「世界表現」の技術である。このように、環境そのものが抽象化され、AI が世界をシミュレーションする能力を手に入れた。

知性　知性は、まず第二期において FSM や複雑なロジックによって構造化され、さらに、その意思決定部分には、プランニングなどの新しい技術が導入され、時間・空間をシミュレートする能力を獲得して来た。知性は、環境に対して自身を守ること、身体を維持すること、そしてゲームの目的（通常はプレイヤーを撃退する）を達成するために、感覚によって環境から情報を受け取り、意思決定を行う。第三期の特徴として、知性は環境から受け取った情報から一端、記憶として情報を蓄積するようになった。C4 アーキテクチャでは、WORKING MEMORY 内に記憶が蓄積される。F.E.A.R.では、ゲーム世界から得た情報を単一のフォーマットでタイムスタンプを付けて管理している。

知識の構造化は、身体との連携においても為される。即ち、知性から身体への命令はどう為されるべきか、という問題である。これには、二つのアプローチがあり、知性が身体へ直接、細かな動作の指示を与えるパターン、もう一つは、知性が簡単な動作指示情報をブラックボードや共有メモリに置いて、具体的な細かな動作は身体がそれを解釈して実際の運動を行う、という方法がある。C4 アーキテクチャは後者のアプローチである、一般的に「大規模になればなるほど機能の分離は汎用性をもたらす」ので、身体構造が複雑であれば後者のアプローチが優れている。小規模な AI ならば、前者のアプローチの方が細かな制御まで知性から指定することが出来て優れている。知性と身体で処理の持ち合うバランスが重要である。

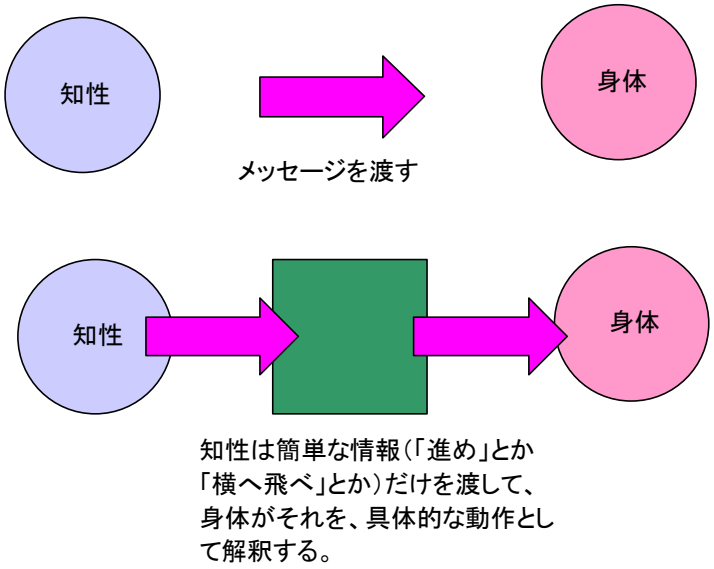
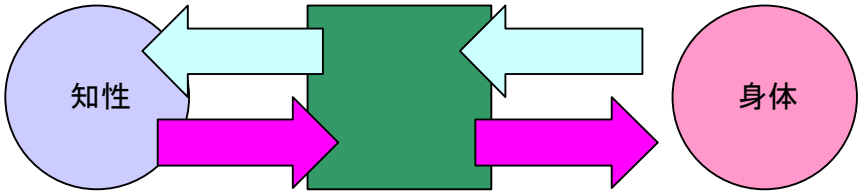


図 3.2-52　知性と身体との関係

知性から身体を制御する場合に、制御情報の受け渡し。前者では、知性が細かな制御まで行う。後者では、抽象的な支持を身体自身が詳細な運動へと解釈する。前者ではスケールが大きくなれば、知性のコーディングが肥大化し、後者では、それを分担して行う形になる。

身体　身体の構造は、スプライト表現から 3D のボーンベースに変化し、さらにボーンの数や、プロシージャル・ボーンなど、ボーンの構造も複雑化すると同時に、IK（inverse kinematics）など地面との接触を考慮した制御が導入されて来た。このような背景には、キャラクターの動作アニメーションを自然に近付けたいという目的がある。身体と環境との相互作用は主に衝突判定を用いて為され、知性と身体の相互作用は、前述したように、二通りの方法で為される。身体と知性の関係において最も大切なことは、その知性が身体とどれぐらいフィットしているか、ということである。人間の知性は「自分の身体を自由に動かせるという全能感」のもとに行動を計画している（これは決して自明なことではない）。AI の場合も、自身の身体の形状、運動を把握した上で行動を定義することが理想である。例えば、「自分の身体はジャンプできるのか」、「うつ伏せになったまま進行できるのか」、「武器を持っていない手でロープをつかめるのか」、と言った情報は、あらかじめ知性が知識として持ち、そこから動作プランを立てるべき情報である。また、そういった静的な情報と同時に、現在の自分の身体状況をリアルタイムに把握していれば、より精緻な行動プランを立てることが出来る。環境のみならず、身体からも知性へ身体の情報が渡される必要があるのである。これは、動作させているアニメーションの再生情報と知性との間に、より細かな情報のやり取りを構築することで実現する。



知性は簡単な情報(「進め」とか「横へ飛べ」とか)だけを渡して、
身体がそれを、具体的な動作として解釈する。

図 3.2-53　知性と身体の双方向の情報伝達

知性は身体に制御情報指針を与える、身体は現在の状態を知性に伝える。動的な場合は、今はどういう動作をしている途中であるか、或いは、どういう動作ができるか、などを身体から与えるというケースが考えられる。静的には、事前に、身体の特徴、身体の運動特性を知性に知識として渡しておいて、身体の特徴と運動特性に応じたパスを返してもらうケースなどが考えられる。

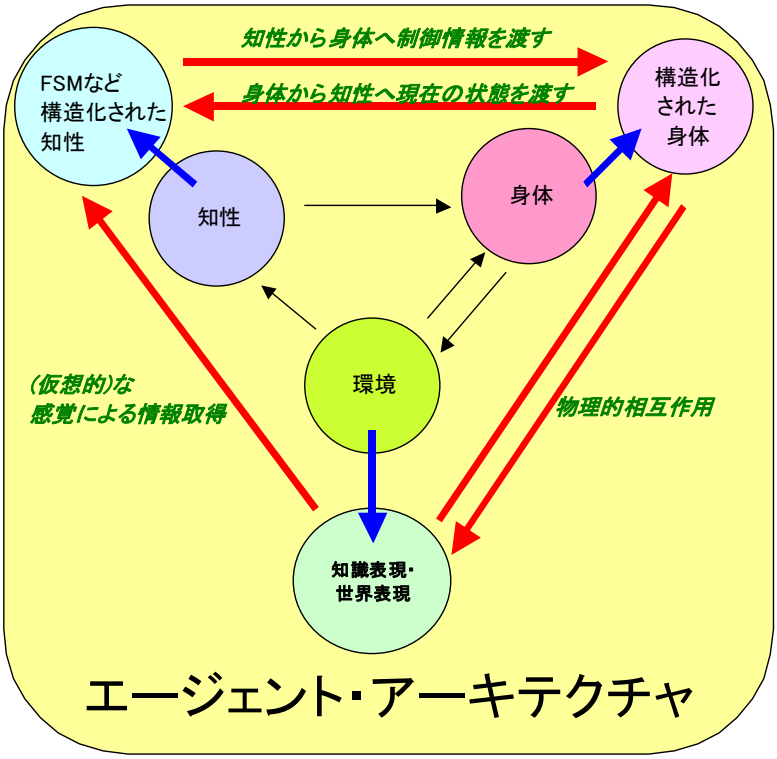


図 3.2-54　身体、知性、環境の相互作用のフレームはエージェント・アーキテクチャへと形式化される

身体のパターンから始まり、環境に対する対応、身体の構造化、身体と環境との物理的インタラクション、身体と知性の相互伝達、こういった経緯を含んでキャラクターAI は、身体、知性、環境、三者の相互作用システムとしてアーキテクチャ化される。

⑤　環境、知性、身体の階層化

知性と環境における情報が複雑で膨大なものになるにつれ、「AI のシステムを階層化する」というアイデアが広く応用されるようになった。AI が処理するべき情報は、空間が広がりオブジェクトが増大し、同時に移動にかかる時間スケールも大きくなり、局所的な戦闘の情報から、大局的な戦略まで、異なる時間点空間スケールの情報について思考する必要に迫られるようになった。そして、多くの場合「局所的な情報は短い時間で必要とされ」「大局的情報は長い時間幅で必要とされる」ので、時間スケールと空間スケールは比例する。この性質を使って、情報と思考の階層化が可能に成る。

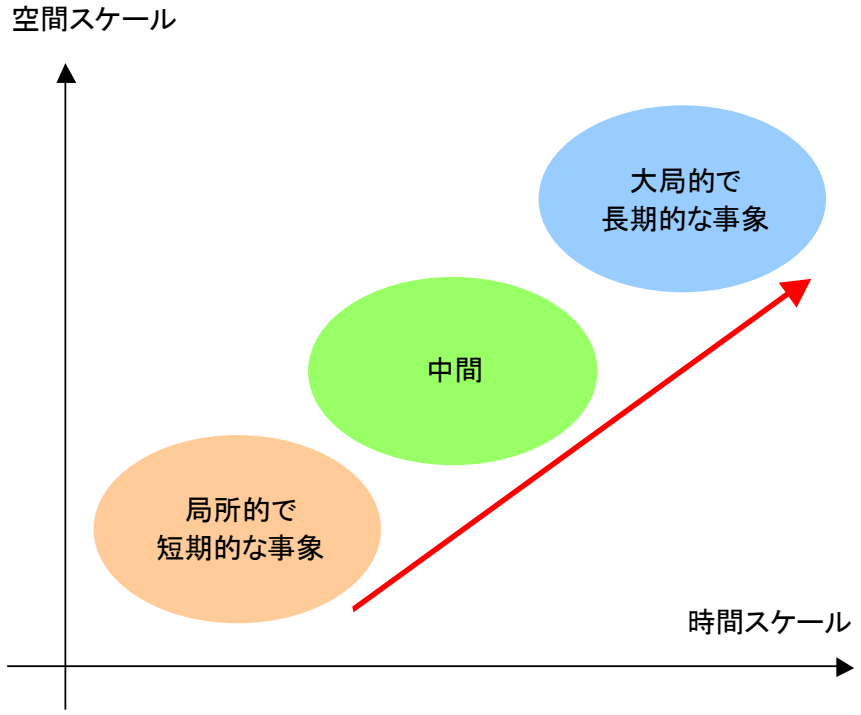


図 3.2-55　AI が処理するべき事象の空間と時間のスケール [8]

まず、世界表現における情報の階層化を考えてみる。『HALO2』では、移動のためのデータを「エリア」「プレイグラウンド」「ウェイポイント」のように階層に分けて管理している[36,37,41]。そして、それぞれの層に応じて、戦略、戦術、単純な移動、という知的判断が対応している。このような階層化は各層の思考をモジュール化し、知性の設計に拡張性をもたらす。例えば、ある層の思考を独立に取替えたり、改善したり、容易に追加することを可能とする。

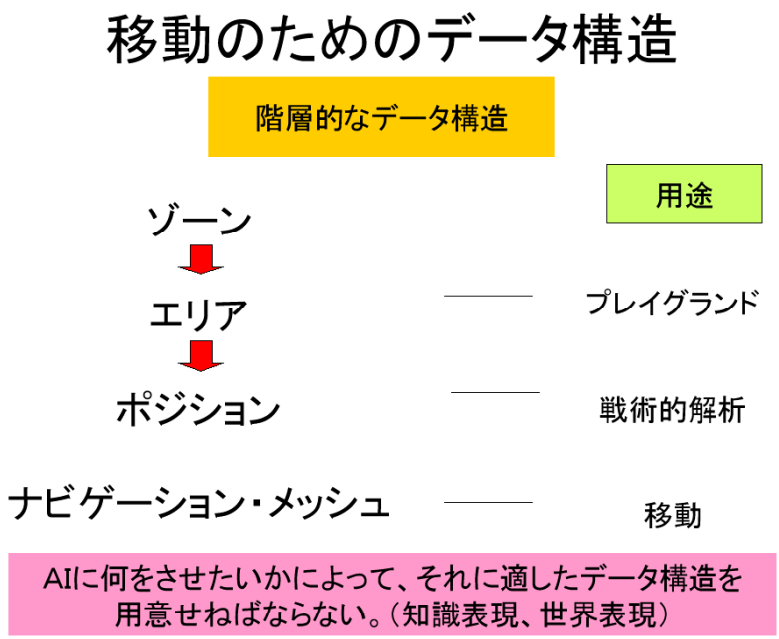


図 3.2-56　『HALO2』における世界表現のデータの階層化と思考の階層化の対応[36,37,41]

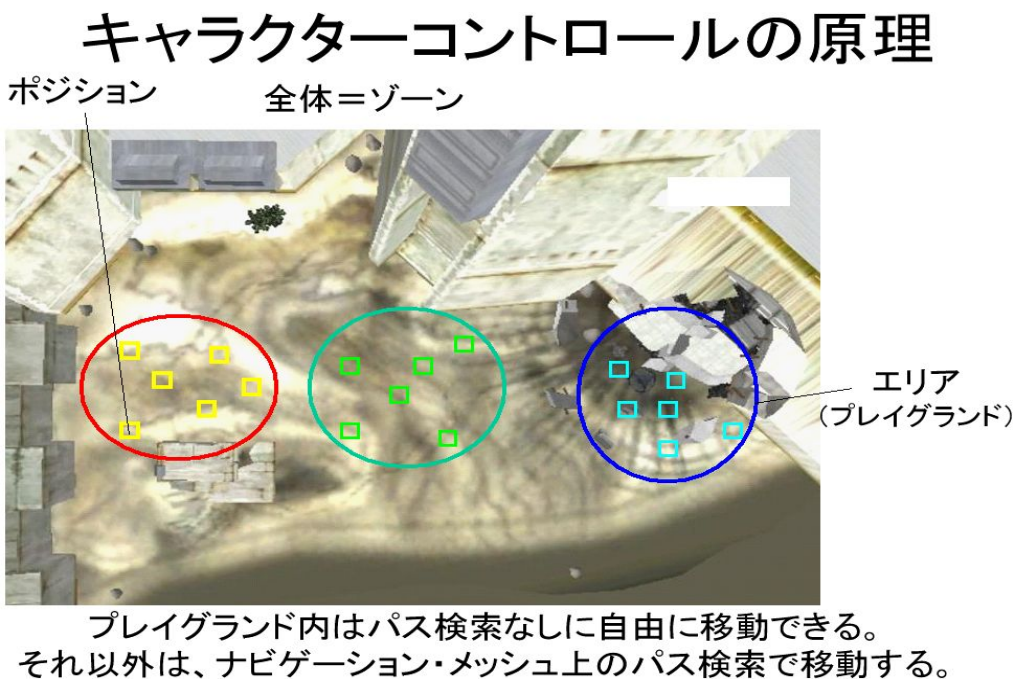


図 3.2-57　『HALO2』における世界表現のデータの階層化[36,37,41]

(5) 2009 年のキャラクターAI （GDC2009 ゲーム AI 分野レポート）

2009 年の GDC では、2 日間に渡る連続セミナー「AI Summit」が開催され、この数年で育まれたゲーム AI 技術（パス検索、プランニング、エージェント・アーキテクチャ、知識表現、キャラクター制御）が、開発者自身によってレビューされると同時に、これからの方向性について話し合われた。大きな方向性としては、「キャラクターのアニメーション動作と身体特性を精緻に反映させた行動プランを組み立てる」という方向であった。これまでは、パスが決定されれば、AI はパスを辿る動作をする、というだけでよかった。しかし、そういったパスフォローイングは、そもそも AI の身体的特性を考慮せず、グラフ探索アルゴリズムで探索されたパスに過ぎない。歩幅を考慮したパス検索、階段に自然にかかとが着地するアニメーション、人間がたどるような曲線のパスを見つけるアルゴリズムなど、さらなるリアリティを求めて、キャラクターの身体動作と AI の意思決定、環境を調和した行動を目指して、幾つかの技術が発表された。

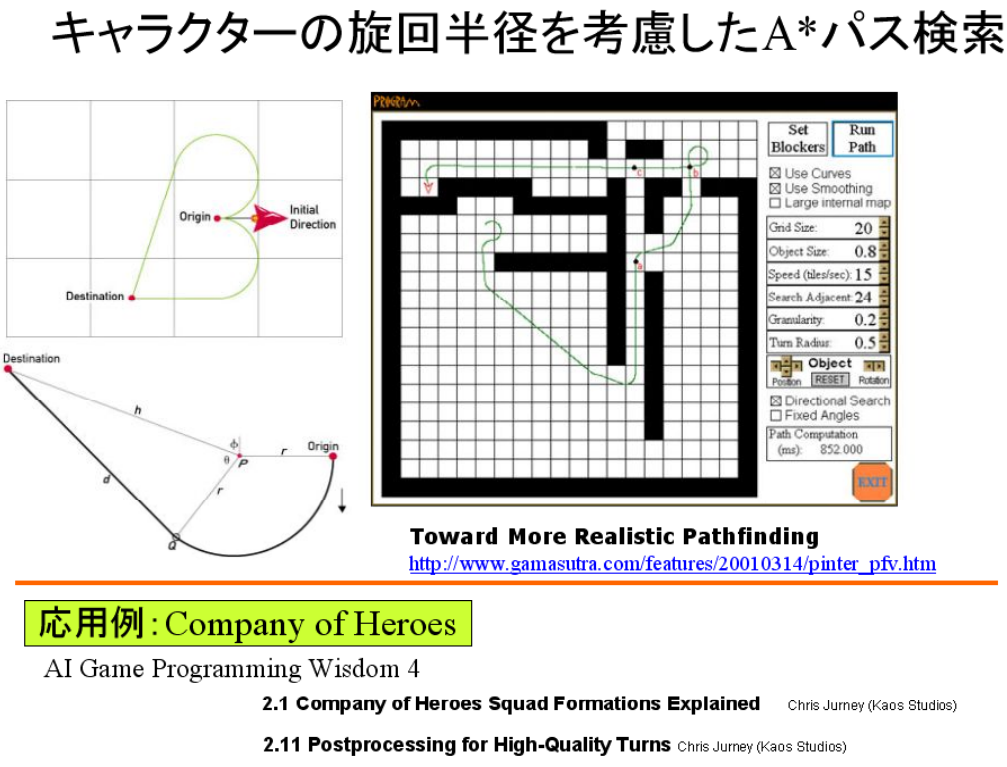


図 3.2-58 キャラクターの旋回半径を考慮したパス検索

大型の車などキャラクターによっては直角に曲がれないので、旋回半径を超えたパスをパス探索の段階でカットしてしまう。例えば上図では、直線に行った方が最短のパスであるにもかかわらず、旋回半径を確保するために遠回りをするパスを導いているのがわかる[42]。

以下に、まず身体制御技術について Autodesk、Natural Motion、Unity の発表を紹介する。全てミドルウェア会社の発表であり、これは身体制御分野が、次世代機開発の第二フェースにおいて焦点となり、その部分をミドルウェアとしてサポートする動向が見えて興味深い。Havok AI に関しては講演という形での発表がなされなかった。

① 講演「Character Pathfinding And Animation: How to Make Them Work Together」
Pierre Pontevia (Senior Director of Product Development, Games Technology Group, Autodesk), Robert Lanciault (Autodesk Inc.)

本講演は製品ではなく、研究成果の発表である。主に二つのことが発表された。

モーション・ブレンディング 様々なモーションは、基本モーションをパラメーターによってブレンドすることで実現することが出来る。例えば、曲がるモーションは、直進するモーションと直角に曲がるモーションの重ね合わせで実現することが出来る。また、スピードは、スピードのモーションは、「歩く」と「走る」のパラメトリックなブレンドによって実現することが出来る。このように、基本モーションをどう組み合わせるかは、ノードグラフの GUI を使って、デザイナーが指定出来るようにしている。

ステアリングの新しいアプローチ ステアリングとはキャラクターの身体制御のことである。ステアリングは普通、発見したパスをフォロー（沿って進む）ことで決定される。つまり「パスを発見し」(Pathfinding)「パスをスムージングし」(Path Smoothing)「パスをフォローしながら、突発的なオブジェクトは自動的に避ける」(Dynamic Avoidance)という 3 つの過程を取る。しかし、こう行ったアプローチでは、

- モーションを考慮しないステアリングのベクトルが決定される。
- アニメーションの遷移が必要である場合に、その遷移時間が考慮に入れられていない。
- 新しいアニメーションがステアリングのベクトルに合っていない。

という問題が起こる。そこで、AI の身体制御には次のような方法を取るべきである、と提案する。

- 歩行モーションとそのスピード
- ステアリング力とスピードの関係
- スピードと旋回半径の関係
- モーション遷移にかかる時間

・ 旋回半径限界
・ 停止に必要な距離
・ 現在の足を置いている場所

などの情報を考慮してパスを組み立てることで、そのキャラクターの運動特性に応じたパスを構築し自然な運動を実現する。理屈だけ聞くともっともな議論であるが、これまで暗黙の内に無視して来た、「知性」と「身体の運動特性」の間の関係を明示的に解決するアプローチであり、非常に先進的な成果である。講演で提示されたデモは、AI が自然な曲線を描いて走っている運動が見られ、研究段階とはいえ、たいへん有望な技術である。

② 講演「Authoring Runtime Animation and Character Physics with Morpheme 2.0」
Simon Mack (CTO, NaturalMotion Ltd)

Morpheme 2.0 は、キャラクター・モーションを編集から、実際のゲーム機（Playstation 3, Xbox 360, Wii and PC）のライブラリによるランタイム環境までサポートする、キャラクター・モーションのための総合ミドルウェアである。NaturalMotion 社は、endorphin, euphoria とプロシージャル・アニメーション（手続き的アニメーション自動生成）のシステムを世界に先駆けて開発して来た先進的な企業である。Morpheme は、その基盤(euphoria)の上アニメーション編集と実機におけるランタイムを提供した形である。講演では、FSM による実際のアニメーション遷移の指定とランタイムにおける滑らかな補間、ステージ上のダイナミックな回避のデモなどが提示された。

③ 講演「Dynamic Walking with Semi-Procedural Animation」
Rune Skovbo Johansen (Creative Programmer, Unity Technologies)

解説は本報告書の「GDC2009 とゲーム技術の研究開発」の「セミ・プロシージャルというアプローチ」を参照のこと。

①～③の講演からわかることは、ゲーム環境内で、キャラクターの身体動作を自然に見せるために、身体を考慮したパス決定や、環境に対して自然な動作を実現するための身体システム自身を構築するという方向に開発環境を整備しようとする動きがあることである。

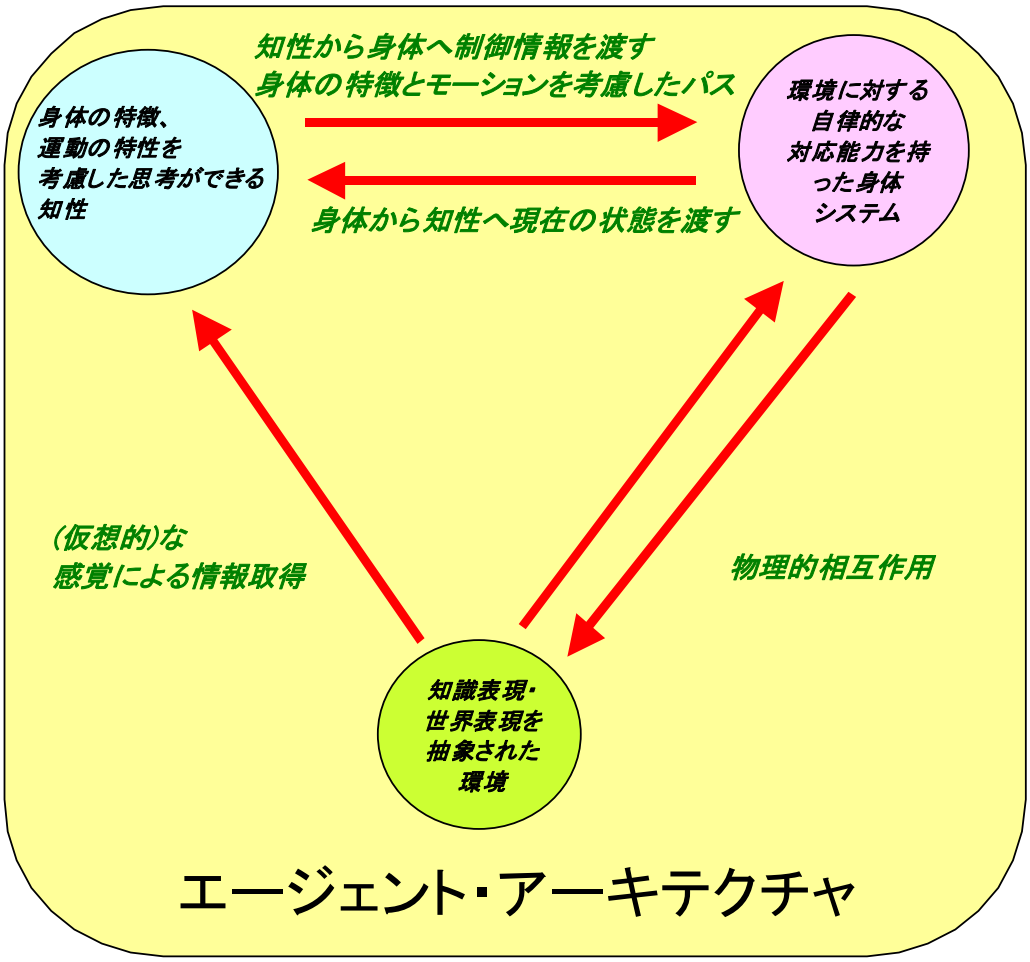


図 3.2-59　エージェント・アーキテクチャはさらに、各要素の相互関係を深めて、より高い知性体へと発展して行く。

これから身体と知性のもっと深く絡み合い、より生物に近い身体を獲得する AI のために、その身体にふさわしい、身体の特性を考慮できる知性が開発されて行くことだろう。

(6)　メタ AI

メタ AI とはゲームの進行を動的に調整・変化させる AI のことである。従来であれば、ゲームを進行させるのは、ゲームシステム、ゲーム・メインロジックそのものであったわけだが、ゲームの大規模化、或いはゲームデザインの進化の方向として、動的にゲームを調整する方法、或いは変化させる方法が本格的に導入され始めている。この傾向は、2009 年の GDC で特に顕著な特徴であった。

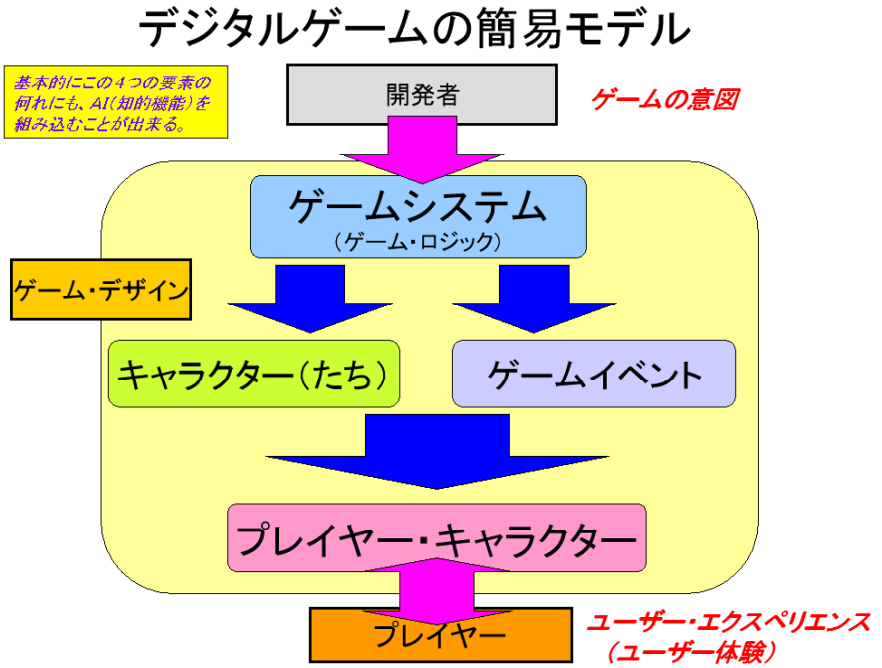


図 3.2-60　デジタルゲームの簡易モデル

ゲーム環境世界を通して、開発者からプレイヤーへ、ゲームシステム、キャラクター、イベントなどを通して、プレイヤーに対してゲーム体験（ユーザーエクスペリエンス）が提供される。全体を統御するのがゲームロジック(ゲームシステム)である。

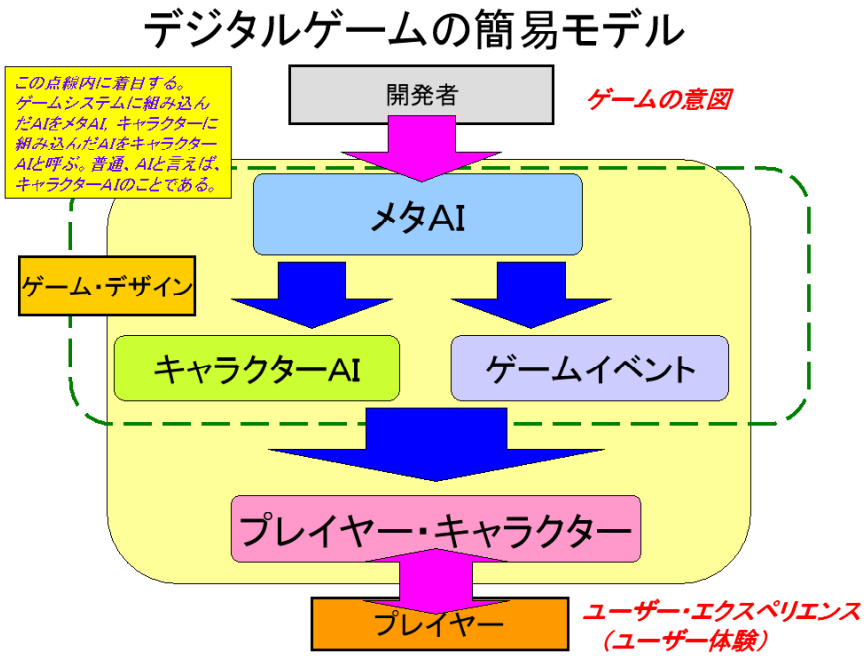


図 3.2-61　ゲームロジックからメタ AI へ

ゲームロジックの部分がメタ AI に変化し、動的にプレイヤーの状態やゲームの状況を見ながら、キャラクターAI やゲームイベントに指示を出して行く。方向としては「ゲーム自体が一つの知性として活動する」という方向である。

メタ AI はゲーム全体を俯瞰し調整する AI である。現在、知られている応用例は、主に 4 つに分類される。

① 動的にプレイヤーのレベルにゲームの難易度を調整するレベルコントロール AI
② ゲームのバランスを自動する調整 AI
③ プレイヤーを楽しませるために、プレイヤーの心理を解析し、ゲーム内容を積極的にコントロールする AI
④ 地形を解析しキャラクターやオブジェクトを配置するプロシージャルな AI

である。以下、順にこれらを説明して行く。

(a) レベルコントロール AI

レベルコントロール AI は主に日本で発展した技術である。「パックマン」「ゼビウス」を始めとして、シューティング・ゲームやアクションゲームなど、プレイヤーのスキルに大きな開きのあるゲームの隙間を埋める機能を持つ。公開されていないものでも、膨大な例があると予想される。

以下に二つの文献から引用する。

岩谷徹氏：　ファンと一般のユーザーを満足させる方法のひとつは人工知能 AI のような考え方です。プレーヤースキルをプログラム側から判断して、難易度を調整していくというものです。これを私はセルフゲームコントロールシステムと呼んで 10 年以上前から開発に使っています[6]。

－ International Game Designers Panel －
http://game.watch.impress.co.jp/docs/20050312/GDC_int.htm

遠藤雅伸氏　あと面白い機能なんですけれど、ゼビウスには非常に簡単な AI が組み込まれています。「プレイヤーがどれくらいの腕か」というのを判断して、出てくる敵が強くなるんです。強いと思った相手には強い敵が出てきて、弱いと思った相手には弱い敵が出てきます。そういったプログラムが組み込まれています。ゲームの難易度というのは「初心者には難しくて、上級者には簡単だ」ということが、ひとつの難易度で(調整を)やっていくと起きてしまうので、その辺を何とか改善したいな、ということでそういったことを始めてみたのですけれど、お陰で割合にあまり上手くない人でも比較的長くプレイできる、うまい人でも最後のほうに行くまで結構ドラマチックに楽しめる、そういった感じになっています。

－　ゼビウスセミナー　－

http://spitfire.client.jp/shooting/xevious2.html

このように、レベルコントロール AI は、幅広いプレイヤーの要求に応えるために、既に 1980 年代の初めから導入されて来た技術である。

(b)　ゲームのバランスを自動調整する AI

バランス・コントロール AI は主に、最初からゲームのコントロールが難しい場合、ゲームの進行を見ながら動的に調整する必要がある場合に多く用いられる。AI と言うほど大きくないものでもコードレベルで実装されているものも多い。

最近では、プロシージャル技術のように、自動的に進行する要素がゲーム内に組み込まれた場合に、その自律的発展をゲーム内のバランスを崩さない範囲になるように調整する役割を果たす。例えば、『アストロノーカ』(muumuu、1999)では、遺伝的アルゴリズムによる AI の自律進化のスピードを調整する。GA のためのシミュレーションでは、ユーザーの行動によっては、一ターンで十分な進化が達成されないことや、或いは、逆に急激に進化してしまうことがある。そこで、AI の進化が一定になるように、GA を適用する回数を調整する[23]。

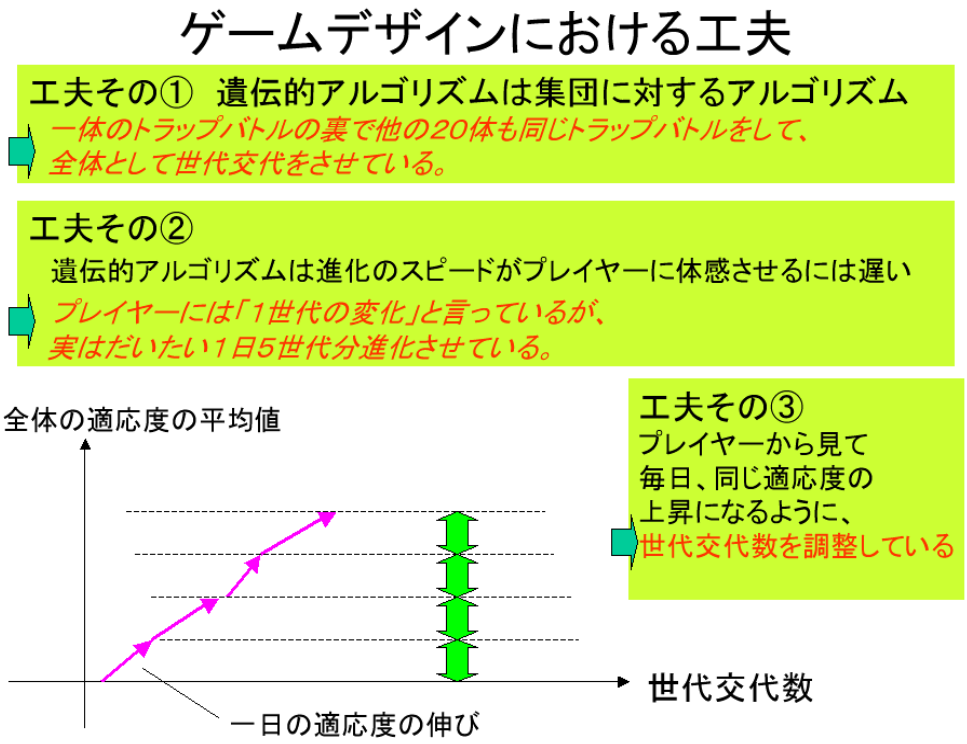


図 3.2-62　『アストロノーカに』おける自律的な進化バランス調整[24]

(c)　ゲームを積極的に動的にデザインする AI

①「レベルコントロール AI」も②「バランスを取る AI」も、ゲームの黎明期からゲームのロジックに含まれていることが多い AI であり、これだけであれば、メタ AI が注目を集めることはなかったであろう。これらは出来上がったゲームを動的に調整する、と言う役割を持った、どちらかと言えば消極的なメタ AI である。しかし最近では、「プレイヤーの状態、心理を解析して、より積極的にゲームのイベント発生まで踏み込んでコントロールする」第三のメタ AI の使用法が展開されている。2009 年 5 月 14 日の GDC で発表された、「LEFT 4 DEAD」(Valve Software, 2008)の AI Director がその典型である。

AI Director は、プレイヤーの緊張度をあるモデルによって計算する。その緊張度を、緩急のリズムを持つように、タイミングよく AI の群れをプレイヤー・パーティにぶつけるのである。そうすることによって、緊張感の波を形成し、ゲームの面白さを人工的に作り出す。また、毎回違ったゲームプレイをユーザーに提供することを可能にし、ステージが少ないゲームにおいても、くり返しプレイ出来るゲームにすることが出来る。

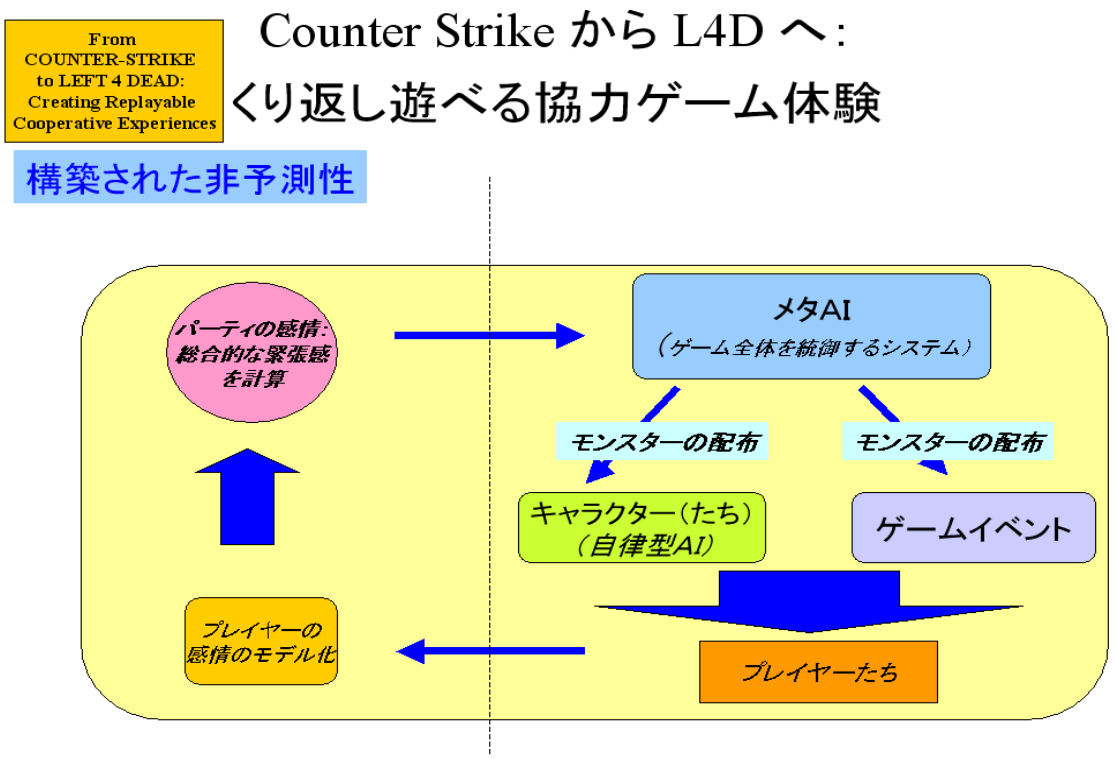


図 3.2-63　「LEFT 4 DEAD」の AI Director のモデル[43,44,45,46,47]

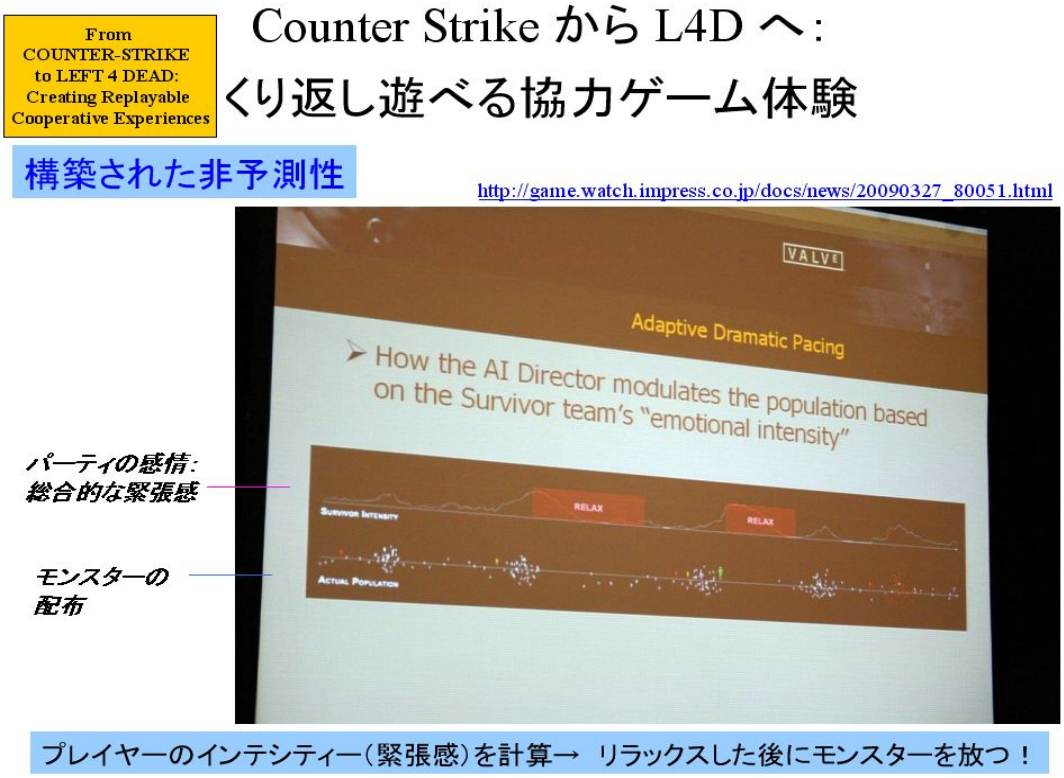
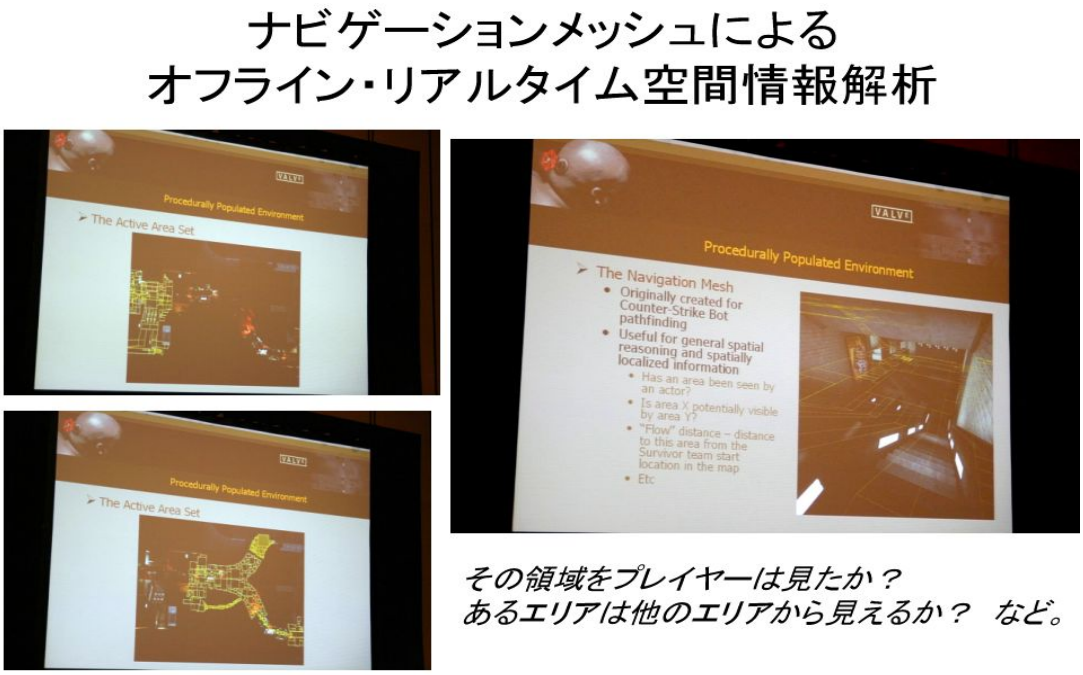


図 3.2-64 「LEFT 4 DEAD」の AI Director がリアルタイムにモニターするプレイヤーの緊張感[44]

注意しなければならないのは、ゲーム全体を見渡して全 AI を制御する、所謂「神 AI」と、メタ AI は異なるものだということである。メタ AI は、個々の AI は、それ自身のロジックや知能を持っており、その出現や間接的なパラメーターを操作するだけである。つまり、上から制御する部分と、実際にプレイヤーに接するキャラクターAI の、役割は明確に分担されている。一方、「神 AI」というのは、全体を俯瞰しながら局所戦闘も制御してしまうもので、オールインワンのシステムになりがちである。逆に、メタ AI の設計は、小さければ小さいほどよい。キャラクターAI は自律した思考を持ち、行動の指針だけがメタ AI から与えられる。プレイヤー近くの行動に関しては、キャラクターAI の自律型思考の方が効率よく、ゲームの俯瞰的思考にはメタ AI が適している。それぞれの役割をお互いが果たすことで、相乗的な効果を実現するべきである。



http://game.watch.impress.co.jp/docs/news/20090327_80051.html

図 3.2-65　「LEFT 4 DEAD」の AI Director がリアルタイムにモニターするゲーム状況[44]

(d)　ヒートマップ

メタ AI をサポートするデータとして、マップ上に位置に応じた統計情報をサーモグラフ化した「ヒートマップ」と呼ばれるデータが存在する[48]。例えば、『Halo3』では、オンラインを通じてプレイヤーから集めた「プレイヤーが何処でどのようにダメージを受けたか、或いは戦闘不可能になったか」という統計情報マップがサイト上で公開されている[49]。赤い場所ほど、統計数が多い場所となっている。こうったデータを解析することで、メタ AI は、敵の出現場所やタイミングを、より計算してコントロールすることが出来るようになる。ただ『Halo3』では、現在のところ、レベルデザインの検証に用いられるか、ユーザーに提供されるデータとしてのみ使用されている。

図 3.2-66　Halo3 における、あるステージでのプレイヤーの死亡頻度マップ（ヒートマップ）。殺傷武器別のヒートマップも描画できる[49]。

図 3.2-67　TEAM FORTRESS 2 における、あるステージでのプレイヤーの死亡頻度マップ（ヒートマップ）[48]

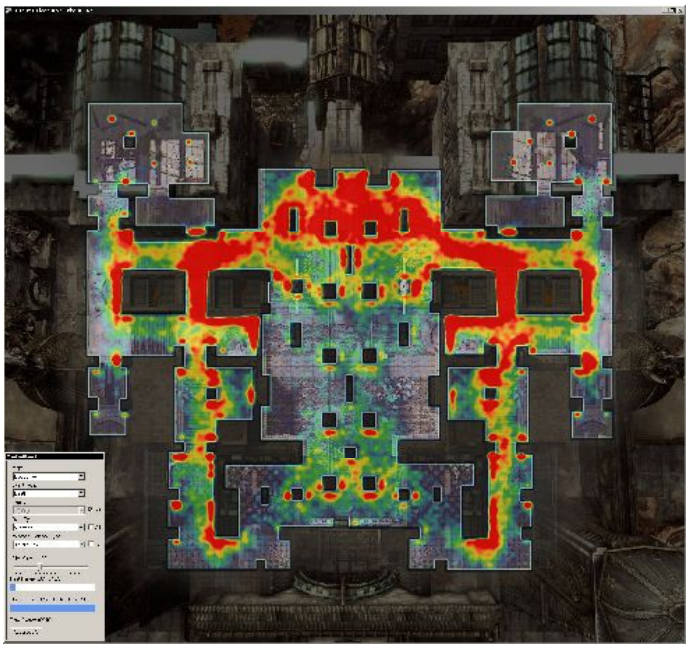


図 3.2-68　Unreal Master Control System におけるヒートマップ 「Unreal Engine 3」 では、ヒートマップ生成機能もサポートされる[50]

(e)　地形を解析しキャラクターやオブジェクトを配置するプロシージャルな AI

Age of Empire (以下、AOE)は地形をプロシージャルに生成する。しかし、それだけではなく生成した地形を解析して、オブジェクトや AI を配置する場所をインフルエンス・マップを用いて決定し配置する。これを単なるアルゴリズムを見なすこともできるが、メタ AI と見なすことも出来る。自身が生成した地形を解析するということは、生成した地形を単にポリゴンタイルの集まりではなく、性質を抽出して「知る」ことで、動的に配置場所を決定しているからである[51,52]。

キャラクターAI がゲーム環境世界の内側の一番内側にいるとすれば、メタ AI はゲームの外からゲームを動的にコントロールする AI である。二つの AI を自律的、かつ連携的に動作させることで、ゲームは新しい進化を遂げる可能性を持っているのである。

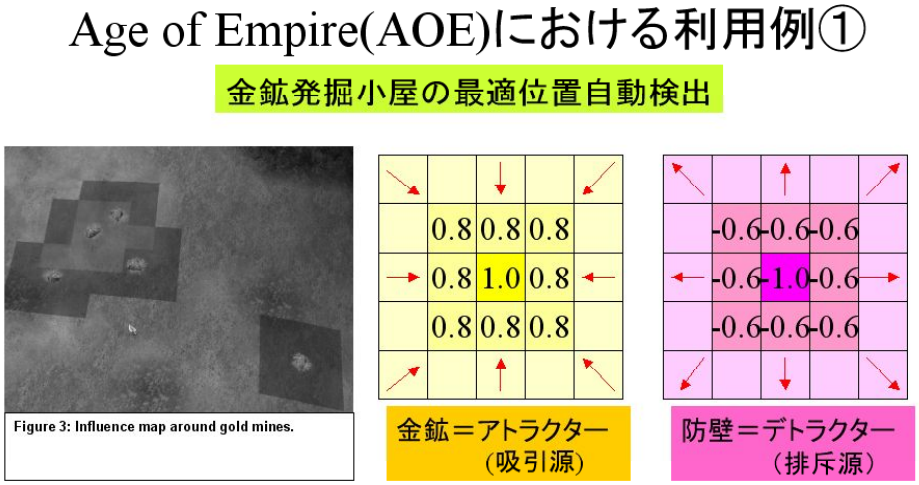


図 3.2-69　AOE における自動オブジェクト位置決定システム[51,52]

インフルエンス・マップとは、タイルを単位として、中心のタイルからどれぐらい離れるかで、影響力を減衰して行くマップであり、ここでは、アトラクター、デトラクターの２種類の影響の分布を使っている。ここでは、金鉱の発掘小屋を配置したいのだが、金鉱小屋はもちろん金鉱に近いほどよいのだが（金鉱を中心にアトラクターを置く）、あまり近くに置くと金鉱へのパスが取れなくなってしまうので、金鉱の中心から一マス置いたところから、影響が始まるようにしている。

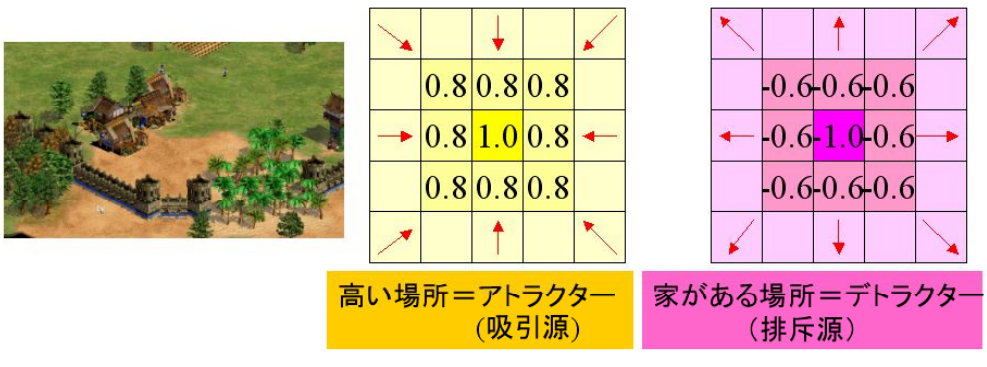


図 3.2-70　AOE における自動兵士配置システム[51,52]

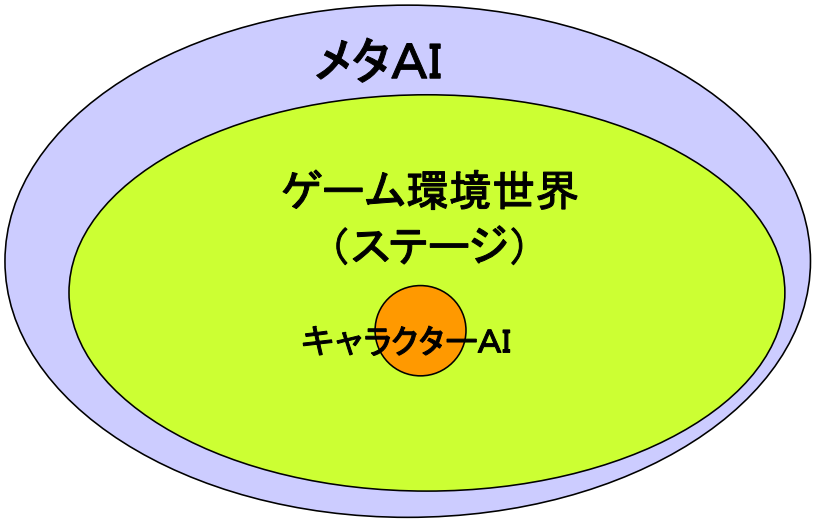


図 3.2-71　メタ AI，ゲーム環境世界（ゲームステージ）、キャラクターAI の関係

キャラクターAI は、環境に適応した身体、知性を持つ必要があり、ゲームに限定されるので、ゲーム環境に特化した最も内側に存在する。一方でメタ AI は、ゲーム環境世界の外側から、ゲーム環境世界とキャラクターAI を制御する。この二つの AI を自律させ連携させることで、新しいゲームデザインが産まれるのである。

(7)　これからの AI が向かう方向

(a)　「身体－環境－知性」モデルの汎用性について

ここまで、「身体」「環境」「知性」という 3 つの要素に分けて、キャラクターAI、メタ AI を解説して来た。具体例に挙げたゲームはマッシブなアクションゲームが多いが、ゲームジャンルを限定して説明しているわけではなく、最も分かりやすい例を挙げて説明しているに過ぎない。例えば、シミュレーションゲームでは、プレイヤーは直接的な身体を持つわけではないが、自分が動かす兵士全体を身体と捉えてもよい。パズルゲームでは、環境がないだろうか？　むしろ、そこではパズルの要素とその運動の規則が環境である。「純粋な対戦ゲームでは AI がないのではないか？」という問があるかもしれないが、プレイヤーの位置を AI で置き換えて考えることも出来るし、また、メタ AI が介在する対戦ゲーム、或いはプレイヤー以外の第三者としての AI（例えばプレイヤーがそれぞれある種族を操作するとして、第三の種族として AI が介入する、など）が存在する場合もある。

囲碁や将棋の AI はどう捉えればよいだろうか？　囲碁や将棋はリアルタイムのゲームではなく、そこにはターンという概念によって時間が凍結されているゲームである。そして、盤＝環境であり、離散的なマスによって分けられている。AI の身体は駒そのものであり、駒にはそれぞれ特性があり、その特性を考慮して AI は駒を動かさないといけない。囲碁や将棋における AI は殆どの場合、プレイヤー（棋士）の代わりとしての存在である

が、駒をそれぞれエージェントとして、複数の駒からなるマルチエージェント・システムとして将棋や囲碁の AI を構築することも可能だろう。

このように、ここで展開したフレームは、それぞれ自分が考えたいゲームにおいて、柔軟に適用して応用して頂ければと思う。

(b) キャラクターAI の進化の方向

キャラクターAI の歴史は、ゲームからの自律化の歴史そのものである。長い時間を経て、キャラクターAI はゲームロジックから独立すると同時に、「ゲーム世界＝環境」との関係を再定義して来た。そして、複雑化する自身の身体構造と運動を、うまく環境に適応して用いるために、身体の特徴と運動を取り込んだ思考をさらに発展させて来た。このように見ると、キャラクターAI の進化は人類の進化とよく似ている。

キャラクターAI の次の進化のステージは「学習」と「予測」である。

ゲーム世界の環境は、もはやかつての、ゲーム的都合のよい制御が入った世界ではなく、物理法則を始め、はっきりとしたゲーム世界内で普遍的な法則が入った世界である。はっきりと法則が入った世界になった初めて、予測シミュレーションが可能になる。それぞれのキャラクターごとに独自の飛行曲線が入った世界では、シミュレーション的な予測が難しい。しかし、法則が入った世界では、ある程度、簡易化したモデル・シミュレーションが可能であり、100%と言わなくても、未来の状態をある程度予測することが可能になる。

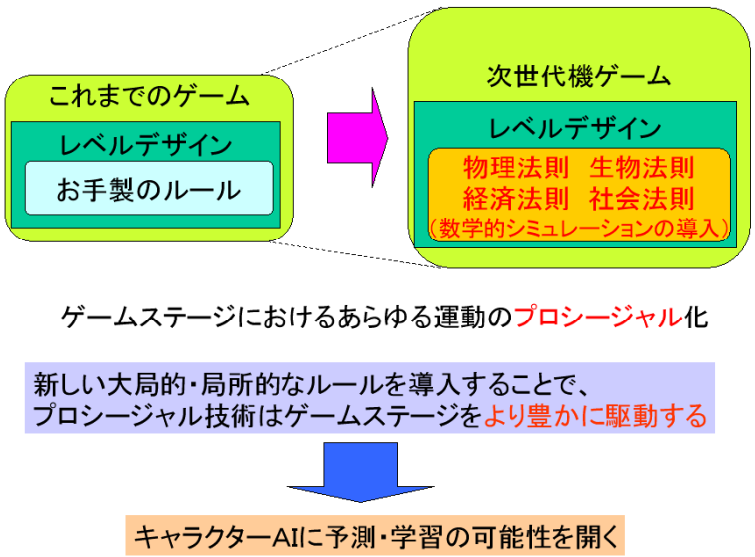


図 3.2-72　ゲームの環境世界の進化はキャラクターAI に「予測」と「学習」の可能性を拓く

次に、ゲームにおけるキャラクターの進化について考えてみよう。進化のアルゴリズムで必要な条件は、遺伝的アルゴリズムにせよ、他の最適化手法にせよ、様々に進化したキャラクターたちを公平に評価するシステムである。例えば、昆虫の羽根の形状を GA で進化させるシミュレーションを考えてみる。羽の形状の評価とは、その形状によって、どれだけ遠くまで飛べるかを判定するとする。すると、飛行の物理シミュレーションが必要である。これが、もし、飛行曲線があらかじめ定義されている世界であるなら、そもそもそこには、学習の可能性はないわけである。つまり、環境におけるシミュレーション環境があって初めて、学習の可能性が開かれる。そして、現在は、物理シミュレーション、経済シミュレーション、生物シミュレーションなど、様々なプロシージャルな法が、ゲーム世界に導入されている時代である。これは、学習という面から見れば、学習に必要なステージを準備してくれていると捉えることが出来るのである。

(c) ゲームとメタ AI の共進化

メタ AI の本質は、ゲームそのものを相対化することである。ゲームプログラム全体がゲームそのものの動かす単体ではなく、ゲームを動かす部分を、「別のプログラム＝メタ AI」が監視しながら調整を行うことである。

前節で、「キャラクターAI の歴史は、ゲームからの自律化の歴史そのものである」と述べた。そして、先に「メタ AI はゲーム全体を俯瞰し調整する AI である」と述べた。とすれば、ゲームの進化の方向の一つとして、現在はメタ AI が制御している「ゲームイベント」も自律化する方向が考えられる。イベントが自動生成し、自動進行し、自動終了するような、そんな世界である。

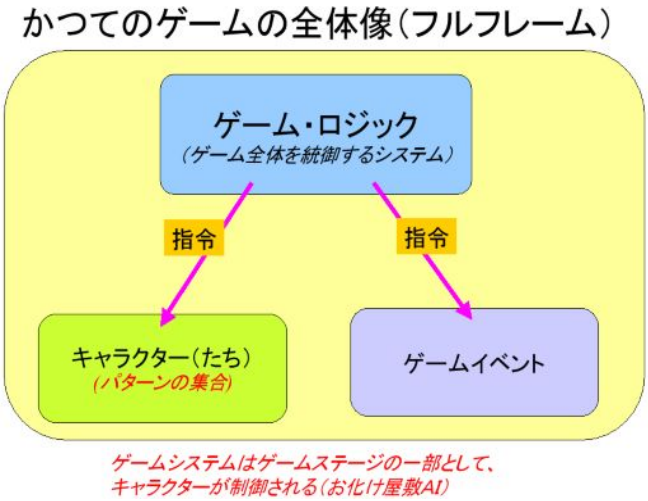


図 3.2-73　ゲームロジックが「キャラクターAI」「ゲームイベント」を完全にコントロールする設計

自律性が全くなく柔軟性に欠け、コードを書いた分だけの内容しかないため創発性に欠けるが、ゲームを完全にコントロールできるという利点がある[8]。

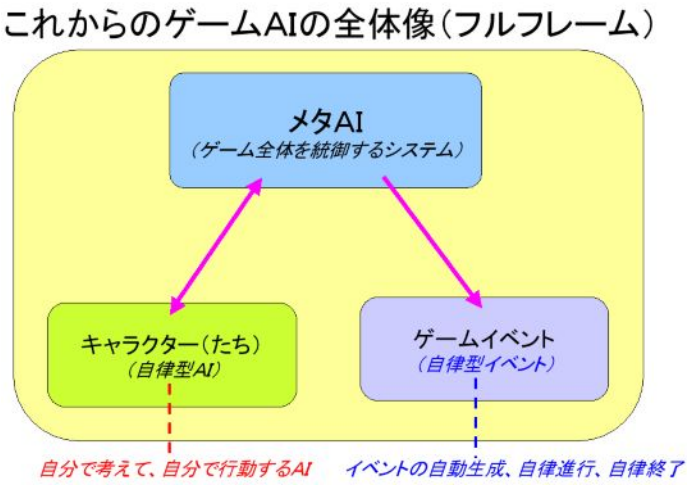


図 3.2-74　自律型ゲーム、自律型 AI

メタ AI によってゲームを一個の知性と為し、キャラクターとゲームイベント

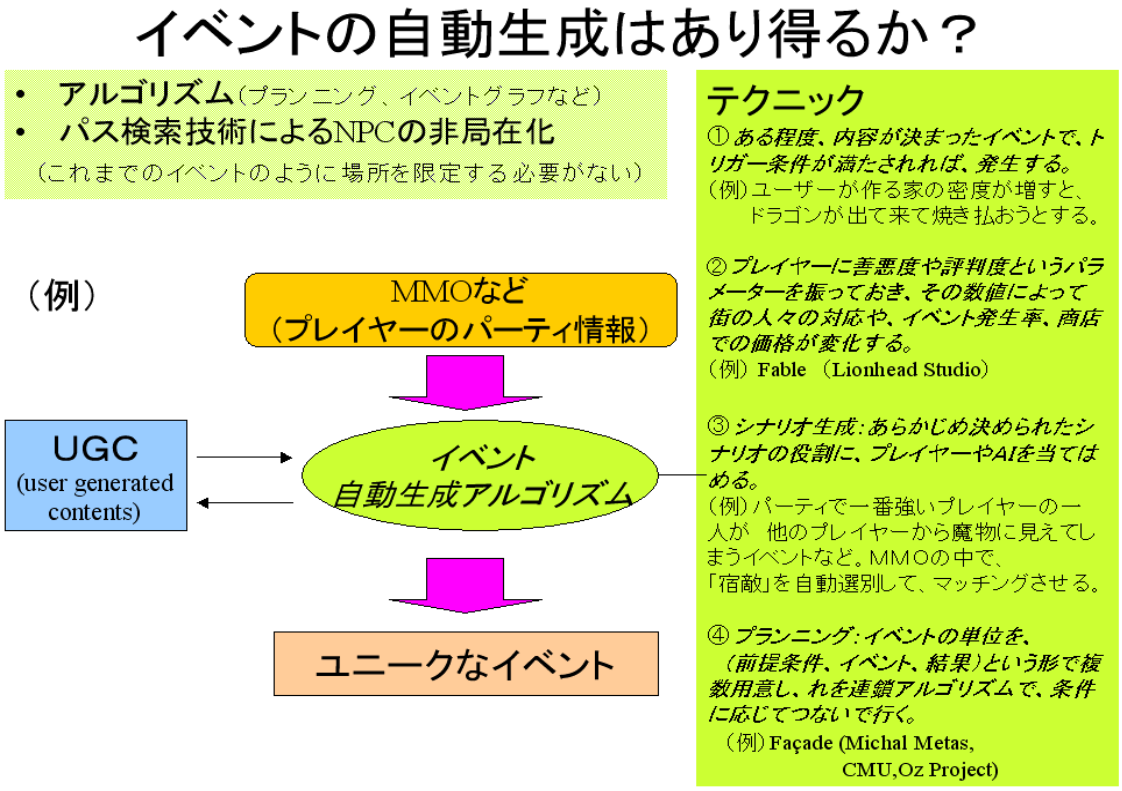


図 3.2-75　イベント自動生成に対する試論[8]

右に書いたような様々なアイデアが考えられるが、さらに、UGC とイベントを絡めることが出来れば、よりユニークなイベントを作ることが出来るようになるかもしれない。UGC をゲーム深くまで持ち込むためには、UGC の後に、一端それを解析して特徴を抽象する、セミ・プロシージャル的アプローチが必要だろう。

例を挙げれば、

① ある程度、内容が決まったイベントで、トリガー条件が満たされれば、任意の場所とタイミングでイベントが発生する。
（例）MMO でユーザーが作る家の密度が増すとドラゴンが出て来て焼き払おうとする。

② レイヤーに善悪度や評判度というパラメーターを振っておき、その数値によって街の人々の対応や、イベント発生率、商店での価格が変化する。
（例） Fable（Lionhead Studio）[53]における善悪パラメータによるイベントの変化

③ シナリオ生成：あらかじめ決められたシナリオの役割に、プレイヤーや AI を当てはめる。
（例）パーティで一番強いプレイヤーの一人が　他のプレイヤーから魔物に見えてしまうイベントなど。MMO の中で「宿敵」を自動選別して、マッチングさせる。

④ プランニング：イベントの単位を、（前提条件、イベント、結果）という形で複数用意し、連鎖プランニングシステムで、イベント単位を自動接続してイベントを自動生成する。
（例）Façade (Michal Metas, Andrew Stern, Carnegie Mellon University, Oz Project) [54]

などである。完全に自動生成は④だけであるが、半自動生成なども視野に入れれば、ゲームデザインの幅が広がるだろう。

(8) キャラクターAI の歴史年表

最後にキャラクターAI の発展の歴史を年表形式にまとめておく。

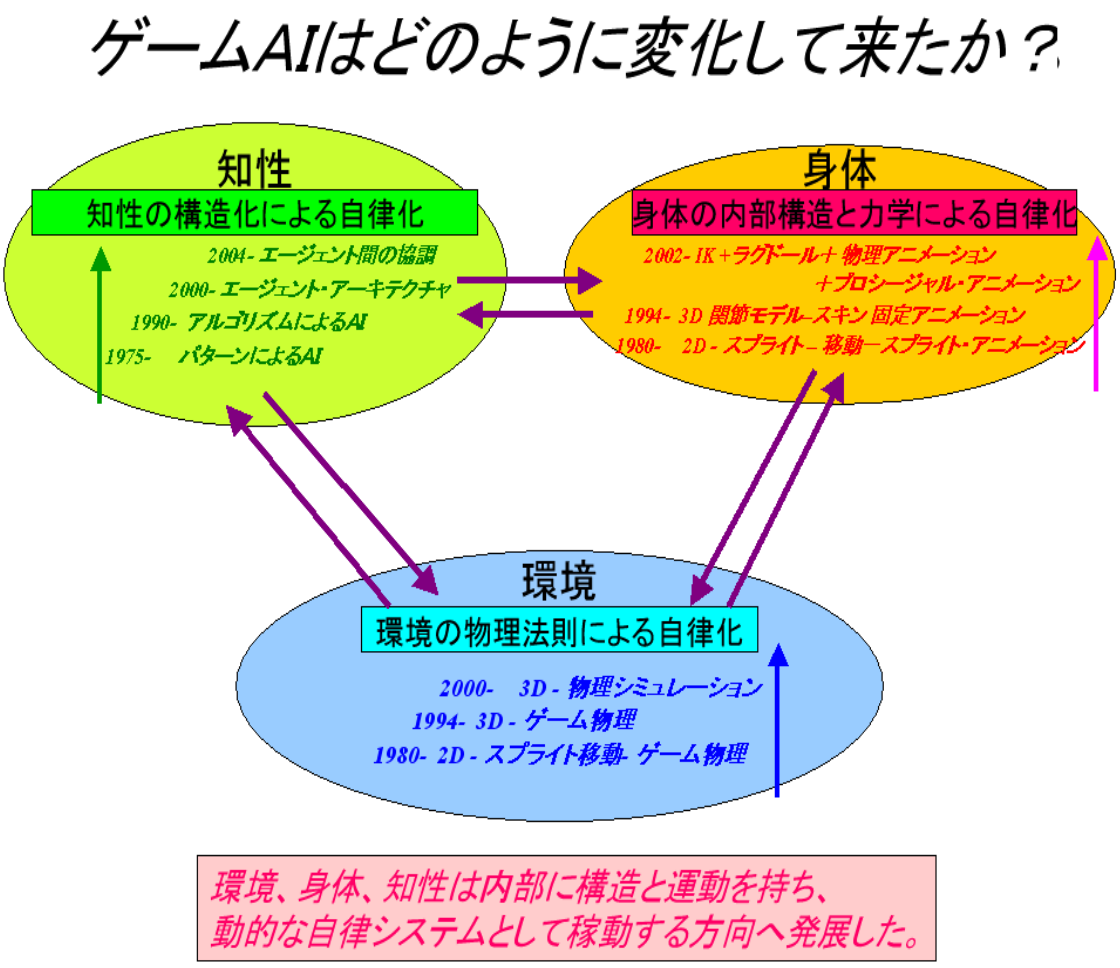
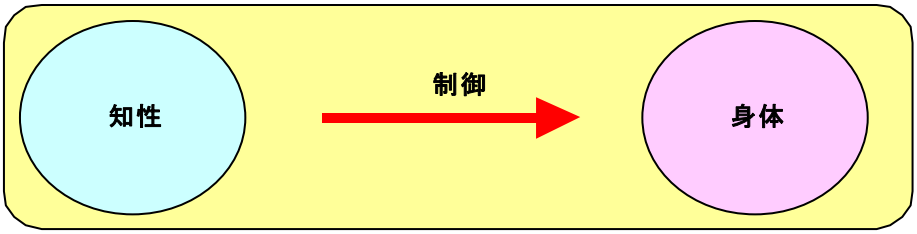
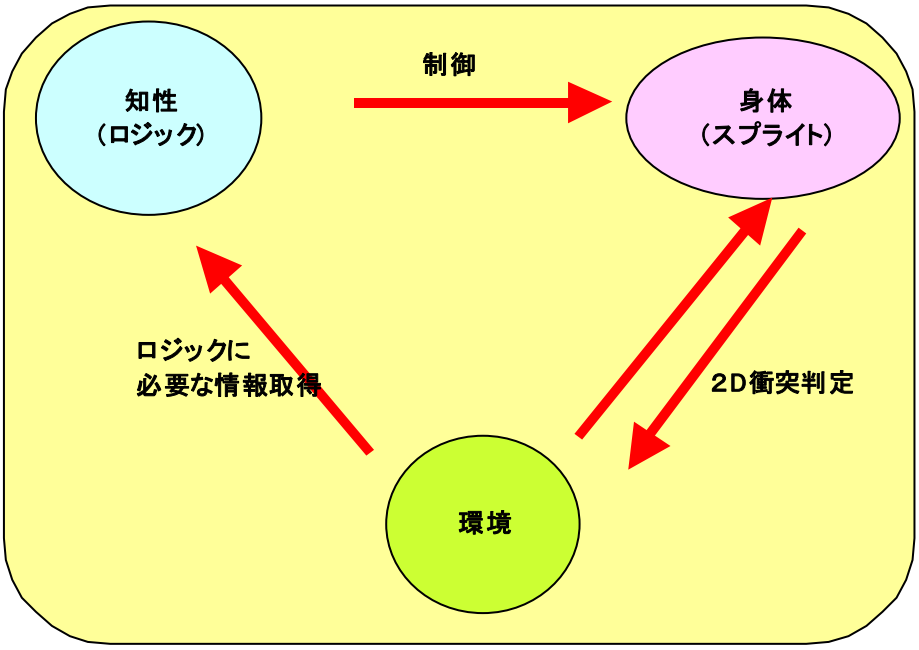
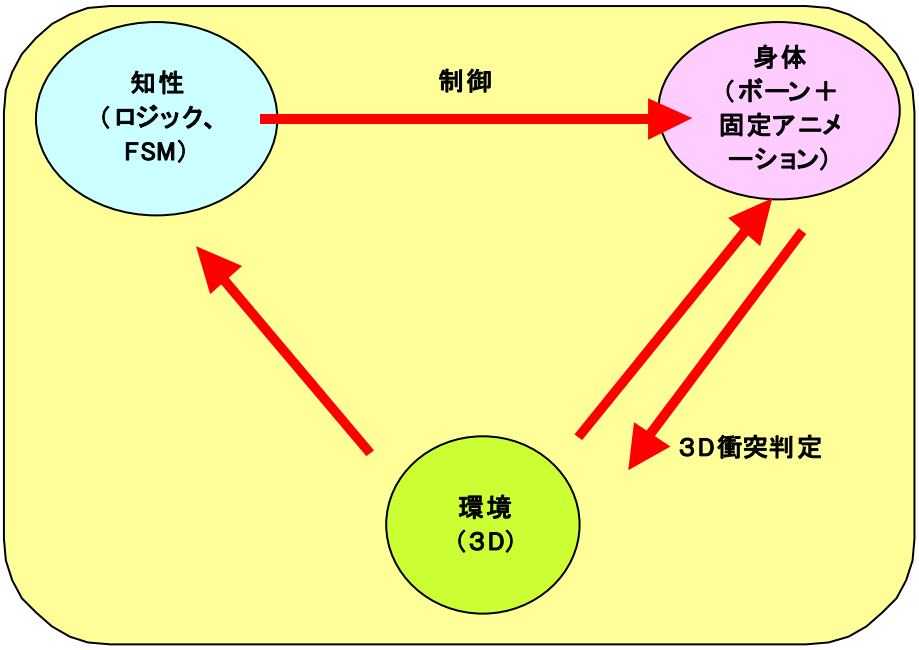


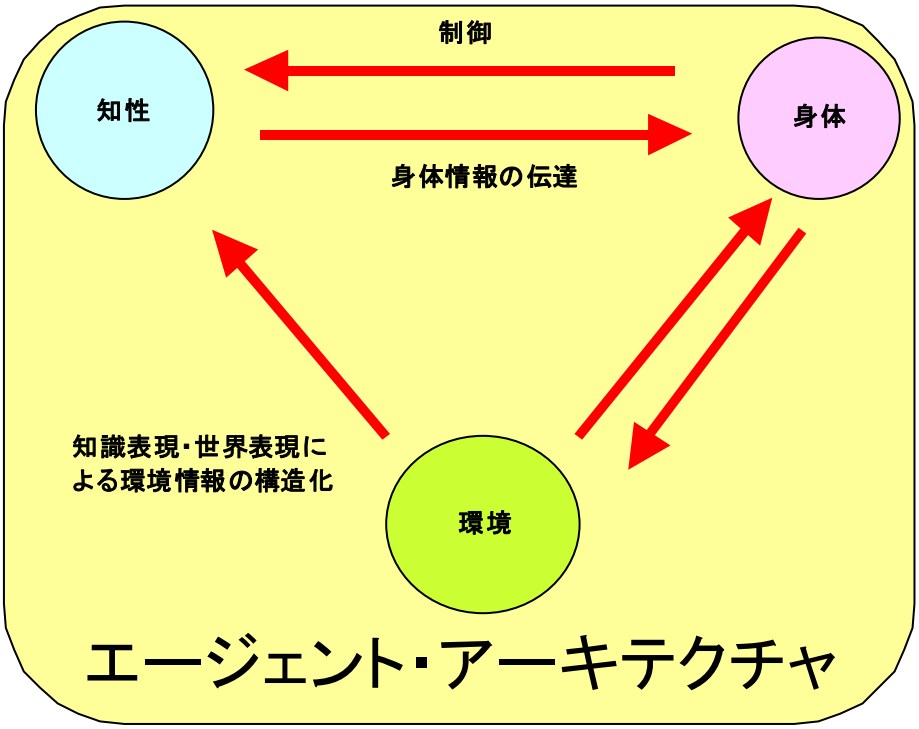
図 3.2-76　「知性」「身体」「環境」のそれぞれの歴史的な進化

環境が変化すれば身体が変化しなければならない。身体が変化すれば、それに応じたその身体をうまく環境内で動かすことが出来る知性でなければならない。そのように 3 者の相互作用によって知性が進化して行く。デジタルゲームでは、まず環境が変化し、身体が対応し、知性が順応するという順番である。例えば、物理シミュレーションが入れば、身体制御にも物理駆動になるようにする。環境とのマッチングを考えればそれが自然だ。すると、その物理駆動になった身体をうまく制御する知性が必要となる、という順番である。

表 3.2-05　キャラクターAI の歴史年表

時期	キャラクターAI	備考
I	 単純なパターンをくり返す AI。決められた動作しかしない AI。 AI はステージの一部であり、ステージ内で動く領域も決められていた。 [年代]　デジタルゲーム黎明期～	2D
II	 知性は環境から情報を取得し、ロジックによって行動パターンを選択する。 身体のスプライト・アニメーションで、環境との衝突判定によって切り替わる。 [年代]　1980 年～	2D

III	 環境が 3D 化。身体はボーン構造化。知性は C 言語など高級言語により、より豊かなロジックが書かれるようになるが複雑化。スクリプトなどより緩和。 環境の複雑化に AI のシステムが追いついていない、かつ、この時期、グラフィック偏重の時代が来て、AI に力を入れる企業はまだ少なかった。 [例]1990 年～ [タイトル]　DOOM	3D

IV	 「知性、身体、環境」三者の関係を総合したアーキテクチャとして AI を構築するフレームが登場する。以後、このアーキテクチャ上に、 ● 知識表現 ● 世界表現 ● プランニング ● HSM(Heuristic Finite State Machine) ● マルチエージェント ● ゴール指向 ● … など、アカデミックな AI 技術が搭載されることになる。 [例] HALO シリーズ F.E.A.R.	3D

(8) References

[1] "Halo", http://www.bungie.net/

[2] CrAIg W. Reynolds： "Boids", http://www.red3d.com/cwr/boids/

[3] ビバリウム： "シーマン", http://www.vivarium.co.jp/

[4] "N.U.D.E.@ Natural Ultimate Digital Experiment", http://www.xbox.com/ja-JP/games/n/nude/detAIls.page/

[5] "OPERATOR'S SIDE", http://www.jp.playstation.com/scej/title/operatorsside/

[6] "「パックマン」岩谷氏、「Rez」水口氏ら4人のクリエーターが語る世界のゲームデザイン論 「International Game Designers Panel」 (GameWatch)", http://game.watch.impress.co.jp/docs/20050312/GDC_int.htm

[7] 三宅陽一郎： "ディジタルゲームにおける人工知能技術の応用", 人工知能学会誌 , 23 , 44 (2008)

[8] 三宅陽一郎： "これからのゲームAIの作り方", http://www.digrajapan.org/modules/mydownloads/visit.php?cid=10&lid=13

[9] Rolf Pfeifer and Josh C. Bongard： "How the Body Shapes the Way We Think", The MIT Press (2006)

[10] Brooks, R. A.： "A Robust Layered Control System for a Mobile Robot", http://people.csAIl.mit.edu/brooks/publications.html

[11] "Jason Brownlee, Finite State Machines (FSM) (AI-depot)", http://AI-depot.com/FiniteStateMachines/FSM-Practical.html

[12] 三宅陽一郎： "Spore におけるゲームAI技術とプロシージャル", http://www.digrajapan.org/modules/mydownloads/visit.php?cid=10&lid=8

[13] Don Hopkins： "The Soul of The Sims, by Will Wright", http://www.donhopkins.com/drupal/node/148

[14] "Ken Forbus, “Simulation and Modeling: Under the hood of The Sims” (NorthWerstern大学、講義資料)", http://www.cs.northwestern.edu/%7Eforbus/c95-gd/lectures/The_Sims_Under_the_Hood_files/frame.htm

[15] "Kenneth D. Forbus “Some notes on programming objects in. The Sims”", http://www.qrg.northwestern.edu/papers/Files/Programming_Objects_in_The_Sims.pdf

[16] "AI SUMMIT SLIDES, NOTES, HEILIGHTS AND PHOTOS(AIGameDev)", http://AIgamedev.com/open/article/GDC09-slides-highlights/

[17] Megan Vista： "Neural Network", http://www.cse.lehigh.edu/~munoz/CSE497/classes/MeganNeural.ppt

[18] "Interview with Jeff Hannan, creator of AI for Colin McRae Rally 2.0 (generation5)", http://www.generation5.org/content/2001/hannan.asp
[19] 森川幸人： "マッチ箱の脳（新紀元社）" (2000)
[20] BrAIn Schab： "AI Game Engine Progrmming (Charles River Media)"
[21] Mat Buckland： "Building Better Genetic Algorithms", AI Game Programming Wisdom , 2 , 649
[22] "『アストロノーカ』（muumuu）", http://www.muumuu.com/games/astro/
[23] 森川幸人： "AI DAY(3)ゲームとAIはホントに相性がいいのか？（GDC2008）", http://www.muumuu.com/other/cedec2008/index.html
[24] 森川幸人： "「テレビゲームへの人工知能技術の利用」, 人工知能学会誌vol.14 No.2 1999-3", http://www.1101.coMMOrikawa/1999-04-10.html
[25] Mat Buckland： "AI Techniques for Game Programming (Premier Press)"
[26] "Neural Networks Research Group, Department of Computer Sciences, University of Texas at Austin, Neuro-Evolving Robotic Operatives", http://www.nerogame.org/
[27] "id Software", http://www.idsoftware.com/
[28] 三宅陽一郎： "デジタルコンテンツ制作の先端技術応用に関する調査研究　第3章「ゲームAI分野」", http://www.dcaj.org/report/2007/data/dc08_07.pdf (2008)
[29] "The MIT Media Laboratory's Synthetic Characters group", http://characters.media.mit.edu/ (2004)
[30] D. Isla, R. Burke, M. Downie, B. Blumberg： "A Layered BrAIn Architecture for Synthetic Creatures", http://characters.media.mit.edu/Papers/ijcAI01.pdf (2001)
[31] R.Burke, D.Isla, M.Downie, Y. Ivanov, B. Blumberg： "The Art and Architecture of a Virtual BrAIn. In Proceedings of the Game Developers Conference", http://characters.media.mit.edu/Papers/GDC01.pdf (2001)
[32] Jeff Orkin： "Three States and a Plan: The AI. Of F.E.A.R(GDC2006) (Document,PPT,Movie)", http://www.jorkin.com/GDC2006_orkin_jeff_fear.zip (2006)
[33] Jeff Orkin： "Applying Goal-Oriented Action Planning to Games", AI Game Programming Wisdom 2, . Charles River Media. , 217-228 (2003)
[34] Jeff Orkin： "Agent Architecture Considerations for Real-Time Planning in Games, AIIDE Proceedings.", http://web.media.mit.edu/~jorkin/AIide05OrkinJ.pdf (2005)
[35] Griesemer,J： "The Illusion of Intelligence: The Integration of AI and Level Design in Halo", http://www.gamasutra.com/features/GDCarchive/2002/jAIme_griesemer.ppt (2002)

[36] Damian Isla： "Managing Complexity in the Halo2 AI,Game Developer's Conference Proceedings.",
https://www.cmpevents.com/Sessions/GD/ManagingComplexity2.ppt (2005)
[37] Isla,D： "Managing Complexity in the Halo2 AI,Game Developer's Conference Proceedings",
https://www.cmpevents.com/Sessions/GD/HandlingComplexityInTheHalo2AI.doc (2005)
[38] Straatman, R., Beij, A., Sterren, W.V.D.： "Killzone's AI：Dynamic Procedural Combat Tactics", http://www.cgf-AI.com/docs/straatman_remco_killzone_AI.pdf (2005)
[39] Arjen Beij, William van der Sterren： "Killzone's AI：Dynamic Procedural Combat Tactics (GDC2005)", http://www.cgf-AI.com/docs/killzone_AI_GDC2005_slides.pdf (2005)
[40] "西川善司の3Dゲームファンのための「KILLZONE 2」グラフィックス講座(後編) ～オランダ精鋭部隊がCELLプロセッサで実装した「脅威推測型AI」の秘密 (GameWatch)",
http://game.watch.impress.co.jp/docs/series/3dcg/20090424_153727.html
[41] "Bungie Publications （Haloの公開された技術情報は全てここに掲載されている）",
http://www.bungie.net/Inside/publications.aspx
[42] Marco Pinter： "Toward More Realistic Pathfinding (Gamasutra)",
http://www.gamasutra.com/features/20010314/pinter_pfv.htm
[43] "良質なゲームを生み出すプレイテストのアプローチ Valveの実験心理学博士が明かす、プレイテスト手法のあれこれ(GameWatch)",
http://game.watch.impress.co.jp/docs/news/20090329_80132.html
[44] "Valve語る、「Counter-Strike」から「Left 4 Dead」へ 協力プレイ、リプレイ性、AIディレクターの秘密(GameWatch)",
http://game.watch.impress.co.jp/docs/news/20090327_80051.html
[45] "Valve Software、「Physical Gameplay in Half-Life2」 (GameWatch) 物理エンジンが生むリアリティーとゲームとしての面白さのバランス",
http://game.watch.impress.co.jp/docs/20060325/valve.htm
[46] "Left 4 Dead Interview (Eurogamer)", http://www.eurogamer.net/articles/left-4-dead-interview
[47] "なんのためにボスキャラがいるのか？ (4gamers) 「Left 4 Dead」に見る，プレイヤーをCo-opに誘う仕掛け",
http://www.4gamer.net/games/035/G003579/20090327043/
[48] "良質なゲームを生み出すプレイテストのアプローチ Valveの実験心理学博士が明

かす、プレイテスト手法のあれこれ（GameWatch)",
http://game.watch.impress.co.jp/docs/news/20090329_80132.html
[49] "BUNGIE GLOBAL HEATMAPS",
http://www.bungie.net/Online/HeatMaps.aspx
[50] "西川善司の3Dゲームファンのための「Gears of War 2」グラフィックス講座　究極の流血表現と残虐表現に見る新生「Unreal Engine 3」の実力とは？（GameWatch)",
http://game.watch.impress.co.jp/docs/series/3dcg/20090515_168708.html
[51] Dave C. Pottinger：　"TerrAIn Analysis in Realtime Strategy Games",
http://www.directxnedir.com/pottinger.doc
[52] Dave Pottinger：　"TerrAIn Analysis in Realtime Strategy Games",
http://vr.kAIst.ac.kr/courses/cs682/data/ssw.ppt　(2002)
[53] Lionhead Studio：　"Fable", http://www.xbox.com/ja-JP/games/f/fable/
[54] Michael Mateas, Andrew Stern：　"Facade", http://www.interactivestory.net/

3.2.6　プログラミング　グラフィックス描画

宮澤 篤
大久保 明
株式会社バンダイナムコゲームス

ピンポンをもとにした、商業的に成功した初めてのアーケードゲーム『ポン（PONG)』が、米国アタリ社で発明されてから、既に 36 年以上の歳月が流れている。当時のゲームは、汎用ロジック IC を組み合わせて設計されており、技術的に見ても未発達で、最も単純な対話型コンピュータグラフィックスの一応用分野でしかなかった。それから現在までに、世界中のさまざまな会社から、その時代の最も進んだコンピューター技術を取り入れた、非常にたくさんのゲームが発表されてきた。今日のコンピューター・ゲームは、幾多の技術革新を経て進化してきた、全く新しいインタラクティブなメディアである、と言えるかもしれない。本稿では、プログラミングのグラフィックス描画を中心に、コンピューター・ゲームを構成するいくつかの基本的な技術について解説する。

(1)　要素技術

ゲームマシンの基本的な機能は、汎用のコンピュータシステムと何ら変るところはない。現在のゲームマシンが登場するまでの歴史は、コンピューターが発展していった過程をそのまま忠実にたどりながら、しかしゲームの面白さを追求していく中で、制御、映像、音声、通信などにおける新しい技術が導入され、進化してきた。

米インテル社（Intel Corp.）が最初の一般向けマイクロプロセッサ 8008 を発表した 1972 年に、米マグナボックス社（Magnavox）から最初のコンソール『オデッセイ（Odyssey)』が、そしてアーケード（業務用）の世界では、米アタリ社（Atari Corp.）を設立した Nolan Bushnell によって『ポン（PONG)』が相次いで開発されたことから、商業的な見方をすれば、この年をゲーム産業における「物語の始まり」と考えることができる。同年には、これらのピンポンゲームをめぐって、ビデオゲームの歴史上、最初の訴訟が Magnavox から起こされた。争い自体は不幸なことといわねばならないが、これによって埋もれていたゲーム黎明期の姿が、白日の下に晒されたのは幸いだった。

1950 年代には、真空管方式のアナログコンピュータを使って（主に弾道計算による）微分方程式の解曲線を、対話的にオシロスコープ上に表示するシステムが存在してはいたものの、果たしてそれらをコンピューター・ゲームと呼ぶべきかどうかについては、大いに意見の別れるところであろう。ここでは 1958 年、米ブルックヘブン国立研究所（Brookhaven National Laboratory）の物理学者 William Higinbotham による、オシロスコープ上のテニスゲーム『テニス・フォー・トゥー（Tennis for Two)』（図 3.2-77）を例として挙げるにとどめるが、このシステムにはスコアが存在しないことから、プレイヤーによる操作の結果が利益か不利益か、どちらかが判断できるというゲームの最低条件を満たしていないといえる。標準の TV モニターを使った商業製品としながらも、ゲームとしての面白さを最大限に追求した功績からは、Odyssey の開発者である Ralph Baer と Nolan Bushnell の二人が、間違いなくビデオゲームの発明者と呼ばれるに相応しい。[1][2][3][4]

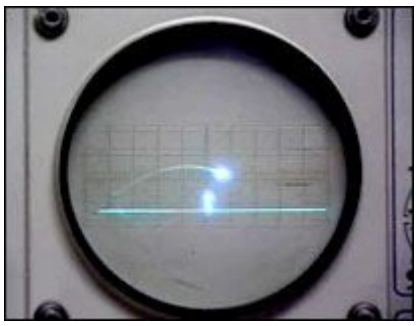


図 3.2-77　『テニス・フォー・トゥー』のゲーム画面

図 3.2-78　『サーカス』のゲーム画面（© Exidy, Inc.）

もっとも、前述の PONG や、たとえば「ブロックくずし」という呼び名のもと大ブームとなった『ブレイクアウト（Breakout)』（1976 年、Atari Corp.）などは、TTL-IC だけを組み合わせて設計された大掛かりなもので、当時の電卓技術を流用したといわれてい

る。

コンピューター・ゲームに汎用プロセッサが使われ始めたのが、二人のカウボーイがサボテンを避けながら撃ち合う『ウエスタンガン（Western Gun)』(1975 年、タイトー、Intel 8080 を使用）や、8 人のプレイヤーがそれぞれのタンクを操縦して、戦場に見立てたフィールド内で戦闘を行う『タンク 8（Tank 8)』(1976 年、アタリゲームズ、モトローラ 6800 を使用)、ジャンプするキャラクターをシーソーで受け止めながら、画面上部を左右に流れる風船に当てる、「風船割り」の通称でも有名な『サーカス（Circus)』（1977 年、Exidy、モステクノロジー社の 6502 を使用、図 3.2-78)、などにおいてであり、以降のゲームは例外なくマイクロプロセッサを用いている。

コンピューター・ゲームにとって必要なのは、高速で動くたくさんの色鮮やかなキャラクターである。グラフィックスの表示速度が遅ければ、いくら緻密で美しい画面を出したとしても、もはやゲームとしては成り立たない。たとえばプレイヤーがコントローラーのボタンを押した瞬間から、どんなに遅くとも 1/30～1/15 秒（テレビの一画面を 2～4 回書き換えるのに必要な時間）後には確実に画面上で反応が返ってこなければ、コンピューター・ゲームの面白さは絶対に伝わらないのである。現在のゲームマシンやパーソナルコンピュータで主流を占めているビットマップ──正確にいえば「ピクセル」マップ方式のグラフィックスは、速度上の問題から、それまでのゲームマシンにはほとんど使われていなかった。

過去におけるいくつかの例外としては、コンピューター・ゲームの存在を世に知らしめたともいえる、有名な『スペースインベーダー（Space Invaders)』(1978 年、タイトー）や、玉簾のようになって画面上を飛び回る敵を避けながら陣地を占領する『QIX』（1981 年、タイトー)、魔法の道具「リブル」と「ラブル」を使って、敵に魔法をかけられたキノコ「マシュリン」を取り囲み、その中を塗り潰しながら消していくグラフィックを、スプライト面の背景として用いた面白い例に『リブルラブル（Libble Rabble)』（1983 年、ナムコ）などがある。

(a) 背景とスプライトの基本技術

背景と動画の多重合成から成り立っていた当時のゲームマシンの画面は、ちょうどテレビのセル画によるアニメーションのような 2 次元の世界そのものであった。ゲームマシンの画面構成をひとことでいえば、PCG（Programmable Character Generator、プログラム可能な文字発生器）を使った疑似グラフィックのキャラクター面による重ね合わせと、それらを独立して上下左右にドット単位で動かせるスクロール機能である。これに、後述するスプライトの機能が加わって、あのスピード感あふれる映像が作り出される。

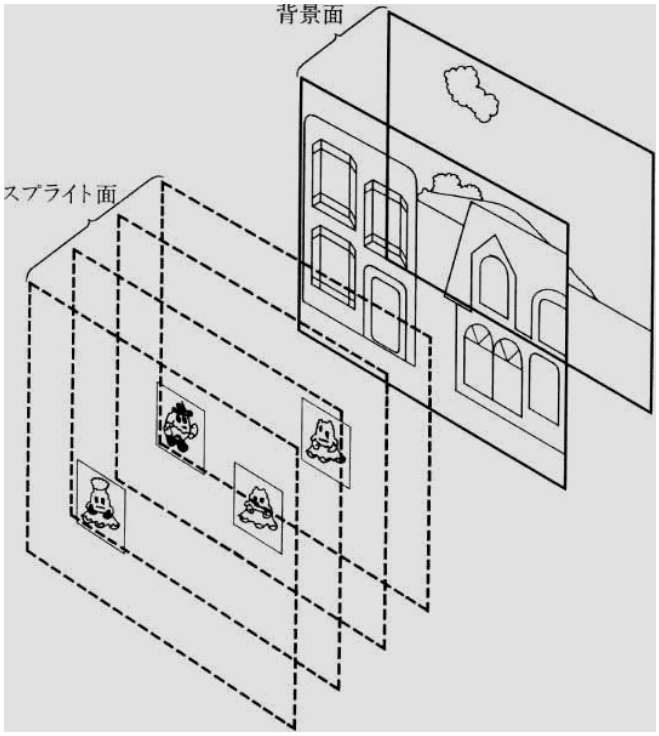


図 3.2-79　ゲーム画面の構成

スプライトとは、ハードウェアによる特殊な表示機能をそなえたキャラクターパターンのことである。これらのパターンについては、ラインバッファまたはフレームバッファ上で、コンピュータグラフィックスでいうところの深度ソート法による隠面消去アルゴリズムが、TV モニターの帰線期間内に行われる。したがってパターンを、座標を変更するだけで表示画面内を自由に、しかも背景に影響を及ぼさずに移動することができ、ソフトウェアによる再配置などは一切必要としない。また、スプライトには優先順位があり、2 つ以上のスプライトが重なった場合、優先順位の低いスプライトが自動的に消えるようになっていて、手前にある物体が後方の物体の前を通り過ぎるように見え、画面に奥行き感を持たせることができる。さらに、スプライトどうしの一画素が一致すると、プロセッサ内部の衝突フラグがセットされるなど、コンピューター・ゲームの制作には欠かせない機能の一つであった。1983 年 7 月にアメリカのマイクロソフト社が提唱した、ホームコンピュータのハード、ソフトウェアの統一規格である「MSX パソコン」などにも採用された、TI（テキサス・インスツルメンツ）社のビデオディスプレイプロセッサ（TMS9918A）は、このスプライトというユニークな機能を持っていることが特徴だった。しかし業務用ゲームマシンでは『タンク（Tank)』(1974 年、キーゲームズ）で[5]、日本では『ギャラクシアン（Galaxian)』(1979 年、ナムコ）で、初めてスプライト表示回路を搭載した基板が使われたのを皮切りに、それ以降のゲームマシンには、性能面でそれらをはるかに上回る、カスタムのディスプレイプロセッサが搭載されていた。

スプライトは、基本的に 8×8 ドットまたは 16×16 ドットなどの大きさで構成される動

画のデータ（スプライトパターン）と、スプライトの垂直位置、水平位置、定義されたスプライトの番号、カラーコード（透明も含む）等の属性をメモリに設定することにより表示される。そして、スプライトのハードウェアがどのようにして動画を表示するのかについては、TV モニターのブラウン管上に画像が生成される仕組みとも大きく関係している。まず、一つの走査線の画素数に相当するラインバッファ（メモリ）を用いて、以下のようなステップで処理が行われる。

(ア) 各々のスプライトに対して、各々の走査線を表示するその前の水平帰線期間中に、そのスプライトが現在の走査線上に存在するかどうかを、スプライトの垂直位置と走査線の位置を比較することにより調べる。
(イ) もし存在するなら、そのスプライトの番号と垂直位置に対応する一ライン分のスプライトパターンを読み出し、スプライトの水平位置に対応するラインバッファ上の該当する位置に、そのパターンを書き込む。
(ウ) すべてのスプライトについて、優先順位の低いスプライトから高いスプライトの順に、ステップ（ア）と（イ）を繰り返す。
(エ) 水平走査によってラインバッファの内容をディスプレイ上に表示し（パターンの書き込まれていない場所には、後から背景画が合成される）、それと同時にバッファを初期化する。
(オ) すべての走査線について、ステップ（ウ）と（エ）を繰り返す。

水平帰線期間だけでは処理時間が短いため、ラインバッファを二つ設けて（いわゆるダブルバッファリング）、水平走査の時間も合わせた水平期間（約 63.55mSec）でスプライトの表示と処理を並列に行うものもある。一方のラインバッファにパターンが書き込まれているときは、他方の内容がディスプレイ上に表示されていて、走査線の終りごとに、二つのバッファは役割を交替する。

スプライトの優先順位は、このラインバッファへスプライトのパターンを書き込む順番で決まる。優先度の高いスプライトは低いスプライトの後から処理されるので、スプライト同士が互いに重なり合う場所ではどこでも、優先度の高いスプライトが低いスプライトを遮っているように見える。

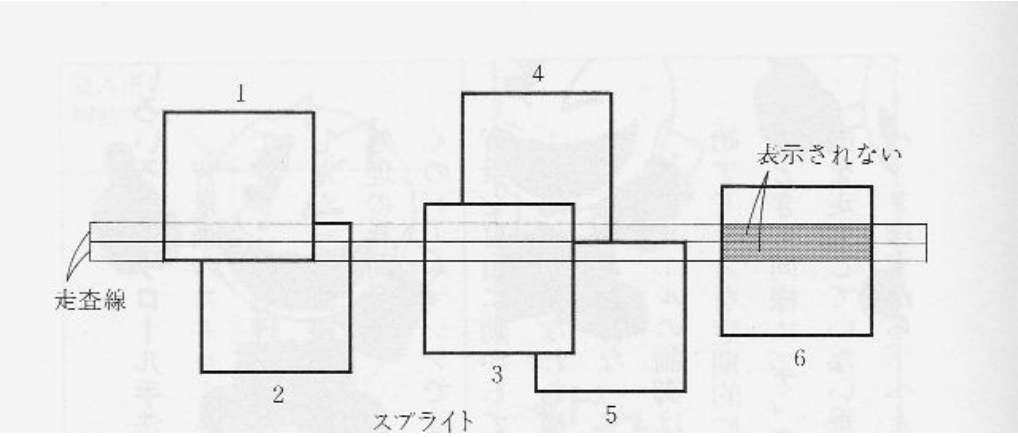


図 3.2-80　ラインバッファによるスプライトの表示制限（制限が 5 個の場合）

ラインバッファによるスプライトの表示処理は一走査線単位に行われているため、この間にいくつのスプライトを処理できるかが表示個数の制限となる。ゲームを制作する場合、この制限を越えることがあれば、スキップされて余ったスプライトは画面上に表示されず（Sprite Blanking と呼ばれる）、ゲームとして成り立たない。1980 年代後半に入ってから、高速で動作する大容量のメモリが安価で供給され始めると、ラインバッファの代りに（ダブルの）フレームバッファをもつものが出現した。この方式は、一画面を走査する時間内に、画面上のすべてのスプライトについて同時に表示と処理を行うことによって、前述の同一走査線上の表示制限をなくし、画面ごとに処理されるスプライトの個数にだけ制限をもつという利点を有していた。

(b)　いろいろなスクロール手法

背景画のスクロールは、ゲームの視覚的なリアリティーを高めるために欠かせない処理である。この処理を行う方法として、フレームバッファ上の背景画を一ラインずつずらしながら、直接書き換えることが考えられるが、ゲーム処理全体に対するスクロール処理の負荷が多く、リアルタイムに背景を動かすのはむずかしくなる。そこで、多くのゲームマシンでは、フレームバッファの表示開始位置を指定するだけで、画面上の背景を自由に動かして表示できる専用のハードウェアを設けて、リアルタイムのスクロール処理、すなわち横方向のスクロール、縦方向のスクロール、そして斜め方向のスクロールなどを行っていた（図 3.2-81）。

スクロールの制御は、CPU 側からハードウェアに対し、フレームバッファの表示開始アドレスを定期的に更新することで実現している。この更新処理は、（スプライトのときと同様に）ディスプレイ上に表示される画像の乱れを防ぐため、走査線が表示画面を走査していない垂直帰線期間に行う。画面上の座標系が、x の値は左から右へ、y は上から下へと増加するように与えられると、横方向へのスクロールは y 座標を固定して x 座標を、

縦方向のスクロールは x 座標を固定して y 座標を、それぞれ垂直帰線期間ごとに更新することで行うことができる。通常、フレームバッファには表示画面より大きなアドレス領域をもたせ、表示画面の外の画像内容を更新することで、同じ背景画の繰り返しではなく、変化に富んだスクロール画面を表現している。

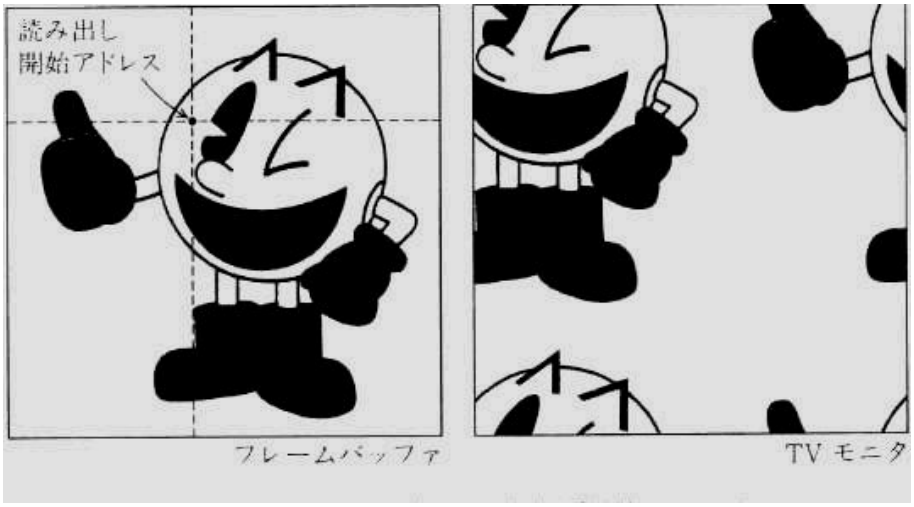


図 3.2-81　スクロール処理の基本

縦、横、二つの方向のスクロールを同時に行うことで、斜め方向のスクロール処理を行っているゲームも存在した。敵のアステロイドシティを低空飛行で攻撃しながら、巨大ロボットと戦う『ザクソン（Zaxxon)』(1982 年、セガ・エンタープライゼス）は、通称スリークオーターズ（4 分の 3）ビューとも呼ばれる、斜め上から見た 3 次元的な映像で、アメリカでも大ヒットとなったシューティングゲームである。このゲームでは、背景が x 方向 2 ドットに対し y 方向 1 ドットの傾きとなっており、この傾きと同じドット単位でアドレス更新を行っているものであった。

複数の背景画をもって、それらが異なる速度で、すなわちプレイヤーから最も遠い面が、前景にあるものよりゆっくり動くことによって、奥行きの感覚をシミュレートするゲームもある。この手法は多重スクロール（英語では Parallax Scrolling）と呼ばれていて、代表的なものに、ミサイルとロケット砲で敵を倒し、障害物を避けながら月面車で走破する『ムーンパトロール（Moon Patrol)』(1982 年、アイレム）や、歌舞伎の演目「出世景清」をベースに制作された純和風アクションゲーム『源平討魔伝（The Genji and the Heike Clans)』(1986 年、ナムコ）などがある。また、鏡の国に平和を取り戻すため、シャボン玉を武器に主人公アリスが活躍するファンタジーアクション『メルヘンメイズ（Marchen Maze)』(1988 年、ナムコ）では、背景の斜めスクロールと組み合せて 3 次元的な表現を行っていた。

(c) ラインスクロール

スクロール処理の中で変ったものに、ラインスクロール（またはラスタースクロール）と呼ばれるものがある。前に述べているように、スクロールするための表示開始アドレスの更新を、垂直帰線期間でなく、水平帰線期間に走査線単位に行ったもので、一つの背景画において走査線単位に独立にスクロールさせることができる。

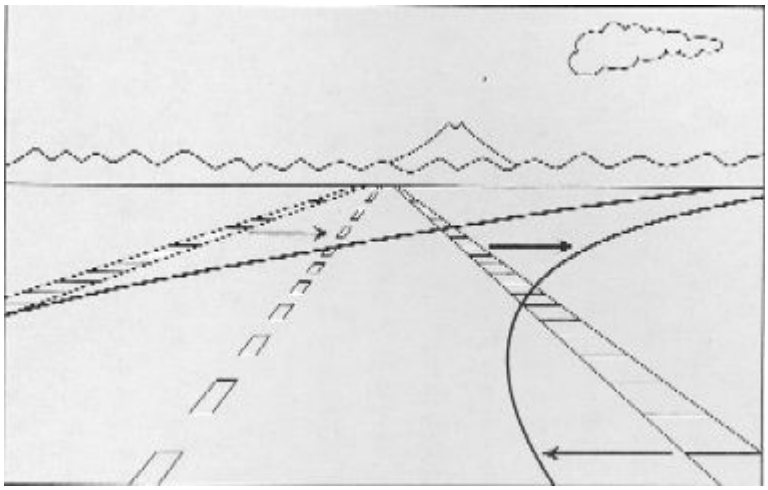


図 3.2-82　ラインスクロールのしくみ

たとえば、表示開始アドレスを一走査線ごとにサインカーブ状に変化させれば、背景画面がゆらゆらと揺れる表現ができる。この代表的なものとして、ハーフミラーを採用した二画面の専用筐体で話題を呼んだシューティングゲーム『ダライアス II（Darius II)』（1989 年、タイトー）の次元気流や、自機や敵機ともに、西部開拓時代の拳銃をモチーフにした縦スクロールシューティング『ガンフロンティア（Gun Frontier)』（1990 年、タイトー）における空間ワープの演出などがあります。面白い例としては『ポールポジション（Pole Position)』（1982 年、ナムコ）に代表される初期の、自車の後方からの視点を採用したドライブゲームがある。これは、ラインスクロールを道路にうまく応用し 3 次元を表現したものである。

(2) 年代別ゲームアーキテクチャ（1970 年代～、スプライト編）

背景やキャラクターの拡大・縮小や回転機能など、コンピューター・ゲームの表現力を高めるための特殊効果は、数学で言うところの座標幾何学の応用であるが、定義通りまともに座標計算をしていたのでは、とても限られた時間内に絵を作り終えることができない。そこで、高速化のためのさまざまな工夫が必要になってくる。

図形の変形の中で、座標計算が最も簡単でしかも応用が広いのが射影変換である。平面

上の射影変換は、任意の位置にある 4 組の対応点によって決定され（図 3.2-83）、数学的には原画像の画素の座標$(x,\ y)$と変換後の画素の座標$(x',\ y')$との対応関係は、同次座標を用いた次式で定義される。

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \quad x' = \frac{x_1}{x_3}, y' = \frac{x_2}{x_3}$$

これまで述べてきたスプライトや画面のスクロール機能（ラインスクロールは除く）といった表示技術は、このうちの平行移動

$$x' = x + c_1, \quad y' = y + c_2$$

に関するものとみなすことができる。コンピューター・ゲームの映像は、アフィン（または線型）変換のような低次な変換から、より高次な射影変換へと、幾何学的な変換の階層をそのまま忠実にたどりながら進化してきたのであった。

まず拡大・縮小については、キャラクターパターン内の一画素を、規則的に繰り返したり間引いたりしながら表示することでとりあえず実現できて、いわゆる最近傍補間（Nearest-neighbor Interpolation）による 0 次の補間と同じ結果になる。ひとたび、画像の拡大・縮小が実時間で実行できるようになると、近くの物体を大きく、遠くなるに従って小さく表示し、これによってゲーム画面に疑似的な 3 次元効果を産み出すことが可能になる。

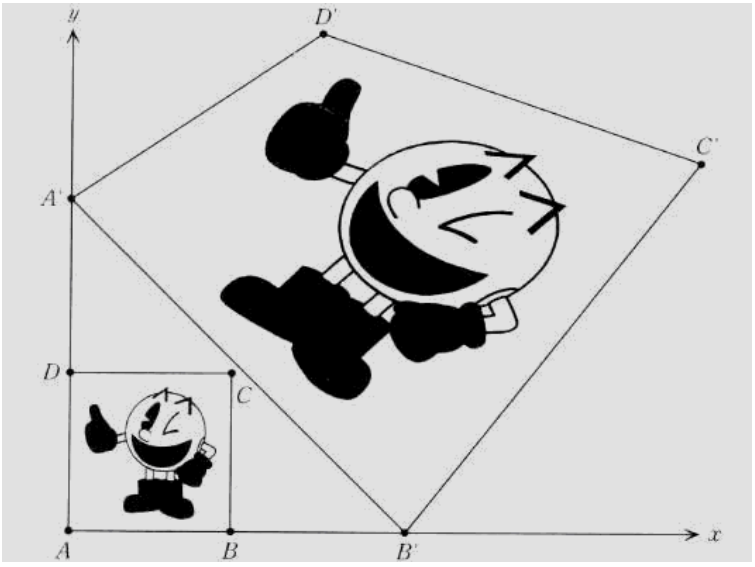


図 3.2-83　4 組の対応点によって決まる射影変換

その結果、例えばセガ社の大型可動筐体による「体感ゲームシリーズ」として、『ハングオン（Hang-On)』や『スペースハリアー（Space Harrier)』(1985 年)、『アウトラン（Out Run)』(1986 年）など、名作と呼ばれた疑似 3 次元ゲームが次々と輩出することになる。

回転については、2 次の変換行列をたかだか 4 つの単純な（ラスタ走査方向に処理が可能な）行列の積に分解し、それぞれを整数演算の範囲内で、高速に計算するようなアルゴリズムがいくつか知られている。しかし、現実のゲームマシンでは、各パターンをリアルタイムに回すのではなく、ディスプレイコントローラがフレームバッファ上の回転された矩形領域を走査することによって、あたかも回転しているように見せているものがほとんどである（図 3.2-84)。初めてハードウェアによるフレームバッファの回転機能を実現したのが、『カウンターステア（Counter Steer)』(1984 年、データイースト）においてであり、その後も『A-JAX』(1987 年、コナミ)、戦車やヘリコプターを操るシューティングゲーム、『アサルト（Assault)』や『メタルホーク（Metal Hawk)』(1988 年、いずれもナムコ）などの作品が生まれている。

また、3 次元空間内の射影によって、長方形の背景パターンを台形に変形し、奥行き感を持った画面を作る技法──いわゆる（奥行き方向への）“倒し込み”が、『G-LOC』(1990 年、セガ)、『ゴルフィンググレイツ（Golfing Greats)』(1991 年、コナミ）などで使われている。

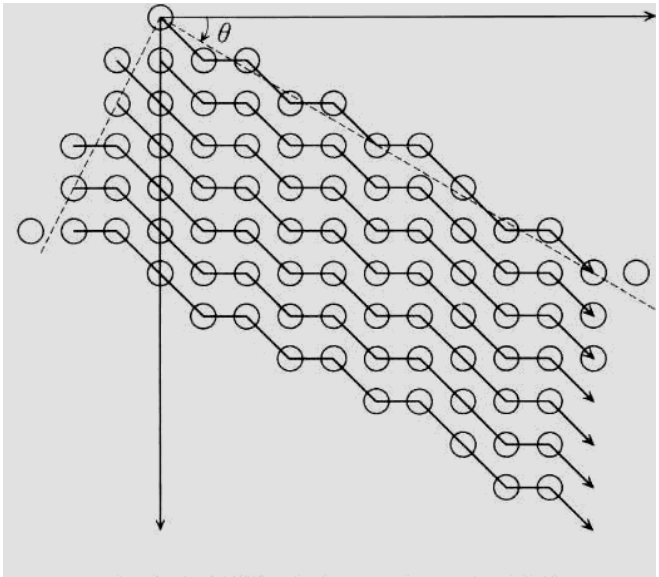


図 3.2-84　回転機能によるフレームバッファの走査

この変換にも、やはり（双曲線発生器を用いて）加減算とシフト操作の範囲内でインクリメンタルに行えるアルゴリズムが存在する[6]。そしてこれらの手法は、やがて全盛を迎える 3DCG ゲームのテクスチャマッピング機能として結実することになるのである。

(3) 年代別ゲームアーキテクチャ（1990 年～、ポリゴン編）

3DCG の世界では、レンダリング（rendering）法として「レイトレーシング（ray tracing)」「ラジオシティ（radiosity)」等が有名あるが、ビデオゲームの世界でレンダリングの基本は「ポリゴン（polygon、多角形）レンダリング」のことである。ポリゴンレンダリングとは、文字通り 3 角形や 4 角形等の多角形ンダリングは他のレンダリング法と比較して計算量が少ないため、高いリアルタイム性を必要とするビデオゲームではこのレンダリング法をハードウェア化してさらに高速化を図ることで高いインタラクティビティ（即応性）を実現している。

ポリゴンハードウェアの構成は、大きく分けて実数計算で 3D 演算を行うジオメトリ処理（ジオメトリエンジン）と、整数計算やビット操作で描画を行うラスタ処理（ラスタエンジン又はレンダリングエンジン）の 2 つのブロックに分かれている。ジオメトリエンジンは、物体の x y z 座標の 3D 計算および光源計算、スクリーンからはみだした部分のクリッピング、スクリーンに投影するための射影変換処理等を行う。ラスタエンジンは、陰面消去およびポリゴンのフレームバッファへの描画制御を行う。

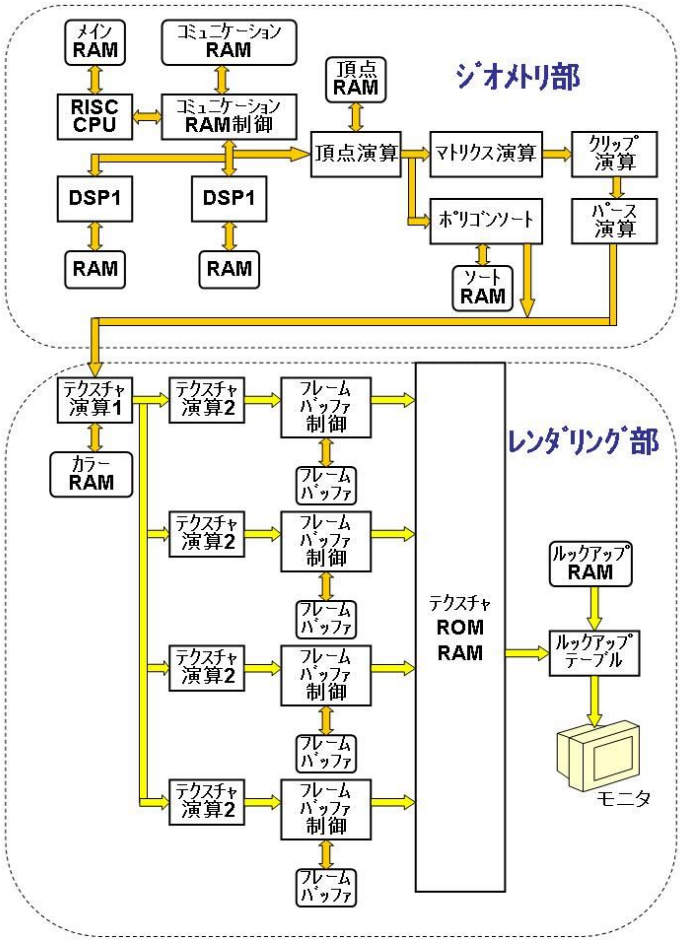


図 3.2-85　業務用ゲームマシン構成例

(a)　リアルタイム性の追求

アクションゲームやシューティングゲーム、格闘ゲーム等を成立させている最大の要件は、リアルタイム性である。通常ビデオゲームでは、フィールドレートは 60f/s（または 30f/s）であり、ハードウェア設計はこの更新レートで最高の性能（ポリゴン数や画像表現、画質）が出せるようなアーキテクチャであることが要求される。

近年では、半導体のプロセスの進化（45 ナノミクロン）等により高精細でフォトリアリスティックな画像をリアルタイムで生成できるゲームマシンも登場している。しかし 1990 年初頭のポリゴンハードの黎明期には、ゲームマシンとして使用可能な半導体の製造プロセスは、1 ミクロン程度であり 1 フレームに数千枚程度のポリゴンをリアルタイムで生成するために各社では様々なアーキテクチャ上の工夫が施されていた。System2x アーキテクチャ（1991 年～、ナムコ）[7]を例にとりゲームマシンがいかにしてリアルタイム性を獲得したかについて解説する。

(b)　4 角形ポリゴン

通常の 3D ポリゴンレンダリングでは、ポリゴンは 3 角形が使用されている。これは、シェーディングやテクスチャマッピングの処理を簡易に行うためだが、System2x では、4 角形をレンダリングのプリミティブとして採用している。これは、1990 年当時の 3D ゲームの登場キャラクターが車や戦車であり 4 角形プリミティブで構成されているものが多く、回路的に複雑になっても 3 角形 2 枚で 4 角形を構成するより、少ないポリゴン数でオブジェクトを構成できるという利点があった。

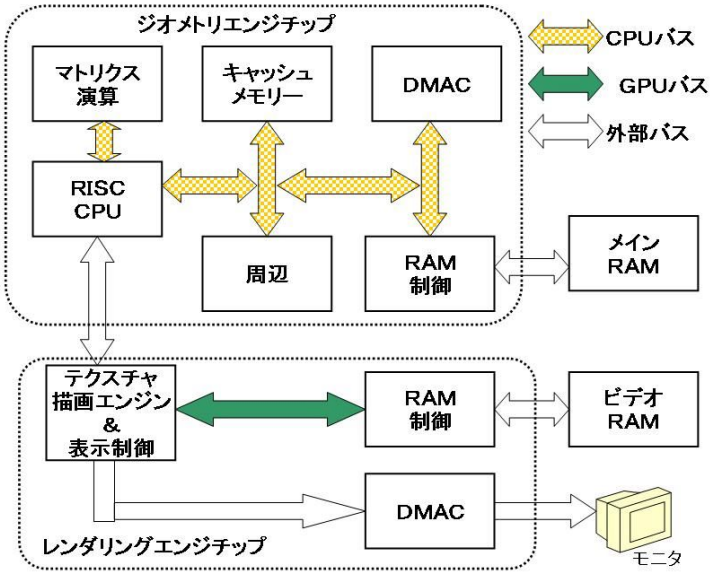


図 3.2-86　家庭用ゲームマシンの構成例

(c) 陰面消去

System2x アークテクチャーでは、陰面消去に、優先順位法（*z* ソート法）と先書き優先法を採用している。System2x の優先順位法は、ポリゴン単位で *pz* 値（ポリゴンの *z* 座標、24 ビット）を与え他のポリゴンとの前後関係の比較はその *pz* 値で行われる。ピクセル単位で比較を行う *z* バッファ法に比べてポリゴン単位で比較を行うこの方式は、RAM 容量や計算処理を軽くすることが可能である。しかし、ポリゴンどうしが *z* 方向でクロスする場合正しく表示できないという不具合が発生する。不具合（優先順位の狂い）をできるかぎり軽減するために System2x では次のような補正を行っている。ひとつは *pz* 値 24 ビットの上位 3 ビットは、絶対優先順位と呼ばれ、ポリゴンの *z* 座標そのものから計算するのではなくオブジェクト（ポリゴンの集合）どうしの前後関係や特別に固定されたポリゴン（地面や背景等々）であることを示す。その他に *pz* シフト（*pz* 値に 18 ビットの値を加算する）*pz* 固定値（*pz* 値の下位 21 ビットを強制的に固定値に入れ替える）等々の補正処理をしている。補正の際 *cz* 値（カラーの *z* 値、13 ビット）を *pz* 値とは別に設けることで、上記のシフト処理等で色が変わってしまうことを防止している。

先書き優先法とは、フィールドバッファに対する描画を手前（＝優先順位が高い）のポリゴンから行う処理である。この処理により、陰面（他のポリゴンの陰に隠れて見えない部分）の描画時間の短縮が可能である。また処理時間の不足で描画を中断する場合（ゲームマシンの場合通常フィールド内での処理時間が不足した場合、処理は強制的に中断され、次のフィールドの処理に移る）奥（＝優先順位が低い）のポリゴンから欠落するので欠陥が比較的露呈しにくいという利点がある。

図 3.2-87　『リッジレーサー』の筐体とそのゲーム画面（© NBGI）

しかし、フィールドバッファを読み出し、それ以前に描画されていなければ書きこむという方法では、描画処理は短縮されるが、読み出し時間の分の処理が増大する。System2x では、1 ドットに 1 ビットが対応したフィールドバッファを別に用意し、描画処理したピクセルは「1」未描画ピクセルを「0」とし、1 アクセスで 16 ピクセル検出し描画処理したピクセルは処理をスキップすることで処理能力を上げている。

(d) テクスチャマッピング

テクスチャマッピングの手法としては、最もシンプルなポイントサンプル法から、バイリニア法、トライリニア法、MIP マップ法等各種の方法が考案されておりそれぞれを組み合わせた方法もあるが、それぞれ利点欠点を持っている。これまでのゲームマシンで使用可能な回路規模においては、マッピング手法も簡単な擬似手法による演算しか行えないためにマッピングされたテクスチャの遠近感の欠如や歪みなどが生じ著しくリアリティーや画質の低下を招く要因になっていた。これは擬似手法での演算では、ポリゴンの x y 座標のみ射影変換され z 座標及びテクスチャの座標(Tx, Ty)が射影変換されていないため、この 5 つの座標が非線型になることに起因している。System2x 系のテクスチャマッピングは、この z 座標及びテクスチャの座標(Tx, Ty)も射影変換して 5 つの座標を線型な関係にすることで、遠近感がありなおかつ歪みのないテクスチャマッピングを実現している。

具体的には、ポリゴンの頂点(x, y, z)を射影変換する式は視点とスクリーンの距離を h とすると

$$x^* = x \times \frac{h}{z}$$
$$y^* = y \times \frac{h}{z}$$

で表される（*は射影変換後の座標値)。x と y は線型であり x^* と y^* もまた線型である。

射影変換という概念をより一般化して、新たな座標 w を想定し w と x、w と y が線型であるとする。w^* と x^*、w^* と y^* もまた線型となるような w から w^* への変換を w の射影変換と考えるとすると

$$w^* = w\frac{p}{z} + \frac{q}{z} + r$$

(p, q, r は任意の定数)という一般式が導かれる。

ここに z と(Tx, Ty)を当てはめるとすべての座標の線型化が実現する。(Tx, Ty)では p =

h, $q = r = 0$ として

$$T_x^* = T_x \times \frac{h}{z}$$
$$T_y^* = T_y \times \frac{h}{z}$$

と定義すると、(x, y)の射影変換と乗数が共通化できる。ただし z の射影変換に関しては上記と同様にすると z^*は定数 h に等しくなってしまうので $p = r = 0$, $q = h$, $z^* = h / z$ と定義すると他の座標系の射影変換の乗数となり、演算の共通化が図れる。

その後各ポリゴンの各頂点の(x^*, y^*)を線型補間することにより左右の輪郭点（ポリゴンの各輪郭線と各走査線とが交差する点）の演算をする。次に左右輪郭点を結ぶ走査線上の全ピクセルの(x^*, y^*)を右と左の輪郭点を線型補間することで求める。同様な方法で全ピクセルの(Tx^*, Ty^*)を求める。同様に全ピクセルの z^*の値を求める。ここで求められた z^*を用いで(Tx^*, Ty^*)を逆射影変換して各ピクセルの(Tx, Ty)とすることで単純な線型補間のみで高精度のテクスチャマッピングを実現している。

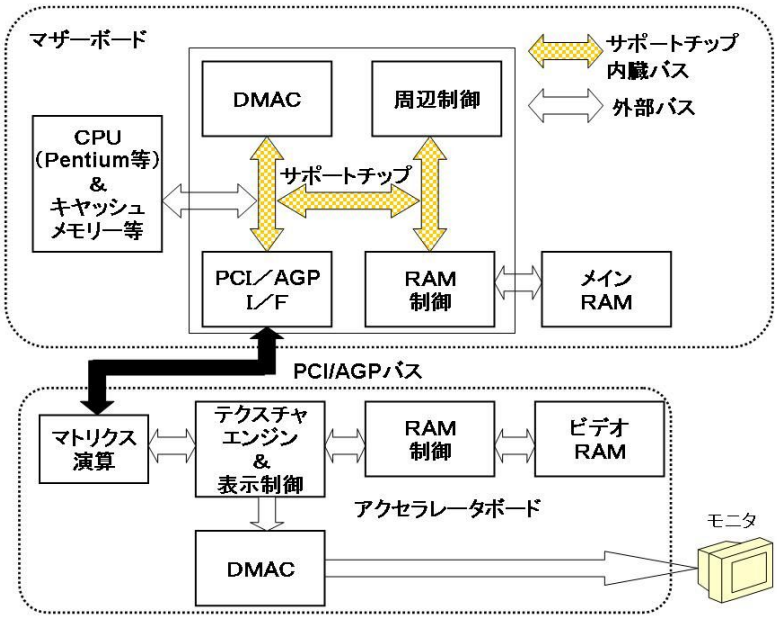


図 3.2-88　PC+グラフィックアクセラレータ構成例

グーローシェーディングもテクスチャデータと同様な補間演算で行っている。通常のグーローシェーディングとは異なり、3 次元空間で線型補間されているのでポリゴン表面上の輝度の変化に対しても射影変換がかかっている。

またテクセルデータ自体をフィールドバッファに書くのではなく、(*Tx*, *Ty*)をフィールドバッファに書きレンダリングパイプラインの最終段でテクセルデータを引くことでパイプラインに流れるデータ量を削減することによりフィールドバッファの容量の削減をも行っている。

ポリゴンハードの黎明期には、各社のこのような各種の創意工夫により、3D ゲームはリアルタイム性を確保してきたが、現在では前述のように半導体製造プロセスの進歩、ゲームマーケットの拡大等々の種々の要因で、ポリゴンレンダリングを比較的素直にハード化でき高精細でフォトリアリスティックな画像をリアルタイムで生成するゲームマシンも登場してきている。今後は、半導体のより一層の進化などで獲得できるリソースを活用してより高精細でよりフォトリアリスティックなレンダリングが実現され、さらに立体視レンダリングなどのようなこれまでにないゲーム表現への応用も期待されている。

(4) 今後のゲームアーキテクチャ

(a) 人工現実感の世界を構成するもの

ヘッドマウントディスプレイに遠近感のあるコンピューター画像を表示し、データグローブと呼ぶ装置をはめてその画像を手の動きで直接操作しながら、音声認識のためのマイクロホンによってさまざまな命令を発する。米国航空宇宙局のエイムズ研究センター（NASA Ames Research Center）では、まだ扱いにくかったコンピューターと利用者のインターフェースを円滑にするためのシステムが開発され、1987 年 10 月には初めて世間一般に紹介された[8]。いわゆる人工現実感、バーチャルリアリティー（VR）の価値とその将来を示す試みが、いよいよ本格的に動き始めたのである。

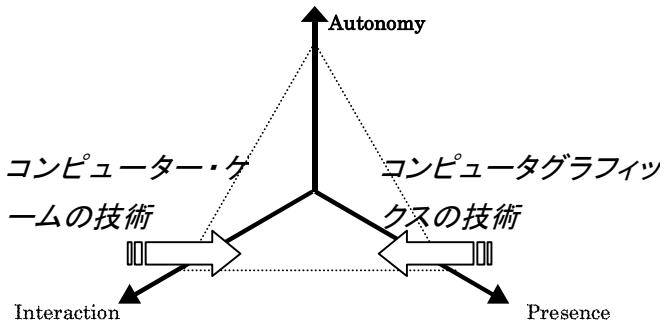


図 3.2-89　AIP キューブ内に示されたグラフィックスとゲーム技術の進歩

この人工現実感の世界を構成するものは、1992 年に MIT Media Laboratory の David Zeltzer 教授が考案した AIP キューブ[9]における、Presence（存在）、Interaction（相互

作用)、Autonomy（自律性）の 3 つの要素であるといわれている。彼自身によれば、Presence とは「コンピューターの作り出す世界とその世界が実際にあるという感覚」であり、システムに求められる臨場感や没入感といい換えることができる。Interaction はいろいろなものをリアルタイムに操作するための能力、Autonomy はどれだけその対象が自律的に動けるか（逆にいえば、人間のコントロールの度合いがどのくらいか）を指していて、これらの性質を軸にして人工現実感のシステムを分類することができる（図 3.2-89)。

ここで興味深いのは、臨場感のある世界を映し出すこと（Presence の獲得）に主眼を置いたコンピュータグラフィックス（CG）技術と、人間とコンピューターとの自然な対話を行うこと（Interaction の獲得）が義務付けられたゲームの技術の、歴史と対比である。1970 年代前半から始まった CG におけるさまざまな技術革新が、ウィンドウシステムに代表される、グラフィカルなユーザインタフェースをごく日常的なものにした（Presence から Interaction の獲得へ)、その一方で 1994 年には、3DO、セガ、ソニー・コンピュータエンタテインメント、任天堂がほぼ同時期に発表した新しいゲーム・コンソール（家庭用ビデオゲームシステムのこと）の高速なグラフィックス、より優れたレンダリング能力、高音質オーディオ、動画機能などが、ほとんどのパーソナルコンピュータを凌駕していた（Interaction から Presence の獲得へ)。

(b) CG とゲームの技術の行き着くところ

たとえば人の動きを人工的に作り出さなくてはならないコンピューター・ゲームや VR 環境などでは、そこに自律的動作を行う能力 Autonomy を獲得することによって、登場させる人や動物といったキャラクターを魅力的なものにすることができる。コンピューター（キャラクター）アニメーションでの人の動きが、物理法則に則ったものになるように計算されていれば、ますます実物に近づいていくことが、すでに Hodgins らの研究[10]によって確かめられている。動きのリアリティーを実現するコンピュータアニメーションの技術などにより、新たなメディアとしてのコンピューター・ゲームや VR 環境が、従来のメディアにない魅力と説得力をもつようになるであろう。

文　献

1. 赤木真澄,『それは「ポン」から始まった―アーケード TV ゲームの成り立ち』, アミューズメント通信社, 2005.
2. Steven L. Kent, The Ultimate History of Video Games: From Pong to Pokemon--The Story Behind the Craze That Touched Our Lives and Changed the World, Three Rivers Press, 2001.
3. John Sellers, ARCADE FEVER The Fan's Guide to The Golden Age of Video Games, Running Press, 2001.
4. PONG-Story - Tennis for Two computer game (http://www.pong-story.com/1958.htm).
5. Van Burnham, Supercade: A Visual History of the Videogame Age 1971-1984, The MIT Press, 2003.
6. Atsushi Miyazawa, SYSTEM FOR SOURCE IMAGE TRANSFORMATION, U.S.Patent 5,077,681, Dec. 31, 1991.

3.2.7　プログラミング　物理・衝突判定

長谷川　晶一
電気通信大学

ゲームを行うためには何らかの場が必要となる。例えば RPG ならば主人公が冒険を繰り広げる街や広野、シューティングやパズルならば自機や敵機が現れたり、ブロックが詰まれたりする画面が必要である。このようなゲームを行う場、ゲーム世界をプレイヤーが理解するためには、ゲーム世界がある程度プレイヤーが慣れ親しんだ世界＝プレイヤーが日常生活している現実世界と似ている必要がある。プレイヤーは類似点を元に画面が何を意味しているのか類推してゲームのルールを理解する。類似点がなければプレイヤーはゲーム画面が何を意味しているのかわからず、ゲームのルールを理解することができない。ブロック崩しを例に挙げれば、ボールが跳ね返る、ボールが当たるとブロックが崩れて消えるなど、現実世界と類似点がある。

3 次元グラフィクス機能が充実し、身体性の高いコントローラーによる自然な操作が可能となった近年のゲーム・プラットフォームでは、リアリティーの高いゲーム世界の映像を作り出し、キャラクターやものをさまざまな角度から見たり、様々に動かしたりできるようにする必要がある。

そのため、ゲーム・プログラミングではその黎明期から現実世界についての知識（＝物理学）を生かしたプログラミングがなされてきた。もともとコンピューターの主目的は微分方程式の数値解を求めること＝物理シミュレーションを行うことであったので、黎明期から物理シミュレーションが組み込まれたゲームが開発されていた。これらの物理シミュレーション処理はゲームプログラムの中に組み込まれた簡単なものであり、ライブラリとしてまとめて再利用するほどの規模も汎用性もなかった。

剛体、流体、布、糸といった基本的な物体モデルを多数組み合わせると、さまざまなゲーム世界をモデル化することができる。近年では、これらのモデルをシミュレーショ

ン可能な物理エンジンと呼ばれるシミュレータが作られ、ライブラリとしてまとめられて複数のゲームに応用されている。本節では、この物理エンジンの構成と技術要素を説明し、アーキテクチャの変遷と今後について記す。

(1) 要素技術

物体や物体内の質量に働く力が求まれば、物体や質量の運動方程式を積分することで運動や変形をシミュレーションすることができる。物体に働く力のうち、重力のように物体や質量の位置姿勢や速度角速度といった状態によらずに与えられるものは、ゲーム世界の設定から直接求まる。しかし、関節に働く力や接触力、糸や布が伸びないように保つ張力は、関節が外れないように 2 剛体上の点の位置をずれないようにするという拘束条件を満たすための力や、物体が互いに侵入しあわないという条件や摩擦の条件や、糸や布の長さが変化しないという条件を満たすように働く力であり、物体の状態（位置・姿勢、速度・加速度）に依存する力となる。

そこで物理エンジンでは、まずモデルの現在の状態から拘束力を求め、次に拘束力を含めて物体や質量に働く力を積分することで物体の運動を求める。

(a) 拘束力の計算

下の図は、拘束力の計算法により既存の物理エンジンを分類したものである。拘束力の計算法には大きく分けて、ペナルティ法と解析法がある。ペナルティ法は、拘束誤差（拘束条件の違反量）に比例した補正力（ペナルティ）を与えることで、拘束条件が守られるようにする手法であり、1 回の計算が単純で高速だが、安定性が保証されておらず、シミュレーションの時間刻み（積分時間Δt）を十分小さくしなければならない。このため結果として計算量が多くなりがちである。

解析法は、運動方程式に拘束条件を制約式として加えて解くことで拘束力を求める。このため、拘束力が正確にもとまれば、常に制約が満たされる。この計算は複雑で計算量も多くなるが、1 回の計算で制約を満たすような拘束力が求まるため、シミュレーションの時間刻み（積分時間Δt）がある程度大きくても安定にシミュレーションすることができる。このため、多くのゲームエンジンがこの手法を利用している。拘束条件を LCP＝線形相補問題（2 変数のうち片方の変数だけが動くという条件下で、1 本の拘束式に速度と力の 2 つの変数が含まれる形式の問題）に帰着させ、近似解を高速計算可能な繰り返し法で解く方法が主に用いられているが、2 体の速度更新計算を繰り返すことで、すべての拘束条件を同時に満たすような拘束力を求める手法（J. J. Moreau の方法）を用いるものもある。

下図の左端の撃力法は複数の衝突を衝突が起きた順番を考えることでひとつずつ計算

する手法である。2 体問題に帰着できるので計算が簡単になるが、短時間のシミュレーションに非常に多くの衝突が発生する状況が起こりえるため、リアルタイムシミュレーションには向かない。また関節や床の上に積み重なった物体のような継続した拘束を扱うことも出来ない。

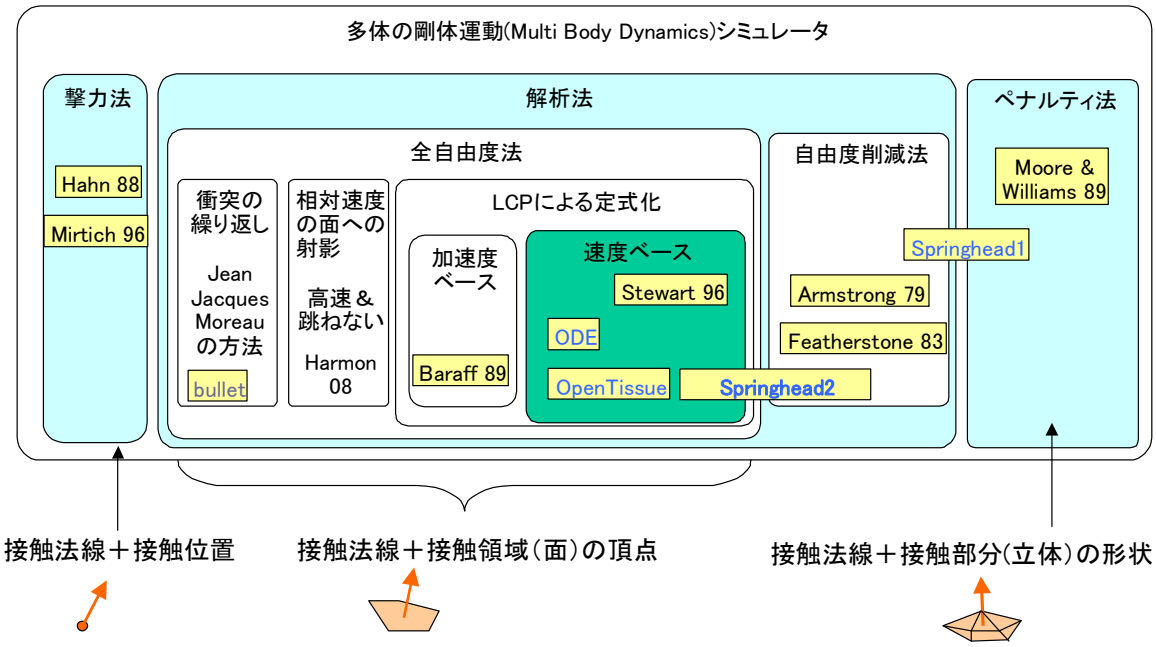


図 3.2-90　接触力計算法によるシミュレータの分類と必要な接触情報

(b)　接触判定

接触力の計算をするためには、まずどの物体がどの物体とどこでどう接触しているのかを調べる必要がある。接触判定はまず大まかに大域的な判定をおこない、接触している可能性がある物体のペアを求め(大域判定)、次に各ペアについて本当に接触しているのか、どのように接触しているのかを調べる(詳細判定)。

①　大域判定の手法

大域判定では、Bounding Volume とよばれる球や直方体のような簡単な形状で物体を囲み、Bounding Volume 同士が接触しているかどうか判定する。

また、時間がかかる総当り判定を避けることによる高速化も行われる。ゲーム世界に n 個の物体があるとき、総当りで判定を行うと n(n-1)/2 のペアについて判定を行うことになる。このため計算量は $O(n^2)$になる。空間を 2 つに分割し、どちらの空間に含まれるかにより n 個の物体を 2 つのグループに分割すれば（両方に含まれる物体は両方のグルー

プに含める)、接触判定はグループ内についてだけ行えばよいことになる。分割後のグループをさらに半分に分割し…という処理を繰り返し行えば、接触判定の計算量はO(nlogn)に近づく。

このほか空間をあらかじめ格子状に分割しておき、同じ格子に体積を持つ物体同士だけの判定を行う手法や、一つの軸に沿って位置に基づいてソートし、重なっている物体のみを判定する手法などが提案されている。いずれの方法も、総当りの計算量 $O(n^2)$を$O(nlogn)$近くまで減らす効果を持つ。

② 詳細判定の手法

詳細な接触判定はどのような形状の物体を取り扱うのかに大きく依存する。また前節の図に示したように、接触力計算の手法によって必要な接触情報が異なる。

へこんだ場所を持たない形状を凸形状と呼ぶ。凸形状は、2 つの凸形状間の距離が極小となる点がひとつしかないので最小距離となる点＝極小となる点となり簡単に求めることができる。このため凸形状についての接触判定アルゴリズムが数多く提案されており、非凸形状も凸形状に分割して判定することが多い。

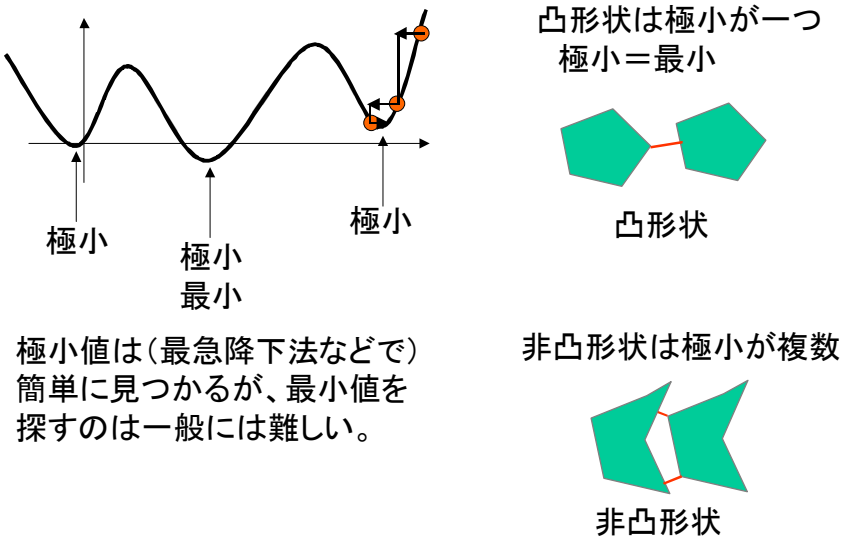


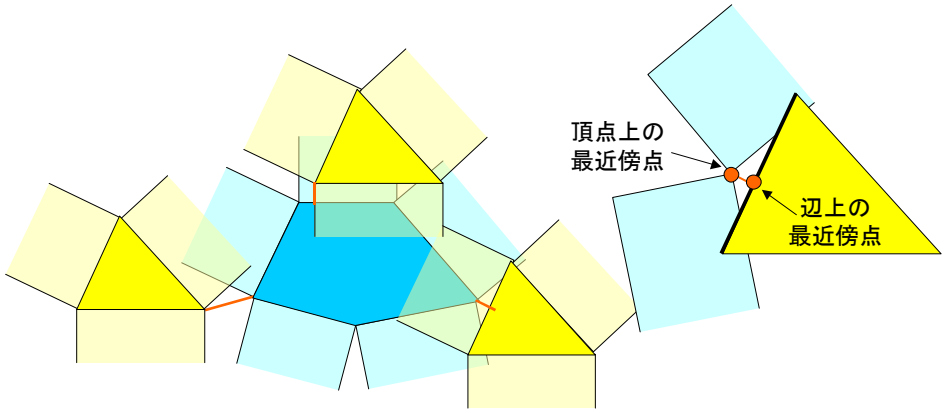
図 3.2-91 最小値・極少値と凸形状の関係

③ 凸形状の接触判定

凸形状同士の最近傍点を見つける手法としてよく使われる Lin-Canny 法(1986)と GJK 法(1988)を紹介する。Lin-Canny 法は凸多面体の接触判定を行うことしかできないが、GJK 法は球体や球体をスイープした形状、円筒や円錐といった曲面を持つ凸形状と多面体の両方を扱うことができる。

Lin-Canny algorithm

頂点/辺/面のボロノイ領域に，もう一つの凸多面体上の頂点/辺/面上の最近傍点が含まれていれば，そのペアが2凸多面体上の最近傍点のペア

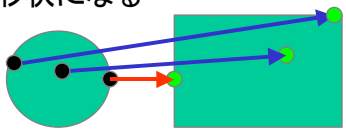


2次元版，5角形と3角形

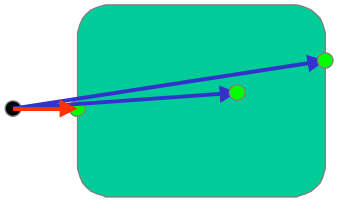
図 3.2-92　Lin-Canny 法

GJK algorithm

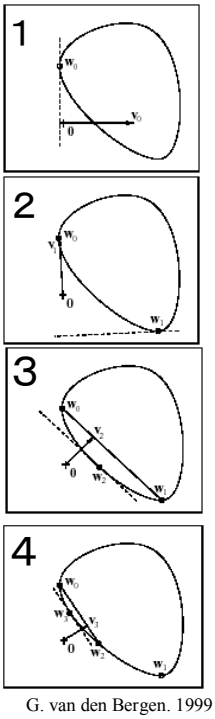
凸形状A上の点から，凸形状B上の点へのベクトルを原点を始点に並べる（Minkowski sum）とベクトルの終点の集合も凸形状になる



元の図形



Minkowski Sum をとったもの



G. van den Bergen. 1999

Minkowski sum中で原点に一番近い点を求める。この点は、support point(与えられた向きの端点）が分かれば、左の手順を繰り返すことで求まる。

図 3.2-93　GJK 法

(c) 連続体のシミュレーション

ゴムやプリンのような柔軟な物体や水のような流体のシミュレーションでは、形状が変形するため、ひとつの物体を多数の要素に分けてシミュレーションする必要がある。このため計算量が多くなるので、本格的なシミュレーションは、リアルタイムシミュレーションを必要とするゲームでは最近まで行われなかった。

① 柔軟物のシミュレーション

柔軟物のシミュレーションでは有限要素法が良く用いられる。これは物体を要素に分割し、要素間に働く力を求め、要素ごとの運動をシミュレーションする手法である。物体を自由に変形させるためには、十分に細かな要素で物体を分割して計算する必要があるため、計算量が多くなる。そこで、変形の向きを限定し、ボーンのシミュレーションとスキンメッシュによって変形を表現する SSD 法（Skeletal Subspace Deformation）や、あらかじめ振動モードごとに変形を計算しておき、各振動モードの励起のシミュレーションから物体形状を求める手法（GPU を用いた DyRT:Dynamic Response Textures などが有名）などの高速化技法も用いられている。

② 流体のシミュレーション

流体のシミュレーションでは、流体ではなく、空間を分割し、その空間に存在する流体の運動をシミュレーションする格子法が用いられることが多かった。この手法は格子同士の接触判定が不要なため格子数あたりの計算量が少ない。

流体を粒子によって表し粒子の運動として流体の運動を表現する粒子法が用いられることもある。空間に対して流体が占める割合が小さい場合には、格子を用いる場合と比べて粒子数が少なくなるため、全体の計算量が少なくなることが多い。最近のゲームでは少量の液体を表現するために粒子法を用いた流体シミュレーションを行うことがある。

(d) 微分方程式の数値積分

微分方程式の数値積分は計算機が登場した当初から計算機の主要な用途だったが、差分化に起因する誤差の影響の議論など、現在でも解決していない問題もあり研究が続けられている。積分誤差が系のエネルギーを減少させる側に生じる後退積分や系のエネルギーの積分誤差が常に一定範囲内に収まるシンプレティック積分を用いると、大きな時間刻みでも安定にシミュレーションできるため、シミュレーション回数が削減できる。一方これらの積分には、積分計算に積分後の時刻に物体に働く力が必要となるためシミュレーション内容によっては計算が難しくなり、1 ステップあたりの計算量が増加する。このため、積分後の力の計算に近似計算を用いるなどして 1 ステップあたりの計算量と安定性によるステップ幅の増大の効果をバランスさせ、効率が最も良くなるような計算法とする。

(2) 年代別ゲームアーキテクチャ

(a) 一体化したゲームアーキテクチャ(～2000 年)

Tennis for Two(1958) や Space war!(1962) のように物体の動きを模したゲームは、デジタルゲームの黎明期から存在していた。これらのゲームでは物理シミュレーション部がゲーム進行や操作入力のような他の部分と特に分離しておらず、ゲームごとに物理シミュレーション部が開発されていた。

(b) 物理シミュレータの分離（2000 年～）

ゲームの映像表現のリアリティーが向上し、3 次元物体の運動表現が必要になると、物理シミュレーション計算の複雑化に伴い物理シミュレーションを専門に行うライブラリが開発されるようになった。

Havok（2000～）、Open Dynamics Engine(2000?～)、MathEngine（1997）といった物理エンジンが開発されるようになり、実際にゲームに採用されるようになった。初期は機能が不十分だったり、不具合が残っていたりするなどしたため、ゲームタイトルの開発にあわせてエンジンの改良を進めていたが、徐々に安定化していった。

このころ、ゲームプラットフォームの多様化やシステムの巨大化に伴い、マルチプラットフォーム対応を謳ったゲームエンジンが開発されていた。当初これらは、ゲームのアセット（データファイル）のロードからグラフィクスやサウンドの提示までの機能を持つものであったが、すぐに物理エンジンやゲーム AI などを取り込んでいった。Unreal Engine のようにさまざまな物理エンジンと提携してそのまま取り込むものや、CryEngine のように独自のエンジンを組み込んでいくものなどが登場する。

(c) ハードウェアとの関係(2006 年～)

Ageia 社は Novodex エンジンをベースに専用ハードウェアアクセラレータを開発し、PhysX として 2006 年に発売している。しかし物理シミュレーションに必要な計算はグラフィクスやサウンドと比べると複雑で特化しやすいものではない。特殊なハードウェアをよりも汎用 GPU やスパコンが持つようなベクトル演算器を必要としている。

実際同時期に GPU を用いて非常に高速にシミュレーションを行う HavokFX の発表があったり、プレイステーション 3 の CELL プロセッサの汎用並列プロセッサでも PhysX が高速に動作するようになったりと、汎用プロセッサの利用が増加している。

(d) オープンソース化(2007 年～)

プレイステーション 3 が発売され、その SDK として Ageia 社の Physics がバンドルさ

れるようになるころには、LCP の解法としてガウスザイデル法を用いる手法や、その計算を剛体ごとに行うことで高速、省メモリ化できることなどといったノウハウが論文や発表資料として公開されるようになり、Open Dynamics Engine などオープンソースの物理エンジンも十分な性能を持つようになってきた。日本のゲームメーカーでも SCE が Bullet Engine をベースにしたエンジンを独自開発したり、各社がタイトルごとに物理エンジンを開発したり導入したりするようになり、物理エンジンに技術に対する不安も解消した。そして物理エンジン単体の販売といった商売が成り立たない環境となった。

このころになると物理エンジンを使いこなしたゲームタイトルも数多く発表されるようになった。初期には、物理エンジンを利用したゲームのほとんどが FPS であったが、コミカルなアクションゲームや RPG といったゲームタイトルにも使用されるようになった。

(3) 今後のゲームアーキテクチャ

本節では、物理エンジンとそのほかの○○エンジンと呼べるようなソフトウェアライブラリ郡の今後の発展の方向と、それに伴うハードウェアへの要求について、筆者の考えを述べる。

(a) 物理以外のシミュレーション

3 次元の剛体運動シミュレーションについては、2006 年ごろには 100 個近くの物体をある程度の精度でリアルタイムシミュレーションできるようになり、ひとまず満足の行く水準に達した。しかしリアリティーの高いキャラクタモーションは物理シミュレーションだけでできるものではない。物理シミュレーションは物体の運動を再現することができるが、キャラクターの動きを作り出すためには、人間や動物の体の運動制御をシミュレーションする必要がある。Natural Motion 社は制御モジュールを多数用意し、タイムラインにしたがって配置することで人間や動物の動作を生成し、リアリティーの高いアニメーションを作成するツール endorphin を 2003 年に発売している。また、Natural Motion 社はこの技術を応用したゲームエンジン euphoria を 2006 年に発表している。筆者らは、2004 年から、物理シミュレーションに加えて、人間の感覚・運動系をシミュレーションすることで、キャラクターの感情を動作により表現することを提案し、研究を進めている。

作品世界でのキャラクターの役割の大きさを考えれば、物理だけでなく、人間や動物の動作をシミュレーションすることは、ますます重要になると考えられる。動作を超えてキャラクターの行動までをシミュレーションによって生成することを考えると、ゲーム AI の議論ともつながってくる。ゲーム AI は、ゲーム進行に合わせて作りこむ必要があるという側面を持つ一方、プレイヤーが納得でき、かつ多様な動作を生成するためには、

妥当な意思決定ルールのシミュレーションによって行動を生成する必要もある。最終的には、キャラクターの感覚・認識・意思決定・行動・動作・運動制御・身体の物理をトータルにシミュレーションし、ゲーム AI とシナリオと協調して動作するようなシミュレータに進化すると考えられる。

また、動作生成だけでなく、加速度センサーやカメラを使った入力や拡張現実感技術が導入されることを考えると、加速度認識、画像認識・合成ソフトウェアが今後重要になると考えられる。これらのソフトウェアも大規模化しやすく、多くのノウハウを含むため、エンジンとしてまとめられ、再利用されるようになると考えられる。

(b) ハードウェア

(a)で述べたように、今後はさまざまな汎用的な処理が必要となるので、CUDA や SPU などある程度汎用的に処理が可能な GPPU(General Purpose PU)の需要が増すと考えられる。また、ゲーム AI など、ゲーム全体の流れに関係が深く、メインメモリ上である程度重たい処理を行いたい部分も増えてくると考えられる。

これらの要求を満たすために、GPPU と映像 IO を持つ画像・物理演算モジュールと、並列 CPU とメインメモリからなる AI・物理モジュールによってシステムが構成されることを予想する。

3.2.8 タスクシステム　技術詳細編

田村　祐樹
株式会社ネバーランドカンパニー

(1) タスクシステムの技術

(a) タスクシステムとは何か？

タスクシステムとは何か？　とはタスクシステムについて耳にした人たちが常に頭に思い浮かべることでもある。

これを「○○である」と断定することは容易ではない。

なぜなら、タスクシステムには「これがタスクシステムである」といった確固たる実装は存在しないからだ。会社単位、ないしはプロジェクト単位において個々に実装される事例もあり、ゲームジャンルによっても要求事項が変化するため、この章において特定の実装を「タスクシステム」として深く掘り下げるのではなく、実用とされている実装の種類と特性について述べる。また、タスクを語る際にキーワードとして頻出する「並列性」という機能に着目する。

ただ、端的に主観を交えて述べるのであればタスクシステムは

「個々が汎用ワークを持ち、並列的に実行処理を差し込めるオブジェクトをタスクという単位で管理するシステム」

と呼べるだろう。

また、多くの場合その実行処理を管理するために TCB（TaskControlBlock）と呼ばれるタスクデータ構造をタスクごとに持つ。

ただし、この説明は場合によっては正しくない。これらのことについて、この章では技術的側面からタスクについて述べる。

また本内容は筆者の経験と調査に基づくものであり、これらがタスクシステムの実装のすべてではないことに留意されたい。

(b) なぜタスクシステムは必要とされたのか？

ゲームがアセンブリで組まれていた時代、ゲームには多数のキャラクターが登場し、プログラマーはオブジェクト単位でプログラムカウンタ（以後 PC と記述）を管理する必要性にかられた。故に、古典的なタスクは PC の遷移を管理するシステムと呼べるものと推察される。

（プログラムカウンタ=PC とは一般的に次に実行するべき命令が格納されているアドレスを指し示すレジスタのこと、インストラクションポインタ=IP とも呼ばれる）

多くのアセンブリ言語は実行単位を言語レベルで保持することはできない。

インストラクションセット（CPU の命令セット）に CALL（サブルーチンコール）、JP（ジャンプ命令）、RET（リターン命令）などが存在したとしても、これらの値が保持され使われるスタックはその都度変化するものだからだ。故に、オブジェクト単位の並列動作を必要とした際、実行中の処理のアドレスをヒープなどに記憶しておき、次のフレームはそこから再開する、というような機能を必要とした。キャラクターA の動作はここから、キャラクターB の動作はここから、といった具合である。これらはまさに「並列性」を主目的としたと考えられる。

ただし、PC だけの制御を行うのであれば単なるジャンプテーブルと変わりない。故に、PC の管理＋α の機能を有した。この＋α にしては後述する。

(c) 現在なぜタスクシステムは使われているのか？

端的に言い表せば、言語がアセンブリから C に変化したとしても、作り上げるゲームは変わらないため（寧ろより複雑になったため）実行単位を管理する必要性は失われなかった。

ゲーム開発がアセンブリから C に移り変わった際、言語における実行単位はラベルやアドレスから主に関数アドレスを意味するようになった。故に、アセンブリのときのよう

に PC 自体を管理するのではなく、関数単位で実行を管理するようになった。関数を登録する仕組みをつくり、周期的に指定の関数が実行されるようにしたのである。

C では関数が第一級オブジェクトではない（実行時に構築できず、変数に格納できない）ため、多くのプログラマーは関数ポインタを使いそれらを表現、実現した。

故に C におけるタスクの実装ではほぼ例外なく関数ポインタで実行単位が管理される。

ただし、アセンブリを利用することで C や C++において関数内などで中断するような事も可能になる。これはインラインアセンブリなどで PC を操作することで可能となり、マイクロスレッド（=ファイバー）という実装に一部みることができる。（が、この仕組みは、PC だけでなくスタックやレジスタ状態をも管理しなければならない）

(d) 未来でもタスクシステムは必要とされるのか？

一般的なゲーム開発で採用される言語は多くの場合 C と C++だが、プラットホームによっては Java、C#（ないしは他のスクリプト言語）ということもある。

C と C++、Java、C#、を隔てるのは実行単位を制御するのにおいて「関数ポインタ」という概念を（必ずしも）必要としない、ということだ。

C++では関数ポインタを利用できるが、殆どの場合は仮想関数を用いて実行をオーバーライドすることが実行単位制御の適切な解答となる。Java、C#においても実装上仮想関数という概念はないが、それらは C++と同様にオーバーライドによって実行処理を切り替える。

これらはクラスに対して行われるため、どちらかといえば実行単位を管理するというよりもクラスを管理するものになるといえる。

故に主流が動的言語になるとすれば、実行単位が関数オブジェクト単位になることも十二分にあり得る。

(e) タスクシステムの優位性とは何か？

並列性を確保しつつ実行を管理できることに尽きる。また、副次的作用に目を向ければ、ソースコード上の可読性を向上させ、構造をシンプルに保つことができる。

例えば、タスクで管理されるキャラクターの実装に限っていえば、

```
PlayerExecute();
FriendExecute();
NpcExecute();
EnemyExecute();
```

と並んでいるよりも、

TaskExecute();

などのようにタスクに任せてしまうことでキャラクターの種別単位での実行関数を差し込むような制御を必要としなくなる。

また、マップやカメラ、その他キー入力などのデバイスをさわるような処理までタスクとして管理することでそれらをキャラクターと協調させながら、タスク構造の中で管理することができるようになる。

(f) タスクシステムのデメリットとは何か？

まず一つとして粒度が問題となる。ありとあらゆるものをタスクにすることは可能だが、際限なくすべてをタスクとして扱えば、実行単位が増大することで呼び出しのオーバーヘッドを生じ、管理の手間を増やす。

また、その動的性質により実行負荷になっていたとしてもそれに気づきにくいということが挙げられる。加えて解放や停止を損なうと意図しないタスクが動いてしまい、動作に不具合を生じるという欠点がある。

しかしながら、これらは「タスクの管理画面」や「タスクの個の負荷計測」といった実装を行うことによりデメリットを軽減することができる。

タスクの管理画面においてはタスクの生存状態やメモリ消費量などを表示し、タスクの個の負荷計測ではどのタスクが負荷となりボトルネックとなっているかを検出することができるため、多くの場合これらデメリットによってタスクを採用しない、という事はない。

さらにいえばタスク同士の連携における生存期間の問題点が挙げられる。多くのものをタスクとした場合、それが生存期間を共にする親子ではなく並列に動作するが故に一方的に参照する事がある。一例として、ホーミングミサイルのようなものをタスクで実装し、ターゲットのキャラクタータスクを追従するために参照する場合、何らかの理由によってホーミングミサイルの到達以前にキャラクターが消滅してしまう事がある。(他の攻撃によってキャラクターが倒されるなど)

そういった際に、既に有効ではないタスクをポインタで参照していることによるメモリ破壊の危険性が生ずる。この例でいえば、ホーミングミサイルが有効ではないタスクから標的の座標を得ようとすれば、無効なポインタから関数を呼び出してしまうだろう。デメリットとしては深刻ではあるが、安全にポインタを扱う方法、例えばハンドルやスマートポインタといった技術を使うことで有効なタスクを判断することができ、常にタスクを安全に参照することができる。故にこれもタスクシステムを使わない理由とはならない。

考え得る一つの大きなデメリットとして「主プログラマーが把握できない勝手に動いている処理があってはならない」という心理的障壁があげられる。タスクシステムの動的性質により、個のプログラマーはあらゆるシーンにおいて実行処理を差し込むことができる。

経験上、これらのことを極端に嫌うプログラマーがいることが挙げられる。その場合は、あらゆる特殊処理は関数単位として挟み込まれることになり、プログラムの動的性質は失われ、ソースコードの可読性は低下する。

たとえば特定のステージでのみ動く特殊処理があった場合、

```
if (stageId == STAGE_D01_02) {
    SpecialExecuteD01_02();
}
```

などといった実装を強いられる。これらをタスクで解決するならば、特定のステージのみそのタスクを実行処理に乗せればよく、これらのような表記は必要ない。

なお、これらは「タスクを使わない」としたプロジェクトにおける記述者の経験に基づく。無論、このように特殊処理であれば明示が良いという意見もあり、どちらが正しいとは述べない。しかしながら、タスクシステムを採用するプログラマーの思惑の一端がこれら動的処理（実行単位）の差し込みにあることは想像に難くない。

(g)　タスクシステムの基礎と＋α

タスクシステムの基礎実装としてはタスク自身が汎用ワークを持ち、かつまた実行アドレスを持つことはすでに述べた。

アセンブリならばスタックと PC の操作を行う構造で実装し、

C であれば、構造体と関数ポインタ、

C++であればクラスと仮想関数で実装することが主たる実装となると考えられる。

ただし、これだけを実現したのではタスクシステムは利便性の高いものとなりえない。

以降は実行管理に付随する機能を列挙する。

① タスクの ID や名前

タスクに対して名前をつける、ID を割り振るなど、デバッグ時の助けになる情報を付与することがある。この機能により、特定のタスクがボトルネックになっている、特定のタスクが生成されすぎている、といったようなことを知り解消するための手がかりとできる。

また、ID を使ったタスクの検索や停止といった操作を行う機能がある場合も多い。

② タスクの可変長汎用ワーク

TCB とは異なる汎用ワークを持つ。

アセンブリでは固定ワークが割り当てられる、ないしは要求ワークが割り当てられる。

C の場合は TCB を先頭に持つ struct などをタスク＋ワークとして扱えるようにする実装が考えられる。

C++であれば継承先クラスで宣言、および確保することが多い。

③　タスクのカテゴリーと実行優先順位

実行関数を持つタスクを生成した場合、タスクを無作為ないしは生成順で実行させることはゲームの進行管理上思わしくない。故に、これら実行順序を管理するためにタスクにはカテゴリー（キャラクター、飛び道具、マップオブジェクト）などといった種別を付与することが一般的である。

加えて、同じカテゴリー内においても実行優先順位を必要するケースは多く、たとえば、同じキャラクターであるプレイヤー、フレンド、エネミーといったものが同一カテゴリーにある場合、これらを実行優先順に実行させるといった実装が行われる。（ただし、どのような順番で呼ばれようとも問題ない実装をしている場合、制御しないこともある）

④　タスクの実行単位

通常の実行関数だけでなく、生成時に呼び出されるコンストラクタ、終了時に呼び出されるデストラクタ、など様々なタイミングを定義する機能を有する。

C++ではクラス自体のコンストラクタ、デストラクタでもまかなえるが、そのなかでは仮想関数を呼び出すことができない、という制約があるため、独自に用意することがある。

細かく実行を制御するため、「生成時関数」「実行関数」「消滅時関数」の他に「いつでも実行可能」「実行関数の前に実行」「実行関数の後に実行」「同一カテゴリーの関数がすべて実行された後に実行」などといった細かな実行単位を持つタスクの実装も実在する。

⑤　タスクの親子構造

タスク同士が木構造を持っている場合がある。親子となったタスクは必ず親の後に子が実行されるようになり、親が破棄された際に子も破棄されることを保証する。例えば、アクションゲームにおけるキャラクターと装備武器が共にタスクで実行されている場合、キャラクターは親で武器は子となる。

⑵　実際に使われたタスク

前章ではタスクシステムに採用されうる実装を紹介した。ここでは筆者が出会った、ないしは作成した実際のゲームにて採用実績のあるタスクシステムの実装の概要紹介を記す。

(a) カスタム Z80 アセンブリによる TCB タスク実装

2000 年頃に RPG で使われた実装。

複数のタスクの並列処理を行い、子タスクの概念を持つことができる。

現在処理を行っている PC（プログラムカウンタ）とは別にフレーム処理開始アドレスを保持しており、RET（リターン命令）を呼び出すことで現在のフレームでの処理を終了し、次フレームより先だって保存してあると記した「フレーム処理開始アドレス」から処理を開始する。

タスク領域はメモリマッピングされプールされた領域から空きタスクを返すことで生成としていた。

先頭には TCB を有し、それをタスク管理ワークと称する。

タスク管理ワークは 32byte で管理され、後半 16byte はユーザーが任意に利用できた。

管理ワークには PC や開始アドレスのほか、連結元タスク、連結先タスク、親タスクなどといった連結リストを管理する情報が含まれていた。

実装としてはつど PC を変更することで継続、中断といった機能を実現する。

Task_Chg_Next ……フレーム処理開始アドレスを現在の位置に設定し処理を継続する

Task_Chg_Ret ……フレーム処理開始アドレスを現在の位置に設定し RET する

Task_Chg Label ……フレーム処理開始アドレスを指定ラベルの位置に設定する

などのように、アセンブラに付随したマクロ機能を利用し関数であるかのように実装されていた。

親子タスクを生成することもでき、

Task_Wait_Child のような子タスクの終了待ちも備えていた。

アセンブリにおける TCB を利用した基礎的実装といえる。

(b) ARM7 C によるタスク実装

2002 年頃のアクション RPG における実装。

初期化（init）、実行（exex）、終了（exit）、メッセージ（proc）を備えた構造体をタスク管理構造体として扱うものとして実装された。

タスク生成構造体に各種関数ポインタなどを設定し、TaskCreate 関数に渡すことで実態を生成する。

実装は prev と next を有する双方向リストで各リストが繋がれており、各々のリストは parent と子タスクの先頭、後方のポインタを有する。

メッセージとはデザインパターンにおける Observer と似た性質を持ち、横断的に情報を伝達する仕組みを持っていた。

例えば、ポーズがかかるとタスクに通知が為されるなどの挙動を可能とする。

タスクワークは malloc にて確保され、タスクの消滅と共に free で解放された。

(c) ARM9 C++によるタスク実装

2004 年頃の RPG における実装事例。
自前の双方向リストコンテナに挿入できるエンティティを継承した class として実装。
class Task を継承したクラスを new して自動でタスク管理に加わる。
初期化と終了はコンストラクタとデストラクタに任せ、継承先にて仮想関数である

```
virtual void execute()=0; // 継承したら実装しなければならない
```

を実装することで実行する。
親子といった概念を持たず、キャラクターやマップなどのカテゴライズされた種別単位で実行するだけのものであった。

(d) ARM9 C++によるタスク実装 2

2008 年頃のアクション RPG における実装事例。
class Task を継承することで実行される。
任意のプロセスカテゴリに追加することができる。
初期化と終了では、コンストラクタとデストラクタ内では仮想関数が呼べないという点を補うため、仮想関数 initialize、termination を持ち、これらで別途初期化処理と終了処理を実装することができる。
挿入の頻度によってデータ構造を使い分け STL の vector と侵入式 list（boost の intrusive に近しい）を利用している。
指定カテゴリー以上のタスクを一度に休止させることができ、デバッグ時などに可動プロセスレベルを指定することで実行レベル以下のものすべてを動かさないようにできる。
親子などの概念を持ち、階層構造で管理される。

(e) C と C++における違い

C における関数ポインタは再代入を許すため関数を動的に差し替えられることが利点である。ただし、その反面どの関数ポインタがどのタスクに割り付けられているのか、それがどのタイミングで変化するのか、といった動的な変異をソースコードから読み解くことが難しくなる。

C++における仮想関数は再代入を許さずクラスとして処理が完結するため、適切に設計されたクラスであればそれを読み解くことが比較的容易くなる。反面、動的に処理を切

り替える事は難しくなるため、実行処理を切り替えたいという時には実行関数の中で Strategy や State といったデザインパターンを用いた方が良い。もしオブジェクト指向に基づくのであれば実行の動的切り替えをタスクに持たせることは振る舞いの制御をタスク機能で行うことになり責務が大きすぎると考えられる。

言語仕様上関数ポインタを使うことも可能だが、さしたる理由がなければ仮想関数を使うべきだろう。

(3) タスクシステムの未来

この章ではタスクシステムの未来について考える。

＋αの機能を考えたとしてもタスク自身の機能を高めるわけではないため、

本来の性質として色濃い並列性と実行単位という側面から類似する機能を持った技術を列挙する。

(a) スレッド

アセンブリで記述された時代には並列性の確保にタスクを用いたが、現在ではスレッドという仕組みを使うこともできる。スレッドは多くの場合 API レベルでサポートされ、マルチスレッドである状態をタスクと同等に扱うこともできる。

しかしながら、真にスレッドが並列実行される場合、排他制御の問題をはらみ、Mutex のようなスレッド同期のための排他オブジェクトを利用しなければならなくなる。

また、利用するライブラリやコードがスレッドセーフ（リエントラント）な構造であるかどうかを意識して使う必要があり、同期を正しく理解していない場合不可解な挙動を示すためタスクほど気軽に扱うことができない。

(b) マイクロスレッド（ファイバー）

マイクロスレッドは同期的に実行できるスレッドのような挙動をする。C や C++などでの実装的にはプログラムカウンタやスタックポインタをアセンブリによって差し替え、レジスタを保持し、CPU の実行箇所を差し替えるというアセンブリ時代のタスク処理に近い挙動を示す。（これらの実装には setjmp、longjmp が用いられることもある）

これにより、

```
for (int i = 0; i < 100; ++i) {
    move(1,1);
    wait(); // マイクロスレッド中断を行う
    // 次のフレームはここから始まる
}
```

といったような、for 文の途中で次のマイクロスレッドへ実行を移すといったような挙動を可能とする。

欠点としては、タスク単位で個々のスタックを持つため、スタックオーバーフローの可能性を高め、そのために多くのメモリを割り付ければメモリ消費量の面で難が発生する。

ただし、個にスタックやレジスタ状態を持たなければならないのは通常のスレッドも同様であるため、そうした点ではメモリ資産がある環境であれば問題とはならない可能性が高い。

スレッドに比べるとより協調的な動作を行うため、同期についてプログラマーが意識せずに済むという利点がある。

(c) コルーチン

コルーチン（co-routine）とはプログラミングの構造の一種である。コルーチンは協調的処理を得意とし、処理を中断したあと続きから処理を継続できる。

接頭語の co-は強調を意味する。

例えばスクリプト言語 Lua にはライブラリとして組み込まれた coroutine と呼ばれる並列性を保つための仕組みがある。

coroutine は yield（中断）と resume（再開）という機能を持ち、関数の実行中などに C/C++に対して処理を戻すことができる。

加えて親である Lua のステートと変数空間を共有する。

厳密にはタスクと＝ではないが並列性実行単位という側面だけを取り出せばタスクと類似の機能を coroutine という単位で扱うことができる。

これもまたスレッドと近しい機能であるが、より協調的であるためスレッドよりも簡易的に扱えるという利点がある。

Perl や Ruby などでもライブラリとして実装されていることがある。

本質的にファイバーと同じ概念ではあるが、コルーチンは言語機能レベルで提供されることが多い。対して、ファイバーはシステムの（アドレスといったような）低レベルの機能を使う。

(d) 近代的なタスク

現代のタスクにおいて決定的に不足しているという機能がないため、近代的なタスクとはより＋αの機能が豊富になったものであるといえる。それはどのようにタスクが使われるかといったような用途により変化するものであり、ここでは近代的な機能について述べることはしない。

しかしながら、実行単位を管理する仕組み自体は失われることはなく、それはタスクシステムと根底を同じくするものとして残るものと考えられる。これらの仕組みがフレー

ムワークなどに取り込まれた場合、ユーザーはそれがタスクであると意識することなく、実行単位を制御できるようになるだろう。

Gamebryo、MT Framework などといったフレームワークはこれらタスクやスレッドのスケジューリング機能を備えるとされている。

(4) まとめ

最初にも述べたとおり、タスクシステムにはこれといった決定的な実装はない。故に技術的な内容に絞ってタスクについての特性を述べた。

また、作成するゲームの内容から必ずしもタスクを利用する必要はないといえる。ただし、実行単位を制御する、という目的から何らかの管理機構を構築するのであればそれは広義でのタスクと呼べるだろう。

開発者にはゲームジャンルやプロジェクト規模、または開発言語などによって適切な機能をもったタスクを実装し利用することが求められる。

また、並列動作だけであればスレッド、マイクロスレッド、コルーチンといった代替技術も存在する。そうした技術を用い、より汎用的、抽象的になった実行単位管理の仕組みはより大きな構造ないしはフレームワークなどに取り込まれ、タスクシステムの技術は開発者がそれと意識せずとも利用できるようになりこれからも深く関わっていくことになるだろう。

参考文献：

［1］やねうらお： “Windows プロフェッショナルゲームプログラミング 2”，秀和システム pp.65 - 115，(2003)

［2］松浦 健一郎： “ゲームのためのタスクシステム”，C MAGAZINE 12 月号 ソフトバンク クリエイティブ pp.64 - 78，(2004)

［3］松浦 健一郎/司 ゆき： “ゲーム・ノ・シクミ シューティングゲーム編 第 11 回 C++によるタスクシステムの実現”，C MAGAZINE 1 月号 ソフトバンク クリエイティブ pp.135 - 143，(2006)

［4］松浦 健一郎/司 ゆき： “シューティングゲーム プログラミング”， ソフトバンク クリエイティブ pp.48 - 104，(2006)

3.2.9　タスクシステム　技術遷移編

大野　功二
オープランニング

(1)　タスクシステムの起源

(a)　タスクシステムとは？

「タスクシステム技術詳細編」でも説明した通り、タスクシステムはノンプリエンプティブ(協調的)なタスクシステムの一種である。このタスクシステムは、ゲーム会社が持つタスクシステムのライブラリ、ゲーム仕様、そして、実装するプログラマーなど、環境によってアルゴリズムおよび実装が異なる。

そこで「タスクシステム技術遷移編」では、タスクシステムの実装を分類し、また歴史的な位置づけを行うことで、ゲーム業界でタスクシステムがどのような影響をもたらしたのかを解説する。

(b)　タスクシステムの種類

本書で扱うタスクシステムを下記の 3 種類に分類した。

① タスクシステムのベースとなったシステム

1980 年代頃に、自機の移動や爆発などのオブジェクトをシーケンス管理するシステムが考案されたと推測される。これは、処理速度を向上させるだけでなく、プログラムの可読性およびメンテナンス性を大きく向上させるためのシステムでもあったと考えられる。

1980 年代から 1990 年代前半まで、アーケードゲームの基本システムとして、アセンブラベースで実装されたとされる。

② TCB を使ったタスクシステム

書籍などで紹介されるタスクシステムの代表的なアルゴリズムは、「TCB(タスクコントロールブロック)」と呼ばれるデータ構造またはクラス構造を使用したシステムである。

TCB では、TCB を指すポインタを使ったノード型のデータ構造で管理される。

また、TCB の実装では TCB の先端と終端を接続してループ構造にして実行する。

TCB を使ったタスクシステムは、様々に拡張された派生型のタスクシステムを多く生み出した。

TCB によるタスクシステムの実装では、自機や敵などのオブジェクトだけではなく、キー入力イベント処理や描画管理処理までを 1 つのタスクとして扱うなど、徹底して管理する実装が多く見られる。

③　その他

この 2 つの系統と異なるタスクシステムとしては、ループ型のデータ構造を持たないタスクシステムである。代表的な例は、下記の通り。

・ TCB を配列として管理するタスクシステム(TCB でループ処理は行わない)
・ TCB を TCB のポインタで管理するタスクシステム(TCB でループ処理は行わない)
・ C++の STL・List テンプレートを使ったタスクシステム
　または、それに類似するアルゴリズムを使用したタスクシステム

補足：タスクの粒度

タスクシステムを使用して開発している現場では、例え同じ会社で同じタスクシステムのライブラリを使っていても、タスクにする「処理の粒度」が異なる場合がある。

例えば、自機・敵・弾をタスクの 1 粒度として扱うだけではなく、ゲーム上に表示されているスコアやゲージなどもタスクとして分割し管理する場合もある。

また、メニューなどであれば、極端な例では 1 つのメニュー項目に対して 1 つのタスクを割り当てた事例も存在する。

粒度を小さくすればするほど汎用性は高まるが、同時に処理負荷の増大やメンテナンス性の悪化を引き起こす原因ともなる。

(2)　タスクシステムの歴史　その 1(1980 年代)

タスクシステムの技術遷移を紐解くために、タスクシステムの歴史について調査した。

タスクシステムの技術遷移が分かりやすいように 10 年単位で、ゲーム、書籍、インターネットでのタスクシステムの扱われ方を時系列で表記した。

タスクシステムは 1970 年代にアーケード業界で開発され、1980 年代に多くのゲームに実装された。

(a)　ギャラクシアン

1979 年 10 月発売　ナムコ(現在のバンダイナムコ)

タスクシステムが使われたプログラムとして、Web サイト「Logician Load」(Peto 氏)で紹介されている。同サイトでは、ギャラクシアンで使われたであろうタスクシステムが、リバースエンジニアリングによって、後の TCB を使ったタスクシステムに発展したと記している。(ただし、本当にギャラクシアンがタスクシステムを使用したかについては、確認は取れていない。)

(b) タスクシステムの伝搬

タスクシステムの伝搬は、下記の 3 つの経路があった。

- ゲーム開発を子会社や外注に発注した際に、ソースコードの買い取りによって獲得
- ゲーム会社間での人材移動によって獲得
- リバースエンジニアリングによって獲得

上記のいずれかの方法で、タスクシステムは関東圏・関西圏で広まったと考えられる。

また伝搬の特徴として、「TCB」と名称されるデータ構造を使ったタスクシステムの事例は、関東圏より関西圏で多く見られる。

(3) タスクシステムの歴史　その 2(1990 年代)

1980 年代終わりから、1990 年代初頭にはゲーム会社間での人材移動が活発に行われた。

またゲーム開発の外注委託も活発に行われていた。

これに伴い、さらにタスクシステムは多くのゲーム会社へと広まった。

(a) 格闘ゲームブーム

1990 年代に開発され発売された格闘ゲームの多くは、TCB を使ったタスクシステムを使って開発されたとされる。

(b) NOON

1996 年発売 マイクロキャビン / 大野功二

筆者が企画・プログラムを行ったアクションパズルゲーム「NOON」において、ゲームのメニューやゲームモードのシーン遷移を「タスクマネージャー」と名付けたツリー型ノード接続タイプのタスクシステムで実装を実現した。

構造としては、TCB に相当するタスクを、タスクリストと呼ばれるツリーデータに登録して使用した。このツリーデータはループ型でもあり、タスクマネージャーはメインループの一部として動作するように設計されていた。

また、キャラクターやパーティクルシステムに置いても、TCB を使ったタスクシステムに近いアルゴリズムを使用した。

これらの開発は筆者が独自に開発したものであり、他社の技術継承は一切ない。

当時、C 言語がゲーム開発において一般的に使われるようになっており、同時多発的に

様々な会社やプロジェクトでオリジナルのタスクシステムが開発されたものと考えられる。

(c) ライブラリで作る簡単ゲーム工房

1999 年 10 月発売　工学社

この書籍は、著者が独立後に開発した C 言語ベースの 2D ゲームエンジンライブラリ「AXIS」を使ったゲーム開発の解説本である。

マイクロキャビンの許可を得て、タスクマネージャーを公開して解説した。また、NOON の開発で得たキャラクター処理やパーティクル処理を、タスクシステムとして共通のライブラリで扱えるように「ファンクションマネージャー」を開発し、同書籍に置いて発表・公開している。

ファンクションマネージャーでは、TCB に近いデータ構造である「ファンクション」という構造体とそのデータポインタでノード接続を実現している。

なお、ファンクションマネージャーは部分的に使われるタスクマネージャーであり、ループ構造は持たない。

現在、AXIS は C++言語ベースへと移行しており、ファンクションマネージャーは C++の STL・List テンプレートへと置き換わっている。

(4) タスクシステムの歴史　その 3(2000 年代)

2000 年代になると、ゲーム開発のプログラム言語が C から C++へと移行した。

また、インターネット上や書籍でタスクシステムの公開が行われ、タスクシステムがゲーム会社だけではなく、一般のアマチュアゲームプログラマーなどにも浸透した。

(a) Logician Lord

2000 年 6 月 10 日更新　Peto 氏

インターネット上で TCB を使ったタスクマネージャーの紹介と解説を最初に扱ったサイトである。現在、このサイトは閉鎖されており The Internet Archive でのみ閲覧が可能である。

コンピューター・ゲームのからくり

http://www.hh.iij4u.or.jp/~peto/Games/games_top.html

http://web.archive.org/web/20041009222313/www.hh.iij4u.or.jp/~peto/Games/games_top.html

このサイトではタスクシステムの歴史について、ナムコのギャラクシアンをアイレムがリバースエンジニアリングで獲得し、後に技術者の移籍によって SNK に伝搬したと紹介している。また、このサイトで説明されている TCB を使ったタスクシステムは、関西圏で見られるタスクシステムの構造・機能に酷似している。

このサイトは、後にタスクシステムの紹介をするサイトや書籍の参考になったであろうと考えられる。また、サイトによっては参考サイトとしてリンクが貼られている。

(b) White Paper(ほわいと・ぺーぱー)

2001 年 11 月 23 日公開 Moha 氏

タスクシステム

http://homepage3.nifty.com/moha/

このサイトで扱っている TCB を使ったタスクマネージャーの解説は、ほぼ Logician Lord(Peto 氏)の内容と同じである。ただし、未使用 TCB の検索はスタック方式である。

またプロフィールから住所が大阪であることが分かるので、Logician Lord のサイトを参考にされたか、または、関西圏のゲーム会社で TCB を使ったタスクシステムの影響を受けていると考えられる。

(c) Windows プロフェッショナルゲームプログラミング 2

2003 年 9 月 1 日発売　やねうらお氏

TCB を使ったタスクシステムを初めて解説した書籍である。

この書籍の発売がきっかけとなり、プロおよびアマチュアプログラマーのタスクシステムに影響を与えたと考えられる。

内容は、タスクシステムの歴史からはじまり、「Logician Lord」(Peto 氏)と同様に「ギャラクシアン」を例に挙げている。また、「古典的タスクシステム」と、「近代的タスクシステム」との 2 つのタスクシステムの実装方法を紹介している。

また、やねうらお氏はゲーム用 SDK「yaneSDK」を開発しており、そのサンプルコードの中にタスクシステムのプログラムがある。

yaneSDK(サンプル 10 がタスクシステムのサンプルコード)

http://yaneurao.hp.infoseek.co.jp/yaneSDK3rd/

このタスクシステムでは「Logician Lord」(Peto 氏)が紹介している TCB との共通点も見られるが、限定的な使われ方に止まっている。また C++でプログラム（近代的タスクシステム）されており、TCB は STL・List テンプレートで実装している。

(d) 「最新 DirectX 実践テクニック　Part3 Managed DirectX とタスクシステムを使ったゲーム・プログラミング」

C MAGAZINE 2003 年 12 月号掲載　本多 直人氏

TCB を使ったタスクシステムが初めて C MAGAZINE の紙面にて紹介された記事である。このタスクシステムでは「Logician Lord」(Peto 氏)が紹介している TCB との共通点も見られるが、限定的な使われ方に止まっている。

また C#でプログラムされており、TCB はノード接続さている。

(e) 「ゲームのためのタスクシステム」

C MAGAZINE 2004 年 12 月号掲載　松浦 健一郎氏

この記事で松浦 健一郎氏が紹介しているタスクシステムでは「Logician Lord」(Peto 氏)が紹介している TCB と酷似しているが、タスクの優先度を使用していないなど限定的な使われ方に止まっている。

なお、下記の雑誌・書籍にて、このタスクシステムは繰り返し紹介されている。

またタスクシステムのアルゴリズムに改良も施されている。

「ゲーム・ノ・シクミ シューティングゲーム編 第 11 回 C++によるタスクシステムの実現」

C MAGAZINE 2006 年 1 月号掲載　松浦 健一郎氏/司 ゆき氏

「シューティングゲーム プログラミング 」(単行本)

ソフトバンククリエイティブ　2006 年 9 月発売　松浦 健一郎氏 / 司ゆき氏

2004 年以後のプロおよびアマチュアプログラマーのタスクシステムに影響を与えたと考えられる。

(f) 「2ch タスクシステム総合スレ part.1」

2007 年 3 月 12 日　スレッド作成

日本最大の掲示板である「2 ちゃんねる」でもタスクシステムの情報が公開されている。

ここで注目すべきは、長 健太氏 (ABA."Saba")のオリジナルシューティングゲーム”Titanion”(2006/11/23)で使われているアルゴリズムが、タスクシステムの代価アルゴリズムとして紹介される点である。

現在でも、このスレッドでは継続してタスクシステムについて考察が行われている。

(4) MT Framework

CEDEC 2006「次世代機に向けたゲームエンジンの設計」
株式会社カプコン 第二制作部ソフトウェア制作室
石田　智史氏

CEDEC で公開された MT Framework は、CPU および OS のマルチタスク、マルチスレッド機能を最大限に生かすことを目的に作られたゲームエンジンである。

タスクシステムと同様に、プレイヤーや敵をタスクとして管理し、さらにマルチスレッドで処理を行う。これによりデッドライジングでは、6 スレッドを同時に実行することで、1 スレッドで実行した場合よりも 2.6 倍の高速化を実現している。

タスクシステムはメモリ容量の大きいアーケードのハードウェアで開発された。そのため、「タスクシステム」と、現在のコンシューマー機などで一般的に行われる「動的にメモリを確保・解放するデータ」との連動は、バグの発生原因となることが少なくなかったと考えられる。

そこで MT Framework では、ゲームプログラム内のクラスを System,Resource,Task の 3 つに分類して、システムとタスクおよびデータの整合性を監視することで、安全に管理している。これによりプログラムの並列化処理に伴うバグの発生を抑えていると考えられる。

明言はされていないが、MT Framework はもっとも発展したタスクシステムであるといえる。そして、タスクシステムの将来をさらに広げたといえるであろう。

(5) タスクシステムの未来

今後、タスクシステムの一部の機能は、C++を代表とするオブジェクト指向言語とテンプレートライブラリに完全に吸収されるであろう。しかし、さらに CPU の高速化とマルチコア化に伴い、MT Framework のようなタスクおよびスレッドマネージメントシステムとして重要性が増すことは間違いないといえる。

また、現在注目されている関数型スクリプト言語が導入されることで、さらに並列処理を効率的に行うタスクシステムの登場も期待されるであろう。

(6) タスクシステムの歴史年表

年	出来事
1979	ギャラクシアン(ナムコ)発売。後にタスクシステムを最初に使ったゲームとの推測がインターネット上で広がる。 この時期にタスクシステムの原型となるシステムがゲーム業界において開発されたと推測される。
1983	ゼビウス(ナムコ)発売。ファミリーコンピューター(任天堂)発売。
1990	スーパーファミコン(任天堂)発売
1991	ストリートファイターⅡ(カプコン)発売。格闘ゲームがブームになる。
1994	SEGA Saturn (セガ・エンタープライズ)発売。 Play Station(ソニー・コンピュータエンタテイメント)発売。
1996	セガサターン版「NOON」(大野 功二/マイクロキャビン)発売。
1999	Dreamcast(セガ・エンタープライズ)発売。 「簡単ゲーム工房」(大野 功二/Country Fox:工学社)発売
2000	Play Station2(ソニー・コンピュータエンタテイメント)発売。 「Logician Lord」(Peto 氏)の WEB サイトで、タスクシステムの解説を更新。
2001	「White Paper(ほわいと・ぺーぱー)」(Moha 氏)の WEB サイトで、タスクシステムの解説を更新。
2003	「Windows プロフェッショナルゲームプログラミング 2」(やねうらお氏: 秀和システム)でタスクシステムの解説を発表。 「最新 DirectX 実践テクニック Part3 Managed DirectX とタスクシステムを使ったゲーム・プログラミング」(C MAGAZINE 2003 年 12 月号掲載 本多直人氏: ソフトバンク クリエイティブ) でタスクシステムの解説を発表。
2004	「ゲームのためのタスクシステム」(C MAGAZINE 2004 年 12 月号掲載 松浦健一郎氏: ソフトバンク クリエイティブ) でタスクシステムの解説を発表。
2005	Xbox360(マイクロソフト)発売。
2006	PLAYSTATION3(ソニー・コンピュータエンタテイメント)発売。 CEDEC 2006「次世代機に向けたゲームエンジンの設計」(石田 智史氏:株式会社カプコン)にて、MT Framework を発表。 「ゲーム・ノ・シクミ シューティングゲーム編 第 11 回 C++によるタスクシステムの実現」(C MAGAZINE 2006 年 1 月号掲載 松浦 健一郎氏/司 ゆき氏: ソフトバンク クリエイティブ)、「シューティングゲーム プログラミング 」(2006 年 9 月発売 松浦 健一郎氏 / 司ゆき氏: ソフトバンククリエイティブ) でタスクシステムの解説を発表。

(7) まとめ

タスクシステムは、C++と STL の登場により、現在では言語およびテンプレートライブラリへ吸収されようとしている。

1980 年代から 1990 年代の間にタスクシステムの技術を持たないゲーム会社やプログラマーでも、現在では自覚することなくタスクシステム的プログラムを実装および利用している。

しかし、タスクシステムの歴史を振り返ることは、ゲーム開発史を振り返るだけでなく、技術の本質を知ることが可能となるため、今度も MT Framework のような優れたタスクシステムへの開発と繋がるきっかけになるであろう。

参考文献：

［1］大野 功二(Country Fox)： “ライブラリで作る簡単ゲーム工房”，工学社 pp.36 - 59，(1999)

［2］やねうらお： “Windows プロフェッショナルゲームプログラミング 2”，秀和システム pp.65 - 115，(2003)

［3］本多 直人： “最新 DirectX 実践テクニック Part3 Managed DirectX とタスクシステムを使ったゲーム・プログラミング”，C MAGAZINE 12 月号 ソフトバンク クリエイティブ pp.36 - 45，(2003)

［4］松浦 健一郎： “ゲームのためのタスクシステム”，C MAGAZINE 12 月号 ソフトバンク クリエイティブ pp.64 - 78，(2004)

［5］松浦 健一郎/司 ゆき： “ゲーム・ノ・シクミ シューティングゲーム編 第 11 回 C++によるタスクシステムの実現”，C MAGAZINE 1 月号 ソフトバンク クリエイティブ pp.135 - 143，(2006)

［6］松浦 健一郎/司 ゆき： “シューティングゲーム プログラミング”， ソフトバンク クリエイティブ pp.48 - 104，(2006)

3.2.10　プログラミング　スクリプト

小久保　啓三
専門学校 HAL 東京

(1)　スクリプティングとは

スクリプトとは、その字義からすれば、演劇や映画といった舞台芸術、映像芸術のもととなる台本のこと、スクリプティングと言えば、その台本書きということに、一般的にはなるだろう。

しかし、ゲーム制作における、スクリプティングが、いかなるものかを明確に定義するのは難しい。スクリプティングと、純粋なプログラミングを分けるポイントが、いくつか存在するというのが一つの理由、そして、その分岐がコンピューターの発達史、ゲームの発展史の中で、それほど明確ではない推移をたどってきたことが、もう一つの理由である。

純粋なプログラミングとスクリプティングを分けるポイントの一つは、それが、ゲームコンテンツをより抽象化して記述するものであるか否か、例えば、アドベンチャーゲームであれば、ストーリー展開を記述するものであるか、といったものである。しかし、コンピューター・ゲームにおいて、一定の一本道をストーリーが進むことは、極めて希である。ユーザーの操作の介入により、ストーリー展開が変化しなければ、ゲームとしての面白みはない。そうなると、ストーリー分岐のための進行制御も、スクリプティングに含まれることになろう。さらに、演出のための視覚的効果、音響的効果も含めていくと、それらの演出制御のどこからどこまでが、プログラミングで、スクリプティングなのかという分かれ目は、非常に不明瞭になる。

もう一つ、スクリプティングを、それ以外のプログラミングと分かつポイントは、スクリプティング、すなわち台本書きを行う言語が、他の部分のプログラム記述言語と異なり、より平易に書けるように設計されている点である。ここでは、このスクリプト記述言語の処理系が、インタプリタ形式かコンパイラ形式かは問わない。実際、これは両方のケースが存在する。ただし、この記述言語の成立を、どこまで遡ればいいのかは、おそらく諸説がありうる。ここでは、思い切って、ゲーム自体の発祥まで巻き戻して、その黎明を見てみることにする。

表 3.2-06　スクリプティングの歴史

1950 年代	1960 年代	1970 年代	1980 年代	1990 年代前半	1990 年代後半	2000 年代前半	2000 年代後半	2010 年以降
EDSAC	System/360	Altair8800	ファミコン	スーパーファミコン	PlayStation	PlayStation2	Nintendo DS	
		APPLE II	IBM-PC	ゲームボーイ	ゲームボーイアドバンス		Xbox 360	
			Macintosh				PlayStation3	
ハードウェア		PDP-11	PC-8801				PSP	
			FM-7					
			X-1					
		アーケード筐体						
OS		UNIX	MS-DOS	Linux	Windows95	WindowsXP	WindowsVista	
			MacOS	MacOS 7			MacOS X	
ゲーム制作に用い、スクリプティング機能を持つソフト				PhotoShop	Maya			
				Microsoft Excel	Flash			
三並べ	アドベンチャーゲーム			ロールプレイングゲーム		大規模 RPG	学習ゲーム	ユーティリティ
	パズルゲーム		アクションゲーム	シミュレーション	レースゲーム		ネットワークゲーム	
主なヒットゲーム		インベーダーゲーム		格闘ゲーム				仮想世界
		シューティングゲーム						
		スクリプティング未分化の時代						
			データ記述発展期					
				スクリプティング黎明期				
					スクリプティング発展期			
ゲーム制作におけるスクリプティングの発展						スクリプティング一般化期		
							スクリプティング標準化期	

(2) システムとコンテンツ未分離の時代

言うまでもなく、コンピューター・ゲームの発生は、フォン・ノイマン型コンピューターの実現に続くことわずか数年、ほとんど同時といっていいタイミングである。汎用計算機は、同時に汎用のおもちゃであることに、人類は容易に気がついてしまったのだ。

初期のコンピューターは、一部の数学者やプログラマーの格好の研究対象であった。表示デバイスも、テキスト文字ベースのモニターやプリンタである。多くは、計算のアルゴリズムを検証したり、一定ルールのパズルを解かせるといった遊びが主体であった。

FORTRAN や、やや遅れての Lisp など、プログラミング言語の確立により、機械語やアセンブラ言語で記述するよりは、より複雑な内容のプログラムが、人間の思考に寄った形で実現できるようになる。

テキストベースの表示デバイスのみを使った遊びとして、計算手順やパズルの解法以外に、ほどなくして、自然言語の文章を使ったアドベンチャーゲームが現れた。

初期のアドベンチャーゲームは、どんなものであったか。

初期 OS(CP/M,Unix,MS-DOS 等)のコマンドラインのような、入力プロンプトに対し、自然言語(英語)に近い単語や命令文を入力することによって、コンピューター内部での世界の状態が変移する。コンピューターがフィードバックとして出力する文章をもとに、プレイヤーは、設定されたゴールを目指すのである。入力時は、シンプルに、選択肢を示すパターンもある。これらのゲームの基本的な枠組みは、高精細、高画質のグラフィカルインターフェースが主流になった現在においても、アドベンチャーゲームというジャンルの総称において、なんら変わっていないことがわかる。

このような初期のアドベンチャーゲームは、前述の通り FORTRAN 言語(あるいは、少し後の Lisp 言語)などによって記述されていた。このような記述形式においては、出力文章などのデータ、状態推移を表すためのデータ構造(配列やリンクドリスト)は、それを操作するアルゴリズムとは分離して存在していた。もしくは、もっと原初的にプログラム中に、出力文章が埋め込まれていた。

プログラミング言語自体がまだ生まれて間もない時代である。ゲームコンテンツを、一つの台本として記述する「スクリプト」という概念は、まだ生まれていない。

ただ、一方で、この時代に、多くのプログラミング言語が新しく作られ、改良され、標準化され、淘汰されていったことは、後に、スクリプト言語をプログラミング言語として捉える場合に、非常に重要な歴史的過程であると言える。例えば、近年、ゲーム業界で、一般化、標準化の兆しを見せる Lua 言語にしても、構造、文法からすると、ALGOL に端を発する PASCAL, C, C++言語族の現時点での末子と見なすことができるからである。現時点、存在する多くのスクリプト言語は、このプログラミング言語自体の発生と進化の

過程を、そのまま螺旋状にたどっているようにも見える。

コンピューター上で、表現したいことを記述することにおいて、大本となったプログラミング言語も、昨今のスクリプト言語も、根底ではいくつかの変わらない要素を持っている。

・ タイプしやすさ(簡潔さ)と、視認性のアンビバレンス。
簡潔な記述を追求すればするほど、表記記号は、コンテキスト依存性が高くなる。例えば、C 言語で用いられる、"&(アンド)"や"*(アスタリスク)"が、文脈によって異なる意味を持つことは、使用者、特に初心者を混乱に陥れる。また、意味と記号を結びつけて習得するのに時間を要することになる。

・ 立体的、複合的な構造を、本来一次元の文字列で表現することの矛盾と工夫。
インデント(字下げ)やカギ括弧、あるいは、従属や含有関係を表す記号":"や"."や"->"などで、複雑な構造の表現が試されているが、現時点では、まだ統一的な表記はなく、人間の直感的、感覚的な認知能力とは離れたものとなっている。反面、安価なテキストエディタさえあれば、記述できるというプラスの面もある。

話を、ゲームコンテンツとプログラムの分離という点に戻すと、この初期の段階で、なんらか、制御プログラムと表示データの分離、あるいは、もっと進んで、物語の制御コードもデータとして分離するようなプログラミングが行われていた。このように、スクリプトのもととなるものは、「データ列」として、プログラム中に混在していた時代が、長く続く。比較的小規模なゲームプログラムであれば、これは今も同じだろう。

(3) パーソナルコンピュータの時代

時代的な節目として、パーソナルコンピュータ(当時の呼称でのマイコン)の普及と、ゲームが商品として成立した時点にも、触れておく必要があろう。
パーソナルコンピュータ(以下パソコン)が、普及することに、大きな影響を与えたもの、それは間違いなく、ビル・ゲイツの 4K-BASIC である。

BASIC 言語は、ダートマス大学で、FORTRAN を下敷きに開発されたものだが、そのインタプリタを、わずか 4K バイトのコードにおさめたことで、プログラミングの楽しさを全世界に広めたことに、ビル・ゲイツの最大の功績がある。アメリカにおいては、Altair, APPLE II といったパソコン。日本においては、NEC、富士通、シャープを始めとするメーカー製のパソコンが、それぞれに、BASIC インタプリタを搭載言語とした。

当時、商品として売られていたゲームのうち、アクション要素の強い計算スピードや描画スピードを問われるものは、機械語による記述が主であったが、アドベンチャーゲームやロールプレイングゲームに属するような多くのゲームが、BASIC 言語で記述され、商品として流通していた。

この時点の BASIC インタプリタが持つ、即時修正の利点、豊富で簡略化された出力のためのコマンド群を合わせ考えると、現時点で、アドベンチャーゲームのための言語シェルが持つのと等しい特徴を備えているといえる。この時代の BASIC 自体を、スクリプティング言語と考えると、「スクリプト」という概念として成立していないものの、その端的な要素は、この時点で現れているとみなすこともできる。

- インタプリタ言語処理系による、即時性。学習の容易さ。
- ユーザーの入力からインタラクティブな反応を返せる実行制御の記述。
- あらかじめ用意され、パッケージ化された演出のための出力手段。

特に、この時代の後期の BASIC 言語が非常に多機能であったことを考えると、非常に極端な見方ではあるが、この多機能 BASIC で書かれたアドベンチャーゲームは、既にそのプログラム記述がスクリプト記述と見なすこともできなくはない。

現代のスクリプトシステムの開発者、巨大化したがゆえ逆に容易にはプログラミングしにくくなったウィンドウシステムを伴うような現代の OS で、簡便なプログラミング手段を提供するスクリプティングシステムを創出する開発者は、この時代を経験していて、その原初体験として、BASIC インタプリタ搭載パソコンの思い出が色濃く出ている場合もある。

(4) 家庭用ゲーム機(ファミコン、スーパーファミコン、ゲームボーイ)の時代

低価格のパソコンの普及が、ゲームの商品化に寄与したとすれば、ファミコンを起源とした家庭用ゲーム機は、ゲームをマスプロダクトに成長させるための第二の起爆剤だったと言えるだろう。ファミコンやスーパーファミコン、あるいはそれ以外の家庭用ゲーム機において、BASIC のインタプリタを実行させるような、メモリや計算スピードの余裕はなく、開発者たちは、自ずと、これらのゲームソフトのアプリケーションプログラムを、おのおののチップのアセンブラ言語で記述していた。

限られたメモリを有効に使うため、プログラムの制御アルゴリズムとゲームコンテンツデータは、整理分離され、圧縮されることになる。

この段階で、よく見られたのは、マクロアセンブラの定義機能を利用して、データ文に

コンテンツデータを埋め込むことであった。ここで埋め込まれたデータは、固定のフォーマットのものから、徐々に、より自由度の高い、一定の取り決めのもとに実行される中間コードのような形態に進化していく。ここにおいて、その中間コードを解釈、実行する処理系(VM=バーチャルマシン)が、存在し始めたことになる。既存のマクロアセンブラを利用して、中間コードを生成し、実行時にそれを解釈するという、ごくごく原始的なスクリプトシステムが、生まれたと言えよう。

実際、ファミコンやスーパーファミコンにおいて、長いストーリー展開を持つロールプレイングゲーム(ドラゴンクエストやファイナルファンタジー)は、当初、このような形態でコンテンツが記述されていた。

その後、中間コードを生成するにあたって、既存のマクロアセンブラを利用するのではなく、独自の変換プログラム(コンパイラ)を用いるように変わってきた。コンテンツ記述の柔軟性と生産性を、より上げるためである。

スクリプトシステムが誕生とともに、成長していった時代である。

[当時のスクリプト言語(というよりデータ記述)の一例]

村人移動データ:

```
座標 X1,座標 Y1,10        ; フレーム
座標 X2,座標 Y2,10        ; フレーム
座標 X3,座標 Y3,10        ; フレーム
0                                          ;END
```

村人メッセージデータ:

```
DATA   ひがしのやまには、まものがでる
0          ; END
```

(5) 大容量ゲーム機(プレイステーション)の時代

ゲーム機に大容量の CD-ROM メディアが使われるようになり、また、メモリや実行スピードにも、ある程度の余裕が生まれるようになり、従来のマクロアセンブラを利用するようなコンテンツ記述はなくなった。RISC チップの構成により、システムのプログラミングそのものも、記述性の高い C 言語に置き換わることになる。もちろん、C のプリプロセッサを駆使して、コンテンツ記述を、C 言語に展開することも可能で、比較的小規模のゲーム・コンテンツでは、そうした手法がとられることもある。大勢においては、各ゲームメーカーが、それぞれのジャンルのゲームの特性に即した独自の言語を開発し、それを中間コードにコンパイルし、実行時にバーチャルマシンでコードを解釈しながら、ゲームを制御するような手法をとっていた。この時代、メーカー内でのスクリプトシステムが、確固とした基盤を築いていく。

[当時のスクリプト言語の一例]

```
[村人 A]
        LOOP
        MOVE        座標 X1 座標 Y1
        WAIT        10              ; フレーム
        MOVE        座標 X2 座標 Y2
        WAIT        10              ; フレーム
        MOVE        座標 X3 座標 Y3
        WAIT        10              ; フレーム
        ENDLOOP
    *talk
        TURN TO 主人公
        MES         「東の山には、魔物が出る」
        RET
```

(6) Windows の時代

上記の時代と前後重なってくるが、Mac や OS/2 といった、ごく一部の人に使われていたウィンドウシステムを持ったパソコンが、Windows95 とともに、かなり広く普及していったことは、ゲームのスクリプティングにおいて、また別の潮流を生むことになる。共通の大きなプラットホームが生まれ、そこでのソフトウェア開発が多くの人に開放された。比較的小規模なゲームであれば、個人や同志のグループで、ゲーム制作が可能となった。簡易に、アドベンチャーゲームが制作できるスクリプトシステムが誕生し始めた。それを下支えしたのは、同人誌文化である。コミックマーケット(通称、コミケ)の会場において、紙媒体と同様、Windows のゲームソフトが売られるようになる。電子紙芝居的に、同人誌をパソコンというメディアに落とし込むには、簡易なスクリプトシステムが非常に好都合であった。

また、パソコンの高速化とともに、各種アプリケーションソフトウェア、ワープロ、表計算、グラフィック編集ソフトなどが、同様に、内部にスクリプトシステムを持つようになり、それを、ユーザーに開放するようになってきた。Microsoft 製品における VisualBasic Script、Flash の ActionScript、3D ソフトの Maya の MEL など、これらのスクリプト言語を駆使して、機能拡張して、ゲームのコンテンツデータを生成することは、現代のゲームメーカーにおいては、ごく一般に行われるようになった。

(7) 高性能ゲーム機の時代

Xbox 360 や PlayStation3 といった高性能ゲーム機の時代を迎え、スクリプティングに新たな潮流が生まれつつある。

ひとつは、スクリプトシステムのインタプリタ化である。高性能ゲーム機では、大量の画像データや 3D データを扱うために、非常に高速な CPU が備わり、またそれらのビジュアルデータを扱うために以前に比べれば大量のメモリも搭載している。テキストデータをメモリにおいて、即時解釈実行するコストを、相対的に、ある程度無視することも可能になってきたのである。ゲームの制作において、もっとも時間がとられる、細かい調整を、プログラムの再構成なしに、継続的に行うことのできるメリットのほうが大きくなってきた。先に述べたような大量のビジュアルデータを含んでいたり、また複雑、大規模になったプログラムを再構成するのは、反対にものすごく時間がかかる。このように、開発の試行錯誤におけるターンアラウンドタイムを短縮する目的もあって、スクリプトシステムの独立性は、非常に高くなってきた。

もう一つの潮流は、スクリプトシステム、あるいは、スクリプト言語の統一、標準化への指向性である。最近、注目されているのは、Lua や Squirrel といった言語と、処理系で、これらは、C 言語のホストプログラムへの組み込みを前提としているため、比較的容易にゲームシステムに導入することができる。

標準化の恩恵としては、プログラマーが、新たにスクリプトシステムを書き起こすことなしに、比較的信頼性の高いシステムを導入することができる点が一つ、また、もう一つには、ゲーム・コンテンツを記述する企画者、演出者にとって、言語が共通化することで、学習の負担が減り、職場やチームが変わっても応用が利く点が考えられる。

ただ、一方で、Lua や Squirrel は、その記述性の高さゆえ、C 言語や C++言語並みのプログラミングセンスを要求する面もあり、どちらかというと、プログラマー、ないしは準プログラマーが、ゲーム内のオブジェクトの挙動や難易度を調整するのに向いていると言える。

(8) これからのスクリプティング

今後のゲームにおけるスクリプティングを予見するにあたって、いくつかのキーワードを挙げる。

- 標準化
- オブジェクト指向
- 並列化(プロシージャル)
- デバッグ環境

・ UI シェル
・ ミドルウェア

スクリプト言語が、同時にプログラミング言語である以上、今までのプログラミング言語が辿った歴史を螺旋状になぞることは、疑い得ない。大きな標準化と小さな分派が、今後大きな変化としてあることが予測できる。オブジェクト指向の考え方は、21 世紀に入って、徐々に多くのプログラマーの受け入れるところとなり、それは時間をかけて、企画者、演出者にも浸透していくことになるだろう。ゲーム内のオブジェクトは、個々に並列に独立して、ときに干渉しながら挙動を持たなければならないが、Lua などの言語内には、仕様の中にその枠組みはなく、個々のオブジェクトの挙動をプロシージャルに記述しつつ、それらの干渉、相互通信を容易に記述できるフレームワークが必要になるだろう。

調整や試行錯誤をやりやすくするため、また、バグを未然に防いだり、とりやすくするために、デバッグ環境の整備は必須となるだろう。開発末期に行くほど、バグによる遅滞でかかるコストが膨れあがることを考えると、これはメーカー自体の存亡に関わる重要なポイントである。

スクリプト言語というと、結果、一定書式に従った一次元のテキストデータであるが、この中で、数値指定するよりは、視覚的なツールで、パラメーターや範囲を指定する方が、はるかに効率的な場合がある。このようなグラフィカルなツール、あるいは UI シェルといったものと、どう連携させるかが、システムを構築する上での課題となるだろう。これらはまた、スクリプトの言語学習が時間的に困難な演出者にとっても、助けとなるはずだ。

上記のものも含めて、いくつかの標準化された知恵などが、ミドルウェアとしてオープンになったり、あるいは商用として売られたりする可能性もあるだろう。

以上が、現時点でのスクリプティングシステムの動向に対する予測である。

私感としては、例えば、Ruby 言語のように、日本発で、世界的に認められる言語もある中、スクリプティング環境にまつわることで、今後何であれ、国内から、新たな潮流やツールが生まれ、発展し、世界に認められていくことを願ってやまない。

3.2.11　ネットワーク通信

フリーライター
佐藤カフジ

(1)　ネットワーク通信とゲームの歴史

(a)　パソコン通信からインターネット時代の幕開け

①　黎明期

ネットワークゲームの技術は、その時代に利用できる通信インフラに合わせて発展してきた。このため、今日に至るネットワークゲームの技術を理解するためには、まず、通信インフラの変遷を正しく捉えることが必要である。ネットワークゲーム関連技術とインフラの変化について、図 3.2-94 に示す。

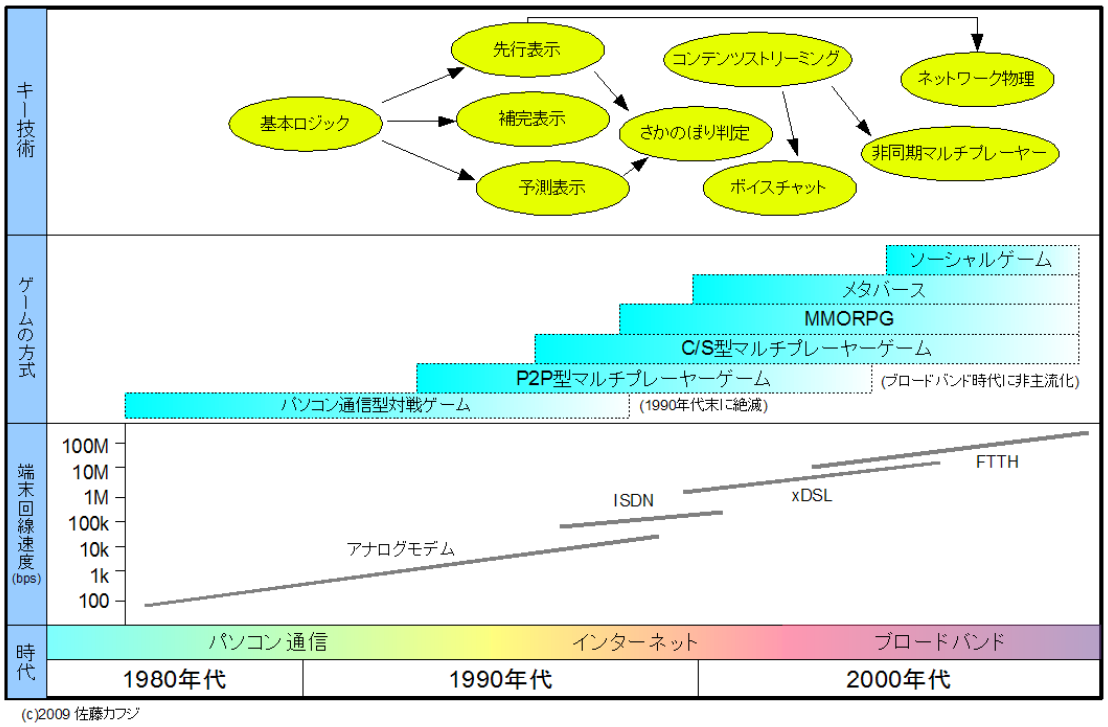


図 3.2-94　ネットワーク関連技術とインフラのロードマップ

現在広域ネットワークの方式として主流となっているインターネットは、遠隔地のコンピューター同士が 24 時間常時接続されているネットワークの一形態である。その前身は、1969 年に米国の 3 つの大学が共同して設立した ARPANET に求めることができる。

ARPANET は当初クローズドなネットワークであった。また 1970 年代から 1980 年代に掛けて、ARPANET に参加しない大学や研究所が別のクローズドネットワークを多数

成立させた。これを受け、ARPANET は 1984 年にアメリカ科学財団のネットワークである NSFNET と相互接続し、これをベースとして、1990 年までには米国内の多くのネットワークが相互接続されることになった。そのころから、このネットワークは、「ネットワーク間のネットワーク」を意味する“The Internet”と呼ばれるようになった。

また日本では、1986 年に日本電気株式会社による PC-VAN、同年に日商岩井と富士通による NIFTY-Serve といったパソコン通信サービスの運営が開始されたが、それらがインターネットに接続されるのはおよそ 10 年後の 1995 年頃の事である。それまでは、日本国内の通信サービスは、特定のサーバーに電話回線を通じて直接接続するクローズドネットワーク方式が主流であり、初期のオンラインゲームのサービスもその中で行なわれていた。

② 最初期のネットワークゲーム

最初期の民生用オンラインゲームとしては、米国 Kesmai 社の「Air Warrior」(1986 年)、「Megawars」(1980 年代末)などが挙げられる。またゲームとは言い難いが、富士通のアバターチャットシステム「富士通 Habitat」(1990 年)が最初期の MMO 型エンターテイメントシステムのひとつである。

これらのゲームは、最初期のサーバー・クライアント型ネットワークゲームとして、GEnie、CompuServe、NIFTY-Serve といったパソコン通信上で運営が行なわれていた。収益方法としては分毎の従量課金制を取っており、2009 年現在の水準に比べて、利用料金は極めて高価であった。

また、1980 年代後期から 1990 年代初頭にあたるこの時期、一般ユーザーが利用できる通信環境は、2400bps 程度のアナログモデムであった。1 秒間に 200 バイト程度という極めて限られた通信帯域で、最初期のネットワークゲームが成立していた点に注意したい。なぜならば、これは、ネットワークゲームを制作する上で避けて通れない、通信帯域の削減というテーマにおいて、最も厳しい条件をクリアしていることを意味するからである。

③ アナログモデムを前提としたゲームとサービス

1990 年代中頃になると、パソコン通信サービスの利用者が増えたこともあって、当時の民生用通信インターフェースとして主流であったアナログモデムを前提としたゲームやサービスが登場するようになった。

その皮切りが、1993 年 12 月に北米で発売された PC/AT 用「DOOM」(id software)で

ある。最初の成功した FPS タイトルである「DOOM」には 2 つのネットワーク対戦モードがあり、ひとつはローカルネットワーク(IPX プロトコル)を経由するローカル対戦、もうひとつがアナログモデムを使い、電話回線を通じて対戦相手のモデムに直接ダイヤルアップしておこなう遠隔対戦である。

「DOOM」における遠隔地間の対戦は、構造的に 1 対 1 の直接通信となるため正しくは“ネットワーク”ではないものの、現在のネットワークゲームに通じる通信技術の萌芽を見ることができる。当時利用できた民生用の通信インターフェースは 9600bps から 28.8kbps 程度のアナログモデムで、「DOOM」はその限られた通信帯域を用いて、秒間 30 回の情報更新を伴うリアルタイムのアクションゲーム体験を、同時に 2 人のプレイヤーに提供することができた。また、IPX プロトコルのネットワークを介して、最大 4 人でマルチプレイゲームを行なうことも可能であった。これを実現した実装方式については後述する。

「DOOM」のモデム対戦が流行した同時期の 1994 年には、遠隔地のプレイヤー同士が複数人で対戦を行なうためのネットワークサービスがスタートしている。ひとつは米国 IVS 社が米国テキサス州ヒューストンで開始したゲーム専用のダイヤルアップネットワーク「DWANGO(Dialup Wide-Area Network Game Organization)」である。

このサービスはダイヤルアップネットワーク上で IPX プロトコルをエミュレートし、ローカルネットワーク対戦に対応したゲームを複数人でプレイ可能にするクローズドネットワークのゲームロビーシステムである。「DOOM」や「WarCraft 2」(Blizzard)、「Duke Nukem 3D」(3DRealms)といったタイトルで最大 4 人から 8 人の同時対戦が可能であった。但し多人数対戦時には 28.8k～56kbps の高速アナログモデムを参加者全員が使用する必要があった。また、同サービスは 1996 年にソフトウェアジャパン株式会社により日本国内に展開され、1997 年末、株式会社ドワンゴに全権が移管された。

また同じ 1994 年に、米 Catapult 社が SNES(スーパーファミコン)及び SEGA GENESIS(メガドライブ)向けのダイヤルアップ通信対戦サービス「XBAND」をスタートさせた。これは、ゲーム機にモデム機能を持つハードウェアを接続しておこなう点を除けば、「DWANGO」によく似た形式である。コンシューマー向けとしては敷居が高くマイナーな存在であったが、1996 年には日商岩井による国内サービスが開始されている。

④　インターネットの時代へ

「DWANGO」や「XBAND」といったパソコン通信型のネットワークゲームサービスは長く続くことができなかった。なぜならば、1990 年代中期～後期にかけて、インター

ネットの利用価値が高まり、ダイヤルアップ方式のクローズドネットワークの存在に必然性が無くなったからである。「DWANGO」は 1998 年頃にインターネット化(『IWANGO』)を果たしたがサービスは長続きせず、「XBAND」は 1997 年に Catapult 社が事業から撤退、セガが国内サービスを引き継ぎ、1999 年にサービスを終了している。

インターネットの急速な発展は、1992 年に WWW(ワールド・ワイド・ウェブ)が発明され、1993 年に最初のハイパーテキストドキュメントブラウザが開発されたことに端を発する。1995 年にはマイクロソフトウィンドズ OS に Internet Explorer が標準搭載され、インターネットが事実上標準の広域ネットワークとして認知されるようになった。

コンピューター・ゲームも、この変化に対応している。最初の本格的なインターネット上の対戦ゲームは、1996 年 6 月にリリースされた FPS タイトル「Quake」(id software)であった。「Quake」ではインターネットを通じて最大同時 16 人で対戦が可能であった。また、1997 年には、マイクロソフト Direct X 5 の DirectInput 機能を用いてインターネット対戦に対応した初の RTS タイトルである「Age of Empire」(Ensamble Studios/Microsoft Game Studios)や、オンライン MORPG の原型「DIABLO」(Blizzard Entertainment)、また、初の本格的 MMORPG である「Ultima Online」(Origin/Electronic Arts)が発売されている。

1996 年から 1998 年頃にあたる時期、一般ユーザー端末の通信環境は 28.8kbps から 56kbps のアナログモデム、もしくは 64kbps から 128kbps の ISDN モデムと、パソコン通信時代から帯域幅が大きく変わったわけではなかったが、インターネットというオープンネットワークを通じて全世界の人々と対戦できることは、ネットワークゲームのあり方に大きな変化を与えた。

(b) インターネット時代のゲーム技術要素

① “距離”の問題

1996 年～1997 年頃にリリースされたインターネット対戦対応ゲームは、それ以前のクローズドネットワークを前提とするネットワークゲームには存在し得なかった技術的問題に直面した。すなわち、ゲームに参加する端末間の“距離”である。

クローズドネットワークでの通信では、端末から端末へ通信データが運ばれる所用時間(レーテンシー)がほぼ一定であると仮定することができた。しかし、インターネットは全世界を繋ぐ複雑なネットワークであり、通信データは複数のホストを経由してやりとりされるため、レーテンシーは不定である。一般に、物理的距離が遠くなるほど、レーテン

シーは増える。経由するホストの数、性能にも影響されるが、無限速のルーティングシステムを使ったとしても、光の速度は有限であるため、遠隔地の通信には必ずこの問題がつきまとうことになる。

このため、最初期のインターネット対戦対応ゲームは、数十ミリ秒から数百ミリ秒程度の変動するレーテンシーを前提とするネットワークコードデザインに取り組む最初の機会となった。特に初期バージョンの「Quake」はローカルネットワークの低レーテンシー環境を前提とした、完全同期型のネットワークコードを持っていたため、インターネット経由のプレイで発生するラグの大きさを直に体験することとなり、問題は明らかであった。

この問題を解決するため、id software では「QuakeWorld」という、「Quake」の派生バージョンを開発した。これはインターネット上のレーテンシーを、ゲームプレイ上は目立たなくするための技術を組み込んだもので、実際は数百ミリ秒のレーテンシーが発生しているような端末上でも、見かけ上はキャラクターを快適に操作することが可能であった。

その中核となる技術は、“プレイヤー・キャラクターの先行表示”と、“ネットワークオブジェクトの補完表示”と言うべき内容となっている。いずれも、「Ultima Online」でも同種のアイデアが実装されていたほか、2009 年現在のネットワークゲームでも、まずまちがいなく実装されることになる基本技術である。これらについて、詳しくは第二章で触れる。

② マッチングロビーの標準化

インターネットを前提とするインゲームの通信ロジックが完成に近づく一方、プレイヤー同士の出会いの場を提供する、マッチングロビーのデザインにも変化が見られた。そのひとつは、1997 年末に発売された MORPG「DIABLO」である。このタイトルでは、オンラインでの継続的なプレイを前提とした“Battle.net”と呼ばれるマッチングロビーシステムをユーザーに提供することで、“距離”に影響されることのない、対戦機会の提供を試みている。

この考え方は、後の少人数参加型ネットワークゲームの基本的なデザインに大きな影響を与えているほか、ネットワーク型ゲーム端末で標準的な考え方となるオンラインコミュニティの支援システの基本的なアイデアを提供した。すなわち、オンラインのコミュニティと、そのゲーム体験の場を 1 カ所に集約することで、ゲームから得られる楽しさの質を向上させるという方向性である。

また「DIABLO」に継ぐ 1998 年、インターネット対応版の「DWANGO(IWANGO)」や、「Quake」用のサーバー検索アプリケーションとしてスタートした「GameSpy」、マイクロソフトが展開したインターネットのネットワークゲームロビー「Microsoft Gaming Zone」といった、ゲームから独立した形のロビーイングシステムも多数現れた。これらの独立したマッチングロビーサービスは、後に「Xbox LIVE」といったオンライン・コミュニティシステムに発展していった。

③　低帯域幅に対応するネットワークゲーム技術

ゲームコミュニティのインターネット化が PC プラットホーム上で静かに進行していったことによる影響は、家庭用ゲーム機にも及ぶ。その形が最初に現れたのが、1998 年 11 月 27 日に発売されたゲーム機、ドリームキャスト(DC)である。DC には最高 33.6kbps のアナログモデムが標準装備されており、インターネットにアクセスすることが可能な最初の家庭用ゲーム機となった。

国内のネットワークゲームは、この DC を対象として大きなステップを踏み出している。DC における最初期のネットワーク対応ゲームのひとつ、「セガラリー2 チャンピオンシップ」(セガ AM3 研/ドワンゴ)は、アナログモデムを経由した高レーテンシー、低帯域幅の環境でレースゲームの多人数対戦を可能にした、最も初期の国産タイトルとなった。

「セガラリー2」で特に 重視されたのが、リアルタイム制の高いラリーゲームを、アナログモデムの低帯域幅(信頼性の高い通信速度は 28.8kbps まで)にて、4 人以上のマルチプレイを可能にすることであった。このため、本タイトルでは、後のレースゲームでも必須となる“ネットワークオブジェクトの航法予測”、“補完表示”、“結果同期”といったネットワークゲームロジックの高度な実装を試み、最小限のデータ交換だけを行いながら、リアルタイム制と動きのスムーズさを確保し、そして接触などによる影響をそれらしく見せることに成功している。

④　大規模参加型オンラインゲーム

1997 年 9 月に正式サービスをスタートした初の本格的大規模参加型(MMO)RPG「Ultima Online」は、サーバー・クライアント型のネットワークトポロジーを、ゲーミングの世界に本格的に導入した最初の作品でもある。それ以前にも 1996 年の「Meridian 59」(3DO company)等、“Mult User Dungeon”と呼ばれた初期のオンライン RPG は存在したが、「Ultima Onine」が本質的に異なるのは、その規模である。

初期の「Ultima Online」では、北米各地に十数のサーバー施設を設置して、それぞれの施設を“シャード”と呼んだ。各シャードは並列世界であり、それぞれ最大 3,000 人ほど

のユーザーが同時ログイン可能であった。それ以前の“Multi User Dungeon”に比べると、100 倍以上の規模である。

これを可能にするために、「Ultima Online」では、基幹業務系のネットワークサーバー技術を応用した、負荷分散型のサーバーシステムを構築した。各サーバーアセット、すなわちシャードは、単一のサーバーマシンではなく、ゲーム世界を面積的に分割して数十台のサーバーマシンにより分散処理される。それらのマシンは、プレイヤーデータを保持する DB などのバックエンドサーバーと、ユーザーの接続を振り分けるフロントエンドサーバーに挟まれる形で接続され、その全体があたかもひとつのゲームサーバーであるかのように振る舞う。

この方式は、後の MMO 型ゲームに当然のごとく受け継がれ、具体的な構成はそれぞれ異なるものの、現在に至るまで多くのオンラインゲームタイトルで同様の手法が採られている。

(c) ブロードバンド時代のネットワークゲーム

① 家庭用ゲーム機のネットワーク化

2000 年代に入ると、一般家庭で DSL ルーターが利用できるようになり、ユーザー端末が Mbps オーダーの広帯域幅を持つようになった。ブロードバンド化が進むインターネット上では、音声や動画などのマルチメディア情報も実用的にやりとりされるようになり、爆発的に利用者を増やすことになった。

このことは、家庭用ゲーム機をオンライン化に向かわせる重要な転機であった。つまり、アナログモデムの時代とは異なり、一般家庭に 100Base-T 程度のローカルネットワーク端子が存在し、さらに、インターネットへの常時接続が標準となったのである。

これを受けて、ソニーのプレイステーション 2（PS2）では、2001 年 9 月に専用 HDD とネットワークアダプタ(100Base-T)の拡張セットを販売開始し、2002 年 5 月 16 日には PS2 初の MMORPG「ファイナルファンタジーIX」(スクウェア)といったオンラインゲームをプレイすることが可能になった。また同時に、オンラインアカウントの管理、支払いなどを処理するオンラインサービス「PlayOnline」がスタートしている。

また、マイクロソフトは 2001 年 11 月 15 日に家庭用ゲーム機 XBOX を発売した。このゲーム機は 100Base-T のネットワークソケットを標準搭載しており、家庭にブロードバンド環境があることを前提として、同機種用のクローズドネットワーク「XBOX

LIVE」上にてゲームのアップデートや追加コンテンツの配信といったサービスを積極的に展開した。

この流れは留まることなく発展する。2003 年頃にはアーケードゲームのネットワーク化が本格化したほか、2004 年 11 月には初の Wifi 通信機能搭載携帯ゲーム機となるニンテンドーDS が任天堂から発売、同年 12 月に同じく Wifi 通信機能を搭載した携帯ゲーム機であるプレイステーションポータブル(PSP)がソニーから発売された。

また、2005 年 11 月に発売されたマイクロソフトの後継機種 Xbox 360 は、「XBOX LIVE」サービスとの連動性をより強化し、インターネットへの接続を前提とするグランドデザインを施している。2006 年 11 月にはソニーのプレイステーション 3 が発売されると同時に、同機種用のオンラインサービス「PlayStation Network」がスタートした。これはオンラインマッチング支援、オンラインアップデート、コンテンツダウンロード、ユーザー間のメッセージ送受信といった包括的な機能を持つクローズドネットワークであり、「XBOX LIVE」とほぼ同等の存在である。また、これらの機種用に発売されるパッケージゲームは、そのほとんどがオンライン対戦対応、もしくはオンライン対戦専用となっている。

② コンテンツストリーミングとメタバース

ブロードバンド化のもたらした大きな変化として、ユーザー端末が大量のデータを送受信することが可能になったことが挙げられる。その恩恵を最大限に活用した最初のオンラインエンターテイメントタイトルが、2003 年 4 月にスタートした「Second Life」(Linden Lab)である。

「Second Life」の最大の特徴は、Linden Lab が運営するひとつの広大な世界に、参加ユーザーが自由に創作活動を行えることであった。ユーザーはゲーム中にビルトインされたコンテンツエディタで 3D 形状を制作し、アップロードしたテクスチャを張り、制御スクリプトを書いて、様々なものをゲーム世界に送り込むことができた。

このような世界では、従来のオンラインゲームとは異なり、ゲーム世界の構成物を、あらかじめローカルディスクに保存しておくことができない。オンラインの世界がどのような変化をするか誰にもわからないためである。そこで、「Second Life」では、ゲーム世界の構成物を、接続ユーザーに対してリアルタイムに圧縮・送信するという、リアルタイムコンテンツストリーミングの手法を採っている。そのデータ量は膨大で、ユーザーひとりあたりに対しての最大通信速度は 5Mbps にも登る。

このように、ブロードバンドを前提として構築された世界は、それ自信がゲームではない代わりに、現実世界のようにダイナミックに変化していくこと、また、参加ユーザーに現実同様の遵法精神やエチケットが求められることが特徴である。そのため、このようなジャンルは“バーチャル世界”、あるいは“メタバース”と呼ばれるようになった。

③　ユビキタス化とソーシャルゲーム

ブロードバンドネットワークの世界的な普及と、Wifi 端末やスマートフォンの発達は、2007 年 6 月に発売された Apple の「iPhone」および 2008 年 6 月に発売された「iPhone 3G」をきっかけとして、ソーシャルゲームという新たなネットワークゲームの可能性を切り開いた。

ソーシャルゲームは、いまだ厳密な定義が成されていない言葉であるが、基本的な特徴としては、数百万から数千万人規模の巨大なオンラインコミュニティの存在を前提とし、ユーザーの位置や個人プロフィール情報といった“ソーシャルな”情報を活用して展開するネットワークゲームであると言える。

このカテゴリーにおける先駆的な作品のひとつは、2008 年 9 月にリリースされた iPhone アプリケーション「Sonic Lighter」(Smule)である。内容は至極単純で、iPhone の画面上でライターに火を付けるだけのものである。ただし、火をつけたという情報は、位置情報とともにサーバーに集められ、世界中の色々な場所でどれくらいの火が灯されているかを地図画面で見ることができる。これはユーザー間の国レベルの競争心を煽り、沢山のユーザーがこぞって「Sonic Lighter」のネットワークに参加した。

注意すべきは、アプリケーションそのものは従来の概念ではゲームと呼べない内容でありながら、巨大なコミュニティの心理を上手く刺激することによって、ゲーム的なエンターテイメント性を提供しえたことである。

技術的には、ライターに火を灯したという情報を 1 つのサーバーに集める、というだけのごく単純なものである。だが、ネットワークの巨大さそのものによって、従来ではあり得なかった効果を上げている。このことは、次の時代のネットワークゲームを定義するための重要なきっかけになり得る。

(2) ネットワークゲームの技術分類

(a) ネットワークトポロジー

① 基本的なトポロジーの分類

ネットワークゲーム技術は、それぞれのゲームタイトルがユーザーに提供するゲームプレイ体験を、ネットワーク上で実現するための各種の方法である。その最も基本的な部分に、ゲームの仕様に合わせたネットワークトポロジーの選択がある。

ネットワークトポロジーとは、ゲーム世界を構成する端末の接続形態のことで、現在までのゲームに利用されている形態は、ピア・ツー・ピア、サーバー・クライアント、オーバーレイ・ピア・ツー・ピア・ネットワークに大別できる。

② ピア・ツー・ピア　ネットワーク

初期のネットワークゲームには、ピア・ツー・ピア型のネットワークトポロジーを採用するものが多くみられた。ピア・ツー・ピアは、ゲーム世界を構成する各端末が、およそ対等の立場で相互に接続する形態であり、各端末は同じゲームに参加する全ての他の端末への接続を確立する。この方式は、「DOOM]や、「Age of Empire」、「Diablo」といった初期の小規模参加型タイトルで、その技法が活用された。

ピア・ツー・ピア型のゲームネットワークで、各端末間でやりとりされる情報は、各端末で行なわれたプレイヤーの入力そのもの、もしくはそれを変形したものである。例えば、「DOOM」では、プレイヤーが押したキーの情報が“移動入力”、“回転入力”、“射撃などのコマンド入力”といった形でパッケージングされ、同じゲームセッションに存在する全端末に送信されて、ゲームが駆動する。

通信の高速性さえ担保できれば、完全に同期した正確なプレイ体験をユーザーに提供できることが、この方式の最大のメリットである。したがって、現在も、対戦格闘のような一部ジャンルのゲームではこの方式を採ることが有利である。

この方式における問題点もある。ネットワーク内の接続数および通信量が、参加する端末数の二乗に比例して増加するため、あまり大きな人数のマルチプレイヤーを実現できない。アクション系ゲームでは 2 名から 4 名まで、ややリアルタイム性の下がる戦略系のゲームでも最大 8 名程度が限界となる。この制限は、ブロードバンド時代となった現代でも全く変わることはない。この制限のため、この手法は近年になり非主流化している。

③ サーバー・クライアント　ネットワーク

サーバー・クライアント型のゲームネットワークは、ゲームに参加する各端末が、単一のサーバーマシンに対してのみ、接続を行なう形式である。端末間の情報のやりとりは、常にサーバーマシンを通して行なわれる。このため、ネットワーク全体の接続数は、参加する端末の数に比例するほか、その負荷はサーバーマシンのみに掛けられる。このため、ピア・ツー・ピア型ネットワークに比べ、参加人数の大規模化が容易である。

「Quake」以降の FPS 系ゲームでは、ほぼすべてのタイトルがこの方式を使ったネットワーク実装を行なっているほか、「Ultima Online」に端を発する MMORPG ジャンルでは、すべてのタイトルがこの方式を採っているといえる。

この方式においては、サーバーマシンと各端末の役割は決定的に異なっている。ゲームを本質的な意味で駆動するのはサーバーマシンのみであり、そこに、ゲーム世界の全ての情報が集約されている。各端末は、サーバーマシンからゲーム世界の情報を受け取り、プレイヤーへの画面表示を構成する。プレイヤーから端末が受け取った入力情報は、サーバーマシンに伝えられ、サーバーマシン上でキャラクターの駆動が行なわれ、その結果が端末にフィードバックされる。

大規模なオンラインゲームでは標準的なネットワークトポロジーと言えるが、本質的な問題点として、サーバー上のゲーム進行と、各端末におけるゲーム進行は、ネットワークのレーテンシーに影響されており、厳密には各端末間でゲームが非同期に進行する。このため、ある端末で起きたゲーム内の現象が、他の端末上では、別のタイミングで起きたように見えることがある。

④　オーバーレイ P2P ネットワーク

これはピア・ツー・ピア型ネットワークトポロジーの一種であるが、2-1-2 の節で説明したピア・ツー・ピア型ネットワークと異なり、各端末が他の端末に張る接続の数が一定数に制限されており、ある端末は、ネットワークを構成する一部の端末グループに対してしか、直接の通信を行なわない。その代わりに、各端末は情報のリレーイングを行ない、ある宛先を持つ情報は、複数の端末を経由して目的地に到達する。これは、インターネット上に別の論理ネットワークを構築するもので、オーバーレイネットワークと呼ばれる。特に 2002 年頃から注目されはじめた技術であり、ファイル共有アプリケーション「Winny」や、動画配信ネットワークシステム「Peercast」などで実現例を見ることができる。

この方式をネットワークゲームで本格利用した商用ゲームは今のところ確認されていないが、その有用性に関して、Microsoft Reserch が実験を行い、報告している

(Gamefest Japan 2007：参考
http://game.watch.impress.co.jp/docs/20070910/gf_live.htm)。その報告では、クライアント・サーバー型のネットワークトポロジーを持つ「Quake III: Arena」(id software)を改造して、オーバーレイ P2P ネットワークを構成した。

各端末は、ゲームフィールド上の物理的な位置が近しいプレイヤーと動的な接続を行なう。ゲーム内の情報は、プレイヤーからの距離や関心度に応じてランク付けされ、重要なデータは頻繁に、重要でない情報はときどきやりとりされる。これによる、画面上のキャラクターの移動情報の欠損を補うために、高度な航法予測 AI 技術の実装が不可欠であったが、数千端末規模の大規模な P2P ネットワークにて「Quake III」の現実的な対戦を行なうことが可能であった。Microsoft Research では、この技法を用いることで、FPS のようなアクション性の高いゲームを、特定のサーバーに負荷をかけることなく、事実上どこまでも規模を拡大できるとしている。

この種のピア・ツー・ピア型ネットワークの問題点は、情報が各端末をリレーして伝わっていくために、情報が伝達するまでに大きなレーテンシーが発生することである。このため、レーテンシーを隠蔽するための、様々なテクニックの実装が不可欠である。また、ネットワークを構成する端末の通信性能により、ネットワーク全体のクオリティが変化するため、一定のクオリティを必ず提供したいと考える商用タイトルでは、採用が難しいのも事実である。

(b) レーテンシーの影響を隠蔽するゲームロジック

① 先行表示と「巻き戻してやり直し」技法

ゲームのネットワークトポロジーに寄らず、インターネットを経由してマルチプレイを行なうネットワークゲームでは、数十ミリ秒から数百ミリ秒までのネットワークレーテンシーを前提としたゲームロジックの構築が必要である。その中で、端末上のプレイヤーが直接操作するローカルプレイヤーキャラクタを、あたかも全くレーテンシーが存在しないかのように見せる技術がプレイヤーキャラクタの先行表示である。

後述する予測表示の技法とは異なり、端末上のローカルプレイヤーキャラクタは、その制御に対して、その端末を操作しているプレイヤーの入力を直接利用できる。このため、シングルプレイゲームと同様のレスポンスで、キャラクターを動作させることができる。

しかしながら、その表示はサーバー上で進行しているゲーム世界や、サーバーの向こう側にいる他の端末とは時間的なズレを持っている。特に問題となるのが、サーバー上で

プレイヤー・キャラクターに対する強制的なアクションが発生(爆風で吹き飛ばされた等)したときに、それをどうやって端末上のキャラクターにフィードバックするかである。でなければ、サーバー上のキャラクター位置と端末上のキャラクター位置が無制限にずれていく。他の端末上のプレイヤーキャラクタとのインタラクションも、これと同様に説明できる。

初期のインターネット対戦ゲームである「Quake World」では、この問題に対する完璧な解法の一種を実装した。ゲーム全体はサーバー上の単一の時間軸に沿って動く。それをサーバーは、ゲームフレーム番号として、フレーム毎にインクリメントしている。端末は、サーバーから得たゲーム世界の情報に、それがいつの情報であるかということを、フレーム番号の形で知らされる。したがって端末は、自分が先行動作させているローカルキャラクタが、「周囲の情報から何フレーム分先行しているのか」ということを常に把握できる。

それを前提として、あるフレームにおいて、サーバー上でキャラクターに対するアクションが発生する。それは「n フレームにおいて、爆風で x 方向に y の力で押し出された」という形をとり、端末に知らされる。端末上のローカルキャラクタにとって、それは「過去のできごとである」ことがわかるため、端末は内部的にローカルキャラクタを n フレーム時点まで「巻き戻し」、サーバーから知らされたイベントをキャラクターに適用する。同時に、n フレームから現在までに、端末上のプレイヤーが既に入力した情報を使い、現時点までのキャラクターの動きを「やりなおす」。そこまで行なって、画面表示に移る。

この「巻き戻してやりなおす」方法では、端末上でレーテンシーの存在しない快適な入力と画面表示をプレイヤーに提供しつつ、サーバー上で起きた事件を正確に適用できるため、ゲーム進行の不整合が発生しない。またこれは、数百ミリ秒の大きなレーテンシーがある環境でも問題なく動作する。現在のほとんどのネットワークゲームは、この方式そのままか、類似する、あるいは変形した方法でローカルキャラクタの先行表示を実現している。

② さかのぼり判定

レーテンシーが存在するネットワーク環境では、プレイヤーキャラクタ間のインタラクションに時間的なズレが生じる。シューティングゲームでは、プレイヤーが敵に向かって射撃を行なうが、その判定はサーバー上で行なわれるためレーテンシーが介在し、端末上で命中したように見えても、サーバー上では命中しない、ということが起こりうる。そこで、プレイヤーは、レーテンシーで発生する画面上の敵キャラクターの位置のずれをあ

らかじめ予測し、見越し射撃をする必要があった。これはプレイ上大きなストレスになる。

この問題を解決するため、2001 年、Valve 社は「Counter-Strike」(Valve)バージョン 1.6 に合わせ、ネットワークコードの改良を行なった。前節で説明したように、サーバーは、現在のゲーム時間を知っており、クライアントもまた、射撃動作を行なった瞬間のゲーム時間を知っている。これを利用して、サーバーは、端末から送られてきた射撃動作の処理を行なうときに、「その端末が射撃の瞬間に見ていた風景」の時間までサーバー上のゲーム世界の時間を巻き戻し、その地点において命中判定を行なうのである。これを実行するためにサーバーは、ゲーム世界の過去の情報を、数秒分保持する仕組みになっている。

これにより、端末上のローカルキャラクタの先行表示と、レーテンシー分遅れて表示されるネットワークオブジェクト間のインタラクションが、端末上で見た目通りの結果をもたらすようになった。但し、過去の位置に対して命中判定が行なわれるため、他の端末上では、「避けたのに当たってしまった」という、不整合が生じる。だが、回避のアクションよりも、攻撃のアクションの方が、ゲーム性における関心度が高いため、「Counter-Strike」ではこの方式がうまく適合した。

③　ネットワーク物理とオブジェクト制御権の移動

レーテンシーの時間分だけ遅れて表示されるネットワークオブジェクトと、端末上で先行表示されているプレイヤーキャラクタ間で発生するインタラクションは、その影響が遅れて伝わるために、表示上の不整合を生じてプレイヤーに不快感を与えることがある。例えば、物理オブジェクト同士が衝突すると、双方のオブジェクトに即座に反作用が生じなければならないが、レーテンシーの影響により相互作用が遅れ、オブジェクトへのめり込みといった見た目上致命的な状況が起こる。したがって、ネットワーク物理では特別な技術が必要である。

この現象を小規模なマルチプレイヤーゲームにおいて改善する技法を、Pandemic Studios が 2008 年「Mercenaries 2」にて実装し、報告している (GDC2008:“Mercenaries 2: Networked Physics In a Large Streaming World”)。「Mercenaries 2」では、大量の物理オブジェクトが運動するゲーム世界をネットワーク化している。このような環境では、物理オブジェクトとプレイヤー・キャラクター間のインタラクションが頻繁に発生するため、レーテンシーの影響をそのまま見せていたのでは、常に表示が破綻したままとなってしまう。

そこで、「端末上で処理するローカルオブジェクトは、レーテンシーの影響なくインタラクションできる」という特性を最大限に活用する。つまり、ネットワークオブジェクト

として処理される物理オブジェクトが、あるプレイヤーとのインタラクション圏内に入ると、そのプレイヤーの端末上のローカルオブジェクトとして処理させる。つまり、ネットワークからローカルへ、オブジェクトの制御権が移動する。

あとは、対象のオブジェクトに対しては端末上で物理制御をおこない、ローカルプレイヤーとのインタラクションをソロゲームと同じように処理するだけである。これにより、大量の物理オブジェクトとプレイヤー間のインタラクションを、表示上の不整合を全く発生させることなく処理することができる。問題は、複数のプレイヤーが同じ物理オブジェクトに近接している場合である。その場合は、前述した不整合が発生するが、ある程度広大なフィールドで展開するゲームにおいては、レアケースである。

(c) 帯域幅の使用量を最小化する基本技術

① 情報のパッキング

ネットワークゲームの実装においては、他の技術ジャンルで見られるようなリッチプログラミングの文化は適用できない。メインプロセッサやグラフィックチップの性能向上ペースに比べ、ネットワーク帯域の向上ペースは極めて限定されており、また、帯域が向上した分はゲームの質の向上や、参加プレイヤー数の拡大といった方向に優先されるからである。また、帯域を節約すればするほど、ネットワーク負荷が下がり、サーバーの維持費用は安く上がるため、経済的な要求も大きい。

したがって、ネットワークゲーム実装のノウハウは、帯域幅の要求が厳しいものであった 1990 年代に、そのほとんどが完成を見ている。その中で最も基本的となる技法は、情報の適切なパッキングである。

つまり、ゲームサーバーが端末にゲーム世界の情報を送るとき、または端末がプレイヤーの入力情報をサーバーに送るときに、データが持つ実際の情報量を損なわずにデータサイズを可能な限り小さくするということである。

その基本的なアイデアは、「Quake」の実装に見ることができる。アナログモデムの通信環境で多人数対戦を実現しなければならなかったこのゲームでは、ネットワークに流す情報を厳選した上で、さらにそれを、各情報が必要とする最小のデータサイズまでビット単位で切り詰め、データパケットを作成している。さらに、オブジェクトの 3 次元位置情報を軸毎に分離し、変化したデータだけをパッキングしている。「Quake」では平面上の移動が多くなるため、多くのケースで縦軸の情報を送信せずに済むからである。また、同様のアイデアで、オブジェクトやキャラクターの各種情報を「変化したときにだけ送信

する」という仕組みを、エンジンに組み込んでいる。

現在のネットワークゲームでは、ブロードバンド化にともなって、ゲームを単に動作させるだけならばこの種の工夫が必要無くなっている。だが、帯域幅の要求量を数割減らすだけでも、サーバーの維持管理コストが大きく低下することもあるため、情報の適切なパッキングは依然として重要な技術である。

② 優先度ベースの間引き

ネットワークオブジェクトの状態変化をやりとりする際、ゲームプレイ上の重要度に応じてその頻度を制御する技法は、その努力を全くおこなわないことに比べると帯域幅の要求を劇的に削減することにつながるため、全てのネットワークゲームで重要である。

特に必須となるのは、1 つのフィールドに数十、数百のオブジェクトやプレイヤー・キャラクターが参加する、MMO タイプのゲームにおいてである。そのようなゲームでは、各端末に送信する情報を、該当するプレイヤーにとってどれだけ関心があるか、という特性をベースに優先度を付ける。このアイデアは最初期のネットワークゲームから実装されている。

基本的な技法としては「エリア・オブ・インタレスト」法が挙げられる。これは、プレイヤーにとって関心のあるネットワークオブジェクトを、ゲーム世界上の地域をベースに判定する方法である。判定は、プレイヤーからの距離、あるいは、障害物による可視・不可視の状況を用いて行なわれる。

実際は、ある一定の距離を超えると、そのオブジェクトを全く無視するという単純な方式が主流である。さらに気の利いた方法としては、近くのキャラクターのアクションは頻繁に、遠くのキャラクターのアクションはときどき送信する、という方法も使われる。また、FPS 系のゲームで、BSP 法などを用いて安価に高度な可視判定が行える場合は、不可視の位置にいるオブジェクトは距離が近くともその姿を送らず、音の情報だけを送る、といった活用が行なわれている。

また、ゲームプレイ上の重要度に基づく優先度付けも一部のゲームで行なわれている。例えば、多くの MMORPG では、敵キャラクターに最高の優先順位が付けられ、それにパーティメンバーが次ぎ、一般のプレイヤー、無害な NPC という順番になる。地域が混雑している際は、優先度の低いものから順に、その情報が送信されなくなる。

③ ネットワークオブジェクトの補完・予測表示

前節で情報の間引きについて説明したが、移動情報を間引かれたネットワークオブジ

ェクトを端末上でそのまま表示すると、位置が飛び飛びに表示されるという問題が発生する。また、ネットワークそのものがもたらすレーテンシーに加え、情報の更新頻度の低さによる表示遅延も発生するため、実際の位置と、表示上の位置に大きなズレが発生する。このことは、リアルタイム性の高いゲームでは致命的である。

これを見た目上改善するため、ネットワークオブジェクトの補完表示、あるいは予測表示といった技法が用いられる。まず、補完表示は、最初期は「Quake」で用いられていた技法で、ネットワーク経由で伝えられた位置情報のうち、最新のものと 1 つ前の物を使い、その 2 点間を時間軸にそって線形補完し、高いフレームレートで滑らかに動くよう表示する。

後者の予測表示は、おもにレースゲームで使われる技法である。補完表示では、サーバーとのレーテンシーのおよそ 1 倍から 2 倍分、端末の画面上のネットワークオブジェクトが遅れた位置に見えることになる。したがって、その瞬間、どちらが先に進んでいるかという情報が重要なレースゲームでは不適である。

このため、レースゲームでは、車両の過去の位置データを 3 点以上とり、これらをサンプルとして曲線補完のアルゴリズムを適用することによって、現在位置を予測して表示する。これにより、コースに対しての水平位置は、車両の速度が急激に変化している最中ではないかぎり、限りなく同時間性が確保される。基本的に、車両の運動が進行方向にたいして滑らかに変化することを利用した方法である。この技法は、アナログモデム経由の多人数対戦を最初期に実現したセガ「セガラリー2」のドリームキャスト版が最初期の実装例である。

またこの技法は、1999 年に発売された MMORPG「EVER QUEST」(SOE America) をはじめとする、多くの 3D MMOPRG でも活用されている。3D MMORPG では、キャラクターの移動データがそれほど重要ではないためあまり頻繁に通信されないものの、パーティを組んで複数人が 1 方向に進んでいるようなときは、進行方向に対する位置関係がなるべく同時性を確保できているほうが理想的であるため、レースゲームと同様のアルゴリズムが活用できるのである。

(d) ゲーム体験を強化するネットワークゲーム技術

① コンテンツ配信

Xbox 360 やプレイステーション 3 といったインターネット接続環境を前提とする家庭用ゲーム機が普及しはじめた 2006 年以降、ゲーム・コンテンツの配信を物理的な記録メディアに頼る事無く、オンライン上で行なう方式が注目されるようになってきた。

最新世代のゲーム機上では、パッケージタイトルのパッチや追加コンテンツを無料あるいは有料で配信する仕組みが標準化されているほか、一部のタイトルはゲーム本体がオンライン上で販売・配信されている(Xbox LIVE Arcade など)。また、PC プラットホームでは、Valve が「Steam」というコンテンツ配信システムを展開し、フルバージョンのゲームタイトルをオンライン販売する、新たな流通システムを形成している。

② ボイスチャット

近年では、エンドユーザー側の回線にブロードバンド水準の性能を期待できるようになったことで、最新世代のゲーム機あるいは PC 上でボイスチャットの利用が一般化している。これは、コンテンツストリーミングの一形態であるといえる。現在のボイスチャットシステムは、2～32 名ほどの小グループでの利用を前提にデザインされており、各プレイヤーの音声は特殊なエンコードを施されてサーバー役のマシンに集められ、そこから全プレイヤーに配信される。

また音響技術の専門企業ドルビーは、ボイスチャットの音声とゲーム中のキャラクターの位置や状態を空間的にリンクさせる技術を開発し、2008 年にミドルウェア「Dolby Axon」として発表した。この技術では、プレイヤーのボイスチャットの音声は、そのプレイヤーのキャラクターの位置から聞こえ、ゲーム中の遮蔽物など環境の影響を受けて音質が変化する。採用タイトルはまだ登場していないが、ゲーム世界の臨場感を高める技術として期待されている。

③ 非同期マルチプレイヤーゲーム

ソロプレイのゲームにコンテンツ配信の仕組みを組み込むことで、新しいスタイルのマルチプレイヤーゲーム体験を導入する例も出てきている。2008 年にリリースされた PC ゲーム「SPORE」(EA MAXIS)では、ユーザーが制作したゲーム内コンテンツを、バックエンドサーバーを通じて全プレイヤーに配信する仕組みを実装している。これにより、プレイする都度異なるクリーチャーが登場する、変化に溢れるゲーム世界を実現した。この種の方式は、非同期型のマルチプレイヤーゲームと言うことができる。

④ 観戦機能

ネットワークを通じ、ゲームプレイの様子をゲームに参加していないユーザーに見せるための機能を観戦機能という。2001 年の「Half-Life」エンジンアップデートにて実装

された「HLTV」を皮切りに、「Project Gotham Racing 3」(Bizarre Creations)、「鉄拳5 BR オンライン」(バンダイナムコゲームス)など、いくつかのゲームでこのような機能が実現されている。

技術的には、ゲームセッション内でやりとりされるプレイヤーの入力やキャラクターの情報を、ほとんどそのまま、ゲームに参加していないプレイヤーに対しても配信するものである。観戦者の端末上ではその情報とローカルに保持しているゲームデータアセットを使い、ゲーム中の風景を再現する。このため基本的には、観戦者もフルバージョンのゲームを所有している必要がある。

(3) ネットワークゲーム技術の現状と未来

(a) ネットワークゲーム技術の特性

① 基本的なゲーム実装技法は 1990 年代に確立・実証

第一章、第二章で記したように、ネットワークゲームの実装技法の多くは、早くも1990 年代には確立と実証が行なわれ、それ以降、同種の技法が現代まで応用されているのが実情である。

その理由を 2 点にまとめる。第 1 点としては、まず、ネットワークは本質的に「データをやりとりするパイプ」に過ぎず、技術分野としては I/O 処理に属し、ゲームのグランドデザインの中では、構造的に非常に低い階層に属するものであるため、ごく基本的なアルゴリズムがあらかた開発されたら、それ以降は変化しようがないということである。

理由の第 2 点目としては、ネットワークゲームを成立させるための技術が、ネットワークの「速さ」や「大きさ」が極めて限定されていた初期の時代にこそ強く求められていたことが挙げられる。つまり、2009 年の現在、恵まれたブロードバンド環境を前提にネットワークゲームを作るよりも、1993 年に低速なアナログモデムを前提にネットワークゲームを作るほうが遙かにハードルが高く、効率性の高いアルゴリズムが必要だった。

したがって、ネットワークゲームを構成する基本技術は、条件が最も厳しい時代に徹底的な研究が行なわれ、そして解決されてきたといえる。これは、画像フォーマット JPEG が 1986 年に標準化されて以来、いまだに主流の画像フォーマットであることに似ている。その後はインフラの発達のおかげで、ネットワークゲームの実装は回線や端末の性能を期待した「力づく」のスタイルでも可能となっているが、そのために、過去のタイトルよりも帯域要求の削減やレーテンシーの隠蔽が雑になり、プレイアビリティの低いネ

ットワークゲームが度々登場することがある。

② 共有されにくいノウハウ

実装技術が早くに確立した反面、ネットワークゲーム開発の現場では、かつて開発された実装技術に学ぶことが少ない。新たなゲームを開発する都度、各ディベロッパーの独自研究が行なわれ、結果的に「車輪の再発明」が行なわれているケースが多い。

その理由を断定することは難しいが、理由の一部として、ネットワークゲームロジックの実装は技術カテゴリーとして狭く、専門技術者が育つ環境が得られにくいこと、また、そのため、技術の研究・検証・発表・継承といったサイクルが作られにくいことが挙げられる。多数の専門家が存在する 3D グラフィックス技術とは対照的である。

こういった現状から、ネットワークゲームの実装における失敗が最新ゲームにおいても度々起こりうる。ゲームにつながらない、ゲームセッションが断絶する、同期ずれが起きる、レーテンシーの影響でプレイに支障を来す、ネットワークオブジェクトがワープしたり、消滅したりする。このような話を、現在のタイトルでも度々耳にする。ノウハウが共有されにくいことによって、このような不利益が起きている事実は、憂慮すべきである。

(b) 今後起こりうる変化

① 大規模化

インターネットが事実上標準の広域ネットワークとなって以来、ネットワークインフラの世界的な向上が続き、それに合わせる形で、ネットワークゲームの利用者も増え続けている。現在では、2008 年 12 月に、中国で展開しているオンラインアクションゲーム「地下城与勇士(邦題：アラド戦記)」が 100 万人の同時接続者数を達成するなど、顕著なサービスの大規模化が進行している。

このような大人数のプレイヤーを受け入れるためのサーバー技術は、ゲーム技術というよりは、より一般的なネットワーク技術による部分が大きいのが現状であるが、運営コストを圧迫することなく、さらなる大規模化を目指すためには、古来のネットワークゲームで使われていた最適化技法の徹底活用、見直し、あるいはオーバーレイ P2P ネットワークといった新たな技術の応用が必要になると考えられる。

一例として、アイスランドのオンラインゲーム開発企業 CCP が展開する「EVE ONLINE」では、2008 年末に“Stackless I/O”と呼ばれる技術をゲームサーバーに導入し、ゲームへの同時ログイン者数 30,000 人～40,000 人のうち、1,000～2,000 人のアクセス

が集中する地域の平均応答時間を劇的に削減することに成功した。これにより、ゲーム中の一地域に、さらに大人数のプレイヤーを受け入れることが可能になっている。
(参考：http://www.eveonline.com/devblog.asp?a=blog&bid=584)

② ユビキタス化

携帯ゲーム機やスマートフォンといったモバイル端末が一般に普及し、ネットワークゲームの舞台はさらに広がりつつある。それによる変化は、iPhone で大きな盛り上がりを見せるソーシャルゲームに象徴されているように、従来のネットワークゲームの概念を押し広げるものである。

また、2009 年には Google Android モバイルプラットフォームといった、非常に参入障壁の低い開発環境が登場する。そのため、この方向はさらに加速していくと考えられる。従来では考えられなかったユーザーコミュニティの大きさ、端末にひも付けられたソーシャルな情報、安価なコンテンツ配信システムなど、様々な特性を活用したネットワークゲームの登場が期待される。

③ 超広帯域幅のネットワークゲームデザインが可能に

回線速度の向上は、プロセッサの進化とほぼ同じく、ムーアの法則に従っており、およそ 10 年で 1000 倍程度のペースで進んでいる。エンドユーザーの通信環境に限っても、1999 年頃に主流であった ISDN モデムの 64kbps という通信速度に比べ、2009 年現在では光ファイバーによる 100Mbps の通信速度が一般的になっており、実効速度ベースで見ても 1000 倍程度の向上が見られる。また、それに対応するための基幹ネットワークの整備も常に進められている。

その変化を最も感じられる現在のネットワークサービスは、「ニコニコ動画」(ドワンゴ)をはじめとする動画のストリーミングサービスであろう。同サービスでは現在、リアルタイムの動画配信サービスとして「ニコニコ生放送」を展開しており、2009 年現在では、最大で 1 万人の同時接続者に対して、300～500kbps 程度の動画を配信することが可能になっている。これは、現在ネットワークゲームで使われている一般的な帯域幅に比べ、10～100 倍程度の速度である。

このペースで回線速度の向上が進んでいくとするならば、ネットワークゲームにおいても、プレイヤーひとりあたり数百 kbps～数 Mbps の帯域幅を使用することが現実的になってくると考えられる。その先駆けである「Second Life」では、ゲーム世界の様々なオブジェクトの情報をリアルタイムに配信する、コンテンツストリーミングの方式を実現していた。より低いコストで同様のことが可能になれば、同様の仕組みが一般的なネット

ワークゲームにおいても可能になるだろう。

現在のところ、そういったテーマに取り組んでいる具体例は見つけることはできないが、可能性としては、同一地域で数千人のプレイヤーが直接対決するようなアクションゲームや、世界の環境がダイナミックに変化していく「Second Life」型のゲームなど、プレイヤーの操作データだけでなく、ゲーム内コンテンツをリアルタイムに配信する方式を用いた、様々な応用を考えることができる。従来では考えられなかったようなリッチなネットワークゲームデザインが可能になるだろう。

3.2.12 開発方法論と、その歴史

長久　勝
ハイパーコンテンツ(株)/早稲田大学メディアネットワークセンタ

(a) はじめに

本稿では、ソフトウェア工学の文脈にそって、開発方法論について概観し、その歴史について述べる。特に開発プロセスに注目して述べる。ゲーム開発に特化した開発プロセスについても述べる。

本稿では、開発プロセスを「どのような成果物を、どのような順序で作成するか、明確に規定したもの」と定義する。この概念は、近年の大規模なゲーム開発において、コンテンツパイプラインを構築する上で、基底となるものである。

既に「コンテンツパイプライン」という言葉は、商業的ゲームを扱う開発者に広く浸透しているので、開発プロセスの概念についても同様であると考えがちであるが、これは正しくない。例えば、筆者が、ある企業で 100 名近い開発者に行ったアンケートでは、XP や Scrum など近年注目されることが多い開発方法論の認知度に比べて、それらの開発方法論を支えている PDC(S)A や Lean の認知度が明らかに低かった。

一般的に、各種開発方法論の現場適用に際しては、テーラリング(tailoring)が必要だとされている。しかし、ゲーム開発に限ったことではないだろうが、開発プロセスについて深く掘り下げて議論できるほど、開発者に十分な理解が広まっているとは言えないのが現状である。

本稿の目的は、開発方法論を、その成り立ちも含めて紹介することで、その背景について理解を促し、開発方法論について考えるための基礎知識を提供することである。

一般的なソフトウェア開発では、ゲーム開発とは異なり、3D モデル、テクスチャ、音声といったコンテンツを大量に制作することがない。一般的なソフトウェア開発チームは、ほぼコンピューター技術者だけで構成される。ゲーム開発の立場から、一般的なソフトウェア開発プロセスを見る場合には、注意が必要である。

(b) ウォーターフォール型(waterfall model)

(1) 概要

最も古いソフトウェア開発プロセスとして知られているのがウォーターフォール型である。ウォーターフォール型とは、要求獲得、仕様策定、設計、実装の工程間にフィードバックがない開発プロセスである。上流の工程に不備がないことを前提としているので、不備を検出した場合に、工程を遡って修正する方法が、明確に定義されていない。このため、不備のある部分以降の全ての工程をやり直すことが多い。

開発プロセスに現れる工程は、粒度や関心事に応じて異なる場合がある。これはテーラリングの問題である。最も標準的な工程のセットは、要求獲得(どんな問題を解決するものを作るのか)、仕様策定(どんなソフトウェアを作るか)、設計(どう作るか)、実装、試験、運用、である。

各工程の成果物は文書とすることが求められる。各工程の成果は、文書を通じて、次の工程に伝えられる。文書によって開発プロセスが進行することを「文書駆動」と呼ぶ。

ウォーターフォール型は、要求の変更や、不備の修正への対応が困難である。厳密に書かれた文書は、開発の手引きとして有効だが、更新コストが高い。しかし、ウォーターフォール型で開発が行えた場合、変更や修正のコストが不要となり、最少のコストで開発が可能とも言える。一般的には現実的な解ではないが、ウォーターフォール型開発が可能な場合においては、適用の価値がある。例えば、開発対象に関わる全てのステークホルダが既知であり、互いのコミュニケーションが十分であれば、見落としなどに起因する要求の変更は起こりにくいし、開発対象が扱う問題領域への理解が十分と考えられる派生開発であれば、全ての工程において不備の入り込む可能性は低くなるので、適用が可能な場合もある。

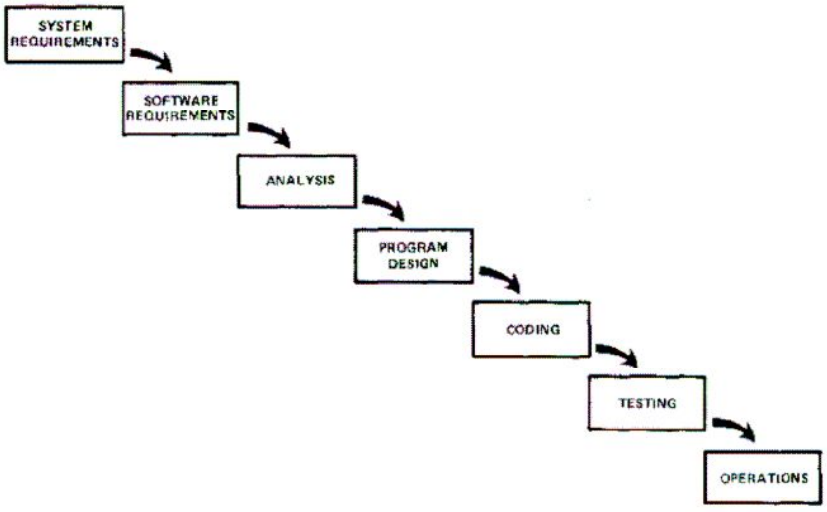


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

図 3.2-95 ウォーターフォール型の開発工程([1]より引用)

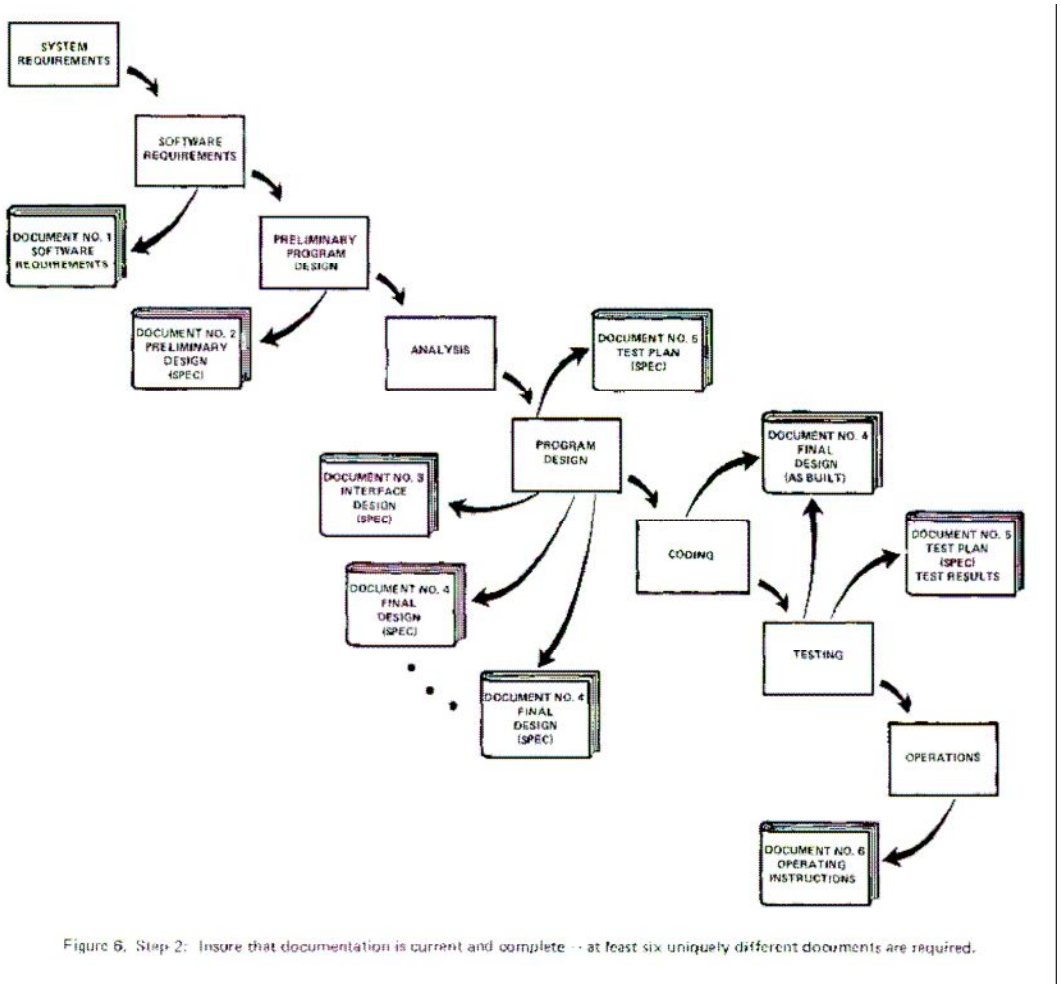


Figure 6. Step 2: Insure that documentation is current and complete -- at least six uniquely different documents are required.

図 3.2-96 各工程の成果物を定義([1]より引用)

(2) 成り立ち

ウォーターフォール型は、1970 年の Winston W. Royce の論文「Managing the development of large software systems」[1]で始めて提唱されたと言われている。しかし、この論文には、工程間のフィードバックを持つ開発プロセスについても記述がある。どちらかと言えば「どのような成果物を、どのような順序で作成するか、明確に規定」するための枠組として、要求獲得から運用までを工程として定義した論文、と評価した方が良い。そもそも論文内に「waterfall model」の記述は現れない。ウォーターフォール型は、Royce の論文から部分的に抜き出して、構築された概念と言える。

では何故、ウォーターフォール型の概念が構築されたのだろうか。そもそも有用でない概念を構築したり、それが広まったりすることは稀であり、少なくとも 1970 年代には、それが有用であると考えられたはずである。ウォーターフォール型が現れた理由は、当時の大規模開発の特徴に見ることができる。

1960 年代、米での大規模なソフトウェア開発は、航空宇宙と軍事の分野で行われてい

た。当時、Royce が所属していた TRW 社では、宇宙活動の計画、実行、事後分析のためのソフトウェア・パッケージを開発していた。Royce の論文も、TRW 社での経験に基づいている。

こうしたソフトウェア開発は国家レベルのプロジェクトであり、潤沢な資金で優秀な専門家集団を維持し、継続的な開発を行っていた。例えば、有人宇宙計画に限っても、マーキュリー計画からアポロ 11 号の月面着陸まで、10 年にわたる継続的な開発を行っていた。これは、開発対象が扱う問題領域への理解が十分蓄積され得る状況、ウォーターフォール型が適用しやすい状況と言える。また別の言い方をすれば、継続的な開発の中で、複数回のウォーターフォール型開発が実施され、事実上は反復型だったと言える。

その一方で、当時は変更管理を行う技術が未熟であった。変更や修正が各工程に及ぼす範囲を特定することは困難であり、各工程で分業する開発者の間で済ませてしまうにはリスクがあった。実際、当時の NASA では、プロジェクトにおいて変更管理を行う専門チームが置かれ、工程間の調整を担っていた。しかし、民需分野でのソフトウェア開発においては、コストの問題から、変更管理チームを運用できるとは限らなかっただろう。

結果的に、先行分野で実績のあるウォーターフォール型に、コストのかかる変更管理対応を付けない形で、民需における開発プロセスの標準が成立したものと考えられる。

また、ソフトウェアを支える理論体系が数学を基盤としており、目的が明確であれば意味的に正しいソフトウェアを形式的に導出できるという形式手法(formal method)の考え方が、既に 1960 年代に存在していた。この考え方が、変更や修正への無関心を招いた可能性もある。今日では、一般的なソフトウェア開発において、要求の獲得は困難で、目的を明確にすることが難しいことが知られており、形式手法を主ではなく従として用いる軽量な手法が提案されている。

ウォーターフォール型は、1960 年代の 2 つの事情、問題領域への理解が十分な分野での開発が多かったこと、変更管理を行う技術が未熟であったこと、を背景に、Royce の論文を解釈し、成立したと言える。この事情が、1970 年代においても変わらなかったため、ウォーターフォール型が広まったと考えられる。

1985 年、米国防総省は、ウォーターフォール型開発を調達要件とした DOD-STD-2167 を制定したが、コンピュータシステムの利用が広がった 1980 年代には新規性の高い案件が多くなり、ウォーターフォール型を一般的に適用することは、既に不可能であった。DOD-STD-2167 に基づいたプロジェクトのうち、キャンセルされず実用にまで到達したのは、全体の 1/4 しかなかったと言われている。

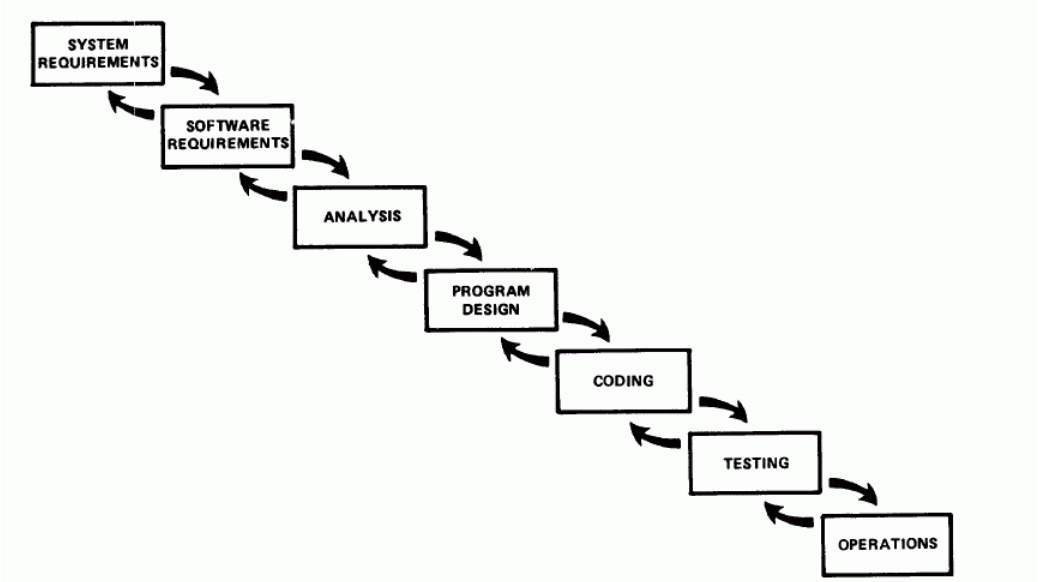


Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

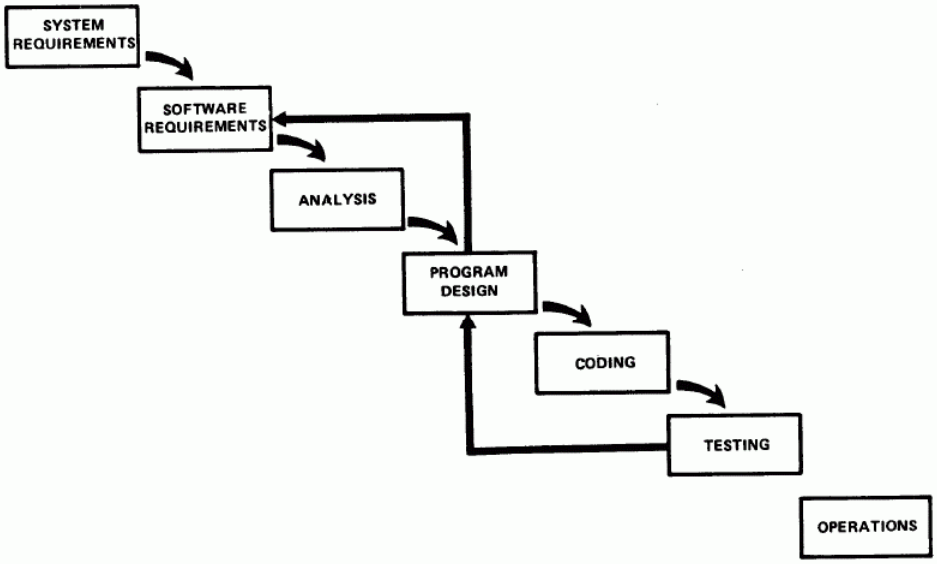


Figure 4. Unfortunately, for the process illustrated, the design iterations are never confined to the successive steps.

図 3.2-97 Royce[1]で扱われている非ウォーターフォール型の開発プロセス([1]より引用)

(c) 試験

ウォーターフォール型であっても、逃れようのない変更・修正を要求する工程がある。試験がその工程である。実装工程の成果物が試験をパスできない場合は、その不具合の原因を作り込んだ工程までさかのぼり、対応が必要となる。この試験による変更・修正を説明するのが「V 字モデル」である。

V 字モデルは多段階の試験構造を持ち、実装上の不具合がなく設計を満たしているか、

仕様を満たしているか、要求を満たしているか、を順に試験する。実装上の不具合や設計を充足していない場合は実装工程に、そこで不具合が見つからず、仕様を満たしていないことが分かった場合は設計工程に、更にそこで不具合が見つからず、要求を満たしていない場合は分析・仕様化工程に、そこで不具合がなければ要求獲得工程に、それぞれさかのぼって変更・修正作業を行う。

さかのぼって変更・修正した場合、当然、それ以降の工程も影響を受けるため、順次作業が必要となる。これは、要求獲得工程の変更・修正を行った場合、それ以降の仕様策定、設計、実装、全ての工程で作業が発生することを意味する。ここで大きな問題となるのが、既に試験をパスした工程で変更・修正が生じるため、試験も全てやり直しとなることである。要求の不具合が見つかるのは、実装、設計、仕様の試験をパスした後なので、要求の変更・修正に紐付いて、仕様、設計、実装の変更・修正が生じ、パスした試験に再度挑むこととなる。

この問題に対しては、事前の試験設計と試験の自動化による回帰試験の効率化や、疎結合なモジュール設計による変更範囲の局地化などの対策が取れる。

また、変更管理が機能すれば、変更・修正の影響が追跡可能となり、影響範囲を局地的に見られるようになるため、各工程の成果物のレビュー精度が上がる。レビューのコストが下がれば、試験ではなくレビューで不具合を検出することも対策となり得る。こうした観点から、試験では実装のバグのみを対象とし、設計以前の工程の不備は、それぞれの工程、もしくは直後の工程でレビューによって発見し、即座に変更・修正を行う開発プロセス「動的 V 字モデル」が提唱されている。動的 V 字モデルは、工程間を密につなぎ、定常的なフィードバックを行うため、ウォーターフォール型ではない。

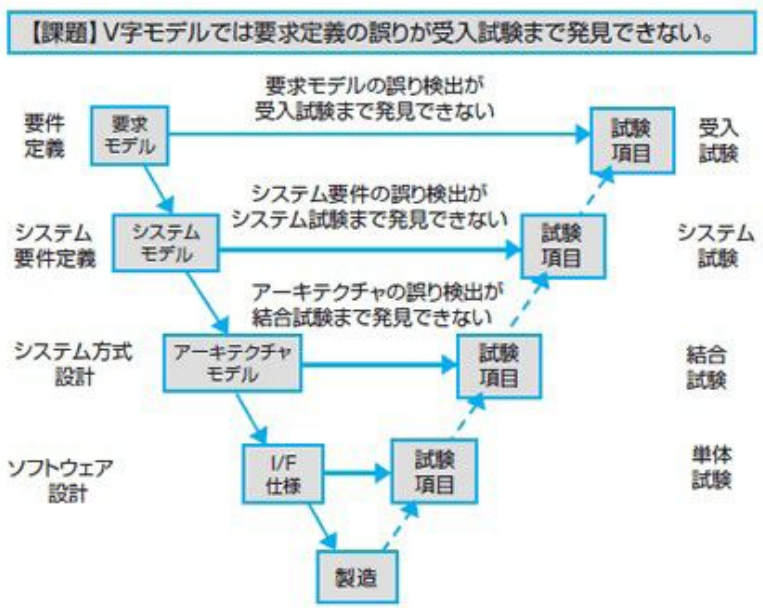


図 3.2-98 V 字モデルの例([2]より引用)

(d) 変更管理

後工程で前工程の不備が見つかった場合、要求が変更・追加された場合、複数の工程にまたがる作業が発生し、各工程の成果物がそれぞれ更新される。この際、各工程の成果物の間で、内容的に同期が取られていないと、成果物を信用して使うことができない。各工程の成果物を正しくリンクした状態に管理するのが、変更管理である。ウォーターフォール型に工程間のフィードバックがないのは、困難な変更管理に踏み込まないためとも言える。

変更管理で重要なのは、各工程の成果物の間で、変更点の意味的な追跡可能性を実現することである。これが実現されていれば、変更の影響範囲だけに注目したレビューが可能となり、内容が同期していることも確認しやすい。

ウォーターフォール型より後に登場する、変更を意識した開発プロセスでは、変更管理をツールの支援で実現している。変更管理の最初の一手は、構成管理ツール(SCM)の導入である。ゲーム開発においても、この 10 年で、バグ追跡システム(BTS)と共に、利用されるようになった。

しかし、ファイルの変更管理では、意味的な追跡可能性までは実現できない。ある時点で追加された要求が、その後、実装のどの部分で実現されたかは、分からない。各工程の成果物について、ある工程の成果物のどの部分が、別の工程の成果物のどの部分と対応しているか分からない。

意味的な追跡可能性を実現するには、各工程の成果物の間で関連する部分を紐付ける必要がある。例えば、機能単位のタグを用意し、各工程の成果物で関連する部分に付与しておき、ある機能を軸に各工程の成果物を串刺しして見られる仕組みなどが有効である。文書の体裁にルールを加える、文書群をパースしてタグ単位で関連する部分をまとめて見られるビューアを用意することで実現できる。各工程の成果物を、横断的関心事で串刺しして捉える、と考えるわけである。

(e) 反復型(iterative and incremental development)

(1) 概要

Royce は要求獲得から実装に到る工程を定義した。これ以降、工程の流れをどうデザインするか、開発プロセスが議論されるようになる。既に Royce が指摘しているが、フィードバックをどう組み込むかが、その議論の中心となった。

レビューや形式仕様記述など、各工程の成果物を検証する技術が十分でないと、様々な不備は試験で集中的に見つかる。V 字モデルはこの特徴に基づいている。試験でまとめて不備を検出するのは効率的である。

そこで、フィードバックは試験後にまとめて行うとし、「要求定義～実装→試験」を繰

り返す、反復型のプロセスを通して、漸近的にあるべき最終成果物を目指す手法が使われるようになった。V 字モデルでは、各種試験に応じて、さまざまな工程へ差し戻していたが、反復型では、可能な範囲で試験を実施し、必ず要求定義工程に差し戻す。各工程の成果物は、1 周につき 1 回しか更新されないので、変更管理は容易である。試験が終了した時点で、全ての成果物は同期している。

反復型には、繰り返し(iteration)をどう実施するか、いくつかのバリエーションがある。ウォーターフォール型と同様に厳密な要求定義から始め、繰り返す度に変更・修正が減っていくことを期待するもの。厳密な要求定義から実現すべき機能の依存関係を分析し、繰り返し毎に機能を増やしていくもの。本質的な要求の実現を目指すところから始め、繰り返しを通じて要求を掘り起こしながら、最終形を固めていくもの。などがある。

厳密な要求定義から始める場合には、各工程の成果物の生成・更新のコストが、ウォーターフォール型と同程度となり、コストが高い。この手法が、Royce が意図していた大規模開発とも言える。大規模開発では、開発者が 1000 人を超える事例も多く、情報伝達は文書に頼らざるを得ない。このため、文書駆動の色合いの濃い、この文書駆動的反復型は、現在も利用されている。現在では、文書の変更管理も含め各種開発支援ツールが利用できる。IBM 社の RUP などが、ここに分類される。

一方、本質的な要求の実現を目指すところから始め、繰り返しを通じて要求を掘り起こしながら、最終形を固めていく場合は、文書駆動のメリットよりも、繰り返しによる漸近的アプローチのメリットに注目し、コストを抑えた開発プロセスとなっている。このインクリメンタルな反復型は、厳密に文書化する代わりにチームの構成員の間での情報共有を必要とする。このため小規模な開発に向いている。但し、現在では、wiki などコストの低い文書化ツールが利用できるため、適用可能な規模が広がった。インクリメンタルな反復型は、アジャイル型の基本にもなっている。最後の周以外をプロトタイピングとし、最後の周以外の工程の負荷を軽減した Boehm のスパイラル型などが、ここに分類される。Boehm のスパイラル型は、ゲーム開発や映像制作に見られる、プリプロダクションとプロダクションのフェーズ分けに、ほぼ相当する。

反復型で問題となるのは、何周すれば求める最終成果物が得られるかである。この収束速度は、問題領域の知識の有無など、チームの能力に依存する。文書駆動的反復型では、2 ないし 3 周程度を想定した見積もりで行われることが多い。新規性の高い開発にインクリメンタルな反復型を用いる場合は、不確定要素が多いため見積もりが困難であるが、チームの過去実績に基づいて見積もられることが多い。近年では、パッケージではなくサービスとして提供されるソフトウェアについて、開発終了の予定を立てずに、継続的にリリースしながら開発する場合があり、随時マイルストーンの予定を更新しながら、インクリメンタルな反復型が使われることがある。

プロジェクト管理の観点からは、繰り返しを区切りとして、契約を段階的に結んだり、プロジェクトをキャンセルしたりすることが可能である。顧客に選択肢が与えられ、プロ

ジェクトの方向を修正できる。これは一般的に利点であるが、足並みの揃わないステークホルダが、プロジェクト内で綱引きを始め、プロジェクトが迷走する場合もある。プロジェクトに加えられる変更は、常に収束を念頭において、要求されるべきである。

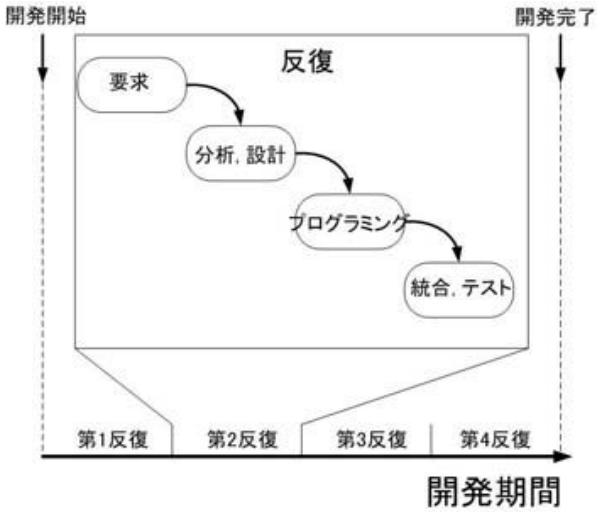


図 3.2-99 反復型の模式図([3]より引用)

(2) 成り立ち

「History Of Iterative」[4]によると、Shewhart と Deming による PDC(S)A サイクルが反復型の起源である。PDCA サイクルは、1950 年代の X-15 高高度極超音速実験機の開発を経て、1960 年代のマーキュリー計画に伝わり、計画に参加していた IBM 社の政府向けシステム事業部(FSD)に伝わった。これ以降、FSD はソフトウェア開発に反復型を取り入れた。

1970 年代に入り FSD はトライデントミサイル搭載潜水艦の指揮統制システムを、6 ヶ月周期 4 周の反復型開発で、遅延なしに仕上げた。同時期に、Royce が所属していた TRW 社では、弾道ミサイル防衛網システムの開発を、3 年ほどの間に 5 周の繰り返しを行う形で実施している。案件の性質上、文書駆動的反復型で実施されたものと考えられる。

FSD では、1970 年代中頃に、対潜ヘリ LAMPS のシステム開発を、2400 人月、4 年をかけて、1 ヶ月周期 45 周の反復型で実施している。期間中の増減を無視した平均で言うと 50 人のチームであるが、この人数で 1 ヶ月周期を実現したことは特筆に価する。文書の整備などを考慮せず、周期の長さ、繰り返し数だけを見れば、これはかなりアジャイル型に近い。

文書駆動的反復型の経験から、繰り返しを重ねれば最終成果物の品質が向上することが分かったため、短い繰り返しを重ねて、同じ期間でもより多くの繰り返しを実施することが指向されるようになった。1980 年代には、実装する機能を少しずつ加えていくイン

クリメンタルな反復型と、初期の繰り返しをプロトタイピングと位置付け、文書化などの作業を軽減した Boehm のスパイラル型が現れた。

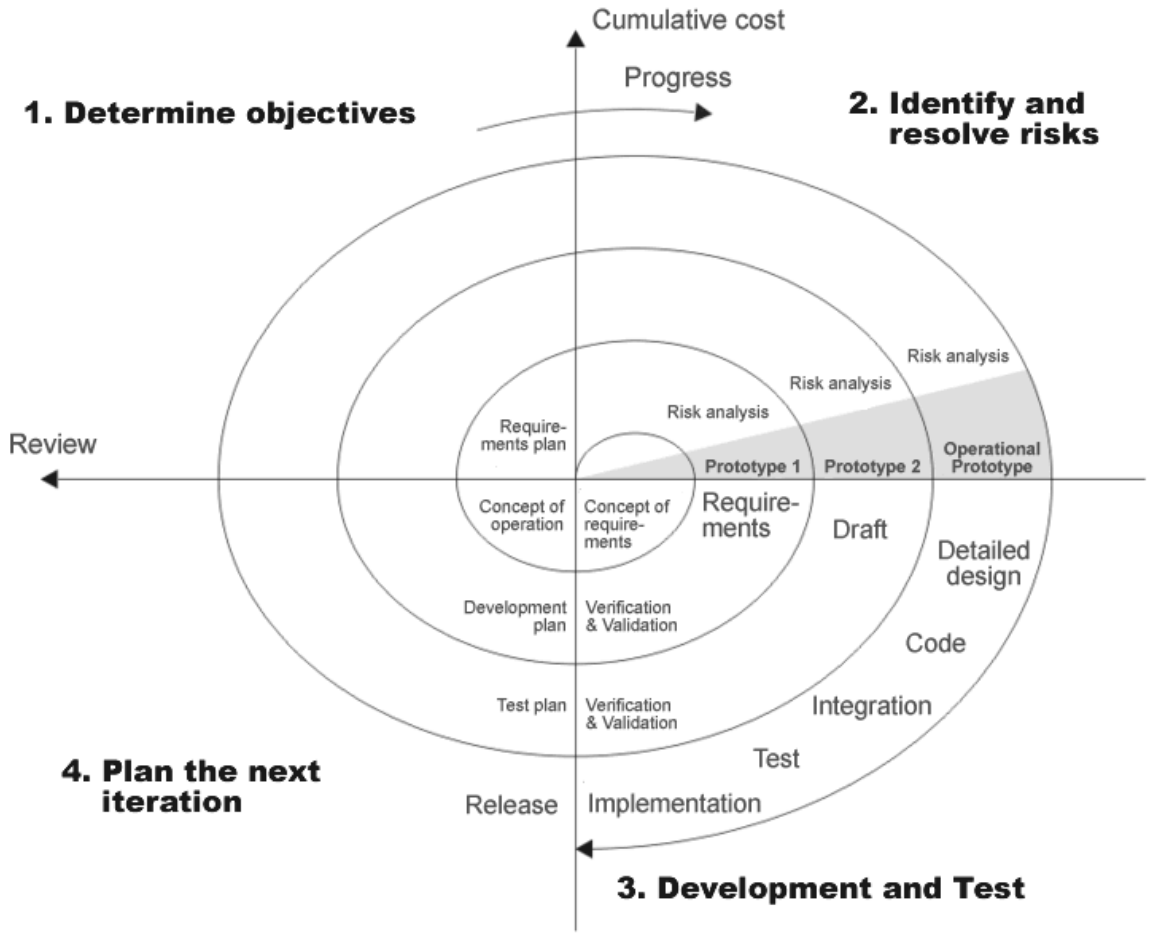


図 3.2-100　Boehm のスパイラル型の模式図([5]より引用)

(3)　PDC(S)A

反復型開発において、何を行うか計画し(Plan)、その計画に基づいて行動し(Do)、その結果を吟味し(Check)、改善を行う(Action)、4 つの行動を繰り返すサイクルを、PDCA サイクルと呼ぶ。PDCA サイクルは、モノ作りの現場への洞察から生まれたが、今日では、サービス業など、モノ作り以外でも広く用いられている、改善活動の枠組みである。Walter A. Shewhart の示した概念を W. Edwards Deming が整理し、1939 年に出版した「Statistical method from the viewpoint of quality control」が、その原点と言われている。

Shewhart は、AT&T ベル研究所に設立当初から所属していた、統計的品質管理の第一人者である。Shewhart は、ベル研究所所属以前に、機器製造の検査技術部門で、管理図

を考案し、統計的品質管理を実践した。Deming は、統計的品質管理を Shewhart に学び、その概念が製造現場だけでなく広く応用可能なことに気付いた。Deming は、製造現場への統計的手法導入を行う一方、経営レベルでの持続的改善活動として PDCA を説いた。

改善活動は、日本の製造業の得意分野であり、戦後の高度経済成長期を支えた強みであるとして、Kaizen が海外でも知られている。この日本の製造業における改善活動は、Deming が日本に持ち込んだ統計的手法を基盤としている。Deming は、自身が開発した標本化技法が米の国勢調査に採用されたことから、戦後の日本における国勢調査の現地支援を依頼され来日した。1950 年、日本科学技術連盟に請われた Deming は、統計的手法に関する講義を数多く行った。Deming から学んだ日本の製造業各社は、統計的手法を用いて、品質と生産性の向上を図った。

Deming は戦中の米において、戦時の工場に対して、統計的手法の指導を行った。戦中は広く適用されたが、戦後、米経済が強くなり、改善活動より規模の成長が重視されるようになると、航空宇宙及び軍事などの分野以外には適用されなくなった。このため、米では、1980 年代にマスコミに取り上げられた後、PDC(S)A の概念と実践を、日本から逆輸入する形となり、今に到っている。日本の製造業の現場で進化した PDC(S)A の概念と実践は、米への逆輸入の後、Lean として整理された。

1980 年代になると、Check について、事前に設定した効果水準を見るだけではなく、事前には分からなかった知見を汲み取ることの重要性が認識されるようになり、Deming は、Check に替えて Study を用いるようになった。

(f) アジャイル型(agile software development)

(1) 概要

2001 年 2 月 11 日～13 日、従来手法よりも軽量な開発プロセスを提唱する著名な 17 名が、ユタ州スノーバード・スキーリゾートに集まり、各手法の統合について議論し、4 つの価値と 12 の原則をアジャイル宣言としてまとめた。これ以降、主に 1990 年代から提唱されていた各種の軽量な開発プロセスは、アジャイル型と総称されるようになった。

アジャイル型は、インクリメンタルな反復型を基盤とし、繰り返しによる適応を重視する。繰り返しの足を引っ張る文書駆動を廃し、文書化を必要最低限とし、チーム内でのコミュニケーションでそれをカバーする。文書駆動では、あらゆる情報を形式知とすることが求められるが、アジャイル型では、必要最小限の形式知と、チームが保持する暗黙知の組み合わせで、文書駆動と遜色のない情報環境を構築する。チームに変動があると、機能しなくなる危険があるが、チームに経験を積ませることの重要性は広く知られており、リスクはそれほど高くはならない。また、チームの規模が小さくなるほど形式知の必要性も小さくなる。

アジャイル型では、チーム内の暗黙知が活用されるが、文書化された形式知に比べて、情報の更新を行いやすい。例えば、簡単な立ち話で認識を擦り合わせ、メモを wiki に残す程度のコストで可能である。これは従来、問題となってきた変更管理コストを抑える、一つの答えとなっている。とは言え、この手法で賄える規模には限界があることも、認識しておく必要がある。

現在では、各手法を、複数の実践(practice)の集合体とみなし、実践単位で組み合わせてテーラリングが行われている。代表的な実践として、オンサイト顧客、短期リリース、受入テスト、テストファースト、計画ゲーム、シンプルな設計、ペアプログラミング、頻繁なリファクタリング、チームでのコード共有、継続的なビルド、持続可能なペース、コーディング規約、システムのメタファー、組織の自律性重視、日次ミーティング、イテレーション中の仕様変更禁止、リーダーによるファイアウォール、1 時間以内の判断、1 日以内の障害の排除、ミーティングからの部外者の排除、7 人程度のチーム、共通の部屋、リリース時のレビュー、頻繁なふりかえり、要求の見える化、品質の見える化、進捗の見える化、などが知られている。また、ソフトウェアかんばん、など、特定の手法と結び付かない独立した実践も提案されている。

実践の間には、しばしば関連性があり、同時に行うとより効果の高いもの、同時に行えないものがある。例えば、持続可能なペース、と、1 日以内の障害排除、は衝突する可能性がある。

ゲーム開発にアジャイル型を取り入れるには、注意すべき点がいくつかある。ゲーム開発は、ソフトウェア技術者だけで行われているのではなく、最終成果物も、その大半がデータだからである。例えば、開発チームが多職種で構成されるので、発生する問題の幅が広くなり、ソフトウェア開発に特化した問題解決に関する実践は、応用の可能性は残るが、適用は難しい。また、データを成果物と見た場合、手作業で大量に作られた成果物に、頻繁に手を入れることは現実的ではないので、リファクタリングは、ツールによる一様な変換が可能な範囲でしか実施できない。

他にも、ゲーム開発では、要求を捉えることが困難で、要求を外在化せずに、いきなり仕様化することが多く、要求に関する実践は難しいだろう。顧客を誰に設定するか、エンドユーザーか、パブリッシャーか、で、オンサイト顧客など、顧客に関する実践の具体的な実現方法が変わってしまう。

ゲーム開発にアジャイル型を取り入れるには、テーラリングが非常に重要となる。

(2) 成り立ち

1980 年代後半から、インクリメンタルな反復型の適応的側面が重視されるようになった。理由は、社会におけるコンピューター利用が進み、新規性の高い小規模な案件が増え、計画重視の手法が適さない開発が増えたためである。アジャイル型の各手法の成立時期を

見ると、初期に開発プロセスに関する手法が成立し、その後、実装技術寄りの手法が成立している。

Deming は、1950 年代以降、PDC(S)A による品質改善を日本に持ち込んだ。これが基礎となり、現場主導の改善活動が、日本の製造業で確立される。1986 年の「The New New Product Development Game」[6]には、1976 年から 1982 年の、日本企業の製品開発事例の分析が集められているが、あいまいさの許容、開発チームの自己組織化、工程の境界をオーバーラップさせる、要員に複数のスキルを持たせる、微妙な管理、組織的な知識伝達、などを特徴的に述べている。また、1980 年代、トヨタ生産方式を研究した MIT のグループから Lean が提案された。1993 年、ソフトウェア開発において、これらの影響の下に生まれたのが、開発プロセス寄りのアジャイル型 Scrum である。

開発プロセス寄りのアジャイル型が用いられるようになると、これに対応した実装技術も議論されるようになった。繰り返しによって、変更・修正の入りやすい枠組みとなり、変更・修正のための実装への取り組みが必要となった。また、同じ時期、オブジェクト指向の概念が普及し、変更・修正を受け入れやすいアーキテクチャが使われるようになった。こうした状況下で、1996 年に、実装技術寄りのアジャイル型である XP が生まれた。

最近の話題としては、オープンソース IDE である Eclipse の開発を通じて得られた知見を、商業製品開発に適用しようとする、IBM の JazzProject がある。JazzProject では、アジャイル型の実践に、オープンソース開発、分散拠点開発、大規模開発のための実践を組み合わせた「Eclipse-Way」を提唱している。

(3) Lean

1988 年、GM とトヨタの合弁会社 NUMMI に在籍していた John Krafcik(現 HyundaiAmericaCEO)は MIT での修士論文「Triumph of the Lean Production System」の中で、カンバンなどからなるトヨタ生産方式を紹介し、分析した。MIT が現在も進める国際的な自動車産業研究によって得られた成果は、1990 年、「The Machine That Changed the World」として出版され、Lean の名で米の製造業に広く普及した。2000 年以降は、日本にも紹介されるようになった。

トヨタ生産方式では、徹底した無駄の排除が指向され、これを現場の改善によって実現している。藤本[7]によれば、部品が製造ラインの途中で作業を受けていない時間を徹底的に減らし、続いて要員の空き時間をなくすように、工程を改善していくとのことである。

日本の製造業では、工程の境界をオーバーラップさせ、要員に複数のスキルを持たせることで、要員の前後工程への理解を促し、改善を生みやすくなっている。しかし、要員の視野には限界があり、局所最適な改善に止まる可能性がある。MIT はここに注目し、トップダウンの改善も組み合わせることで、トヨタ生産方式を Lean に進化させた。

ゲーム開発においても、コンテンツパイプラインのデザインに不備があると、要員の空き時間が生まれる。製造業と同じ問題構造であるので、Lean のアイデアは十分に機能すると考えられる。製造業との差異を考える上では、各種産業を設計情報転写のバリエーションとして捉える藤本の理論が役に立つだろう。

(4) Scrum

Scrum はプロセス管理を重視したアジャイル型で、ゲームも含め、多くのソフトウェア開発で、プロセス管理が十分でなかったため、適用事例が増えている。従来手法よりも管理コストが低くなるので、コスト圧力からインフラ系の大規模開発でも適用事例が増えている。500 人を超えるプロジェクトの例もある。

「The New New Product Development Game」[6]では、比喩としてラグビーが使われていることから、Scrum と名付けられた。

Scrum では、ゴールを吟味し、タスクに分割し、インクリメンタルな反復型開発を行う。繰り返しの中でゴールは修正され、適応的な開発が行われる。開発チームの規模としては 10 名未満が推奨されているが、開発チームの代表者を集めて上位チームを作る ScrumOfScrum と呼ばれる手法で、大規模開発に対応可能だとされている。上位チームが 10 名を超える場合には、複数の上位チームに分け、上位チームの代表者を集めて最上位チームを置く、3 階層構造とすることも実践されている。

Scrum では、開発チームの取りまとめ役 ScrumMaster、管理者 ProductOwner、開発チーム、顧客、からなる役割モデルが採用されている。

具体的には次のような手順で開発が行われる。

まず、関係者は、実装すべき機能や内容をリスト化し、ProductOwner が優先順位を付ける。出来上がった優先順位付やることリストは ProductBacklog と呼ばれる。

次に、ProductBacklog の上位何項目を、次の Sprint(1 ヶ月程度の開発期間)で実装するか、関係者全員で決める。対象項目を実現した実行可能なソフトウェアを実装するための作業を、直接の実装行為以外の作業も含め、4 から 16 時間程度に詳細に分割し、Sprint 中に行うべき作業リスト SprintBacklog を、開発チームで作る。これらを決める会議を SprintPlanMeeting と呼ぶ。

Sprint を実施する。Sprint 中、外部からの変更要求は受け入れない。毎日 15 分程度の進捗会議 DailyScrum を実施する。必要に応じて問題解決のための会議も実施する。SprintBacklog の消化状況は、BurndownChart と呼ぶグラフに示し、開発チームで開発速度を意識する。

Sprint の成果物を SprintReviewMeeting にかける。プレゼン資料でなく、動くソフトウェアで説明する。結果を元に、ProductBacklog を更新し、SprintPlanMeeting に戻り、繰り返しを続ける。ProductBacklog が空になったら開発終了とする。

ProductOwner は、何を作るかを仕切る。要求を顧客から吸い上げ、順位付け、整理して開発チームに示す。ScrumMaster は、どう作るかを仕切る。進捗管理と、問題解決を担当する。

Scrum では、実装に関する実践を規定しないため、他の手法と組み合わせる余地がある。最も広く実践されているのは、Scrum と XP の組合せである。

近年、海外のゲーム開発では、Scrum を適用した事例が増えている。経験に依存した局所最適の可能性が高い独自の開発プロセスではなく、実績がある Scrum を経験に基づきテーラリングする流れとなっている。特に、面構成を持つゲームなど、レベルデザインを面やマップ単位で並行して行えるゲームでは、ScrumOfScrum を用いて、品質や納期を向上させた報告が多い。

(5) Cabal

1990 年代後半、Valve 社が Half-Life を開発する際に用いたのが、ScrumOfScrum とエンドユーザー重視を組み合わせた開発プロセス Cabal[8]である。

FPS 開発で重要なのはゲームエンジンとレベルデザインである。近年、mod やゲームエンジンの外販が行われるため、ゲームエンジンについては、技術者を中心とした専属の開発部隊を置く場合が多い。従って、実際のタイトル開発は、レベルデザインに多くを負っている。

Cabal では、FPS がマップ単位で独立したレベルデザインを行うことに注目し、ScrumOfScrum を用いて、各マップを並行して開発できる体制を敷いた。並行開発を実現したことで、大規模化が可能となった。各開発チームは、独立して開発を進めるが、ゲームエンジンに関する問題や、レベルデザインのノウハウなど、共有すべき情報は、上位チーム経由で流通させることができる。

また、オンサイト顧客の実践として、頻繁なプレイテストを行い、定常的なフィードバックを受け、顧客のニーズを確認しながらの開発を実現している。一般的に、定常的なフィードバックは、開発におけるノイズになりがちであるが、ゲームエンジンの開発を切り離し、レベルデザインに特化したことで、繰り返しの周期を短くすることに成功し、頻繁なフィードバックを受け入れられる体制になったと分析できる。

Cabal は、アジャイル型の観点から見た場合、オンサイト顧客、短期リリース、組織の自律性重視、共通の部屋、などの実践を確認でき、FPS 開発に特化した開発プロセスとして稀有なものである。北米では、Valve 社の成功を受け、Epic 社など他社にも追従の動きがある。

(g) さいごに

限られた紙面で駆け足となったが、ソフトウェア分野における、開発プロセスについての開発方法論を、一通り述べた。欧米では、事例など豊富に文献があるが、日本語で短くまとめたものは見ないので、本稿が少しでも役立てば幸いである。

本稿には、歴史と共に、最近の視点からの分析を併記した。分析を述べた部分に関しては、筆者による一つの解釈とご理解頂き、参考に止めて頂きたい。

本稿の結びを「まとめ」としなかったのは、まだ、まとめる程、さまざまな事柄が明らかでないと考えたからである。興味を持たれた読者が、自分なりの「まとめ」を得るために、調査・研究に挑戦されることを願い、筆を置くものとする。

表 3.2-07　関連年表

年	出来事、論文、刊行物など
1939	Walter A. Shewhart, W. Edwards Deming "Statistical method from the viewpoint of quality control"
1950	Deming が日本に PDC(S)A を紹介
1970	Winston W. Royce "Managing the development of large software systems"
1985	DOD-STD-2167
1986	野中 郁次郎, 竹内 弘高 ”The New New Product Development Game”
1988	Barry Boehm "A Spiral Model of Software Development and Enhancement"
1990	James Womack, Daniel Jones, Daniel Roos “The Machine That Changed the World”
1993	Jeff Sutherland、John Scumniotales、Jeff McKenna、Ken Schwaber らが Scrum を始める
1996	Kent Beck が XP(Extreme Programming)を考案
1997	Valve 社が Half-Life の開発に Cabal を適用
2001	アジャイル宣言

参考文献：

[1] Winston W. Royce : ”Managing the development of large software systems”, http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf, (1970)

[2] 山本 修一郎 : ”要求モデリングと誤り”, http://www.bcm.co.jp/site/youkyu/youkyu48.html , (2008)

[3] 藤井 拓 : ”アジャイルなソフトウェア開発とは”, http://www.ogis-ri.co.jp/otc/hiroba/technical/IntroASDooSquare/chapter1/IntroASDooSquareApr2005.html, (2005)

[4] Craig Larman , Dr. Victor R. Basili : ”History Of Iterative” , http://c2.com/cgi/wiki/wiki?HistoryOfIterative, (2005)
[5] Wikipedia : ”Spiral model”, http://en.wikipedia.org/wiki/Spiral_model
[6] “Manifesto for Agile Software Development”, http://www.agilemanifesto.org/, (2001)
[7] 野中 郁次郎, 竹内 弘高 : ”The New New Product Development Game”, http://apln-richmond.pbwiki.com/f/New+New+Prod+Devel+Game.pdf, (1986)
[8] 藤本 隆宏 : ”能力開発競争”, 中央公論社, (2003)
[9] Ken Birdwell : ”The Cabal: Valve’s Design Process For Creating Half-Life”, http://www.gamasutra.com/features/19991210/birdwell_pfv.htm, (1999)

第 4 章　国内のゲーム関連技術教育についての調査

4.1　変容する日本のゲーム産業と人材マネジメント

藤原　正仁
東京大学大学院

4.1.1　目的と背景

本稿の目的は、変容するゲーム産業と開発者の人材マネジメントの現状と課題について明らかにすることによって、展望を見出し、今後の処方箋を描き出すことである。

本稿において、「ゲーム産業」とは、主として据置型ゲームソフトおよび携帯型ゲームソフトを開発・販売する「家庭用ゲーム産業」を指す。しかしながら、後述するとおり、近年、家庭用ゲームをコア事業とする企業においても、パソコンゲーム、携帯電話ゲーム、オンラインゲームなどにも事業領域が拡がりつつあり、ゲーム産業は変化の過渡期にあると言える。このような変化は、劇的な技術革新、国際競争の激化、ネットワーク化、多様化する顧客ニーズなどを背景とした経営環境の変化として捉えられる。

ゲーム産業においては、最先端の技術を取り入れたゲームハード機（プラットフォーム）と従来型のパッケージソフトを中心とするゲームソフトの供給との間は相互作用的な関係にあり、いずれかの供給量が増加すると一方の供給量も増加するというネットワークの外部性が指摘されてきた。例えば、「プレイステーション」と「セガサターン」の比較分析によって実証研究を行った田中（2000）によれば、「プレイステーション」ではネットワークの外部性が働いていたが、「セガサターン」においては外部性が働いていないことが明らかにされ、新たに競争が始まるプラットホームの 1 つの世代のなかに限って外部性が働いていることを指摘している。しかしながら、近年のゲーム産業においては、如上の経営環境の変化を背景として、従来のようなプラットホーム主体のゲーム開発戦略からゲーム開発・販売会社主体のマルチプラットフォーム戦略への転換を図っており、ゲーム産業におけるビジネスは変容しつつある。加えて、ゲーム産業では、オンライン上でのコンテンツ配信やゲームユーザ層の拡大化戦略などがとられており、変容するゲーム産業の現象を捉える上で、これらの戦略はより一層重要となってきている。これは、つくれば売れる時代の終焉とともに、いかに売るかという戦略重視の時代の到来を意味している。

本稿において、「開発者」とは、プロデューサー、ディレクター 、プランナー 、プログラマー、グラフィッカー 、サウンド・クリエーター、ネットワークエンジニアなど、ゲーム開発に携わる人材を指す。開発者の活躍の舞台となるゲーム産業の特徴としては、次の 3 点が指摘できる。すなわち、第一に、ゲーム産業は、日本のコンテンツ産業のなかで最も早く産業化が進展し、唯一貿易黒字の国際的産業となっている（経済的価値の創

造)、第二に、シリアス・ゲームにみられるように、教育や医療、福祉などへの応用可能性を有している（社会的価値の創造)、第三に、高度な先端科学技術を活用したインタラクティブ性を有するエンターテインメントを提供している（技術的価値の創造）という点で、他のコンテンツ産業と比して競争優位性を持っている。ゲーム産業において、これらの価値を創造していくためには、多様な知識や能力、才能（コンピタンス）を発揮しうる開発者の協働が重要である。

また、ゲーム産業においては、制作工程の細分化とクリエイティブな専門職による分業化が進んでいる。パッケージゲームソフトの制作工程は、一般に、①ゲームソフトの企画、仕様書作成、プロトタイプの制作、②メインプログラム・原画・音楽・シナリオの作成、全データの統合、α版ゲームソフト（試作品）完成、③β版ゲームソフト（最終試作品）完成、デバック・テスティング・チューニング、マスター（完成品）の作成、④マスターの納品、レーティング、ゲームソフトの量産という過程を辿り、③の調整と修正を繰り返し行うことによって、完成度（面白さ）を追究していく。そのため、上記のクリエイティブな専門職による分業とそれらの統合が必要になる。ゲーム制作においては、タスクを時間的流れに沿っていくつかのユニットに分割し、その部分を別の実践者に任せるという「工程的分業」（福島,2001）が生じるのである。

したがって、変容するゲーム産業において多様な価値を創造していく上で最も重要な経営資源となる「人材（開発者)」のマネジメントについて検討していくことは、ゲーム産業の競争力を強化していくことにもつながるものと思われる。日本のゲーム産業が競争優位性を維持・向上させていくためには、どのような人材マネジメントが求められているのだろうか。本稿では、「人材マネジメント」を①人材の獲得、②人材の育成（キャリアディベロップメント)、③人材の評価、④人材の処遇という 4 つの観点から考察を行うこととする。これらは人材マネジメントにおける中核的な役割である。

以上の問題意識をもとに、本稿では、変容するゲーム産業と開発者の人材マネジメントの現状と課題について明らかにすることによって、展望を見出し、今後の「処方箋を描き出すための新たな眼鏡を開発する」（藤井,2005）ことを目的とする。

4.1.2　方法とデータの概要

本調査は、質問紙調査とインタビュー調査（半構造化面接）の 2 つの方法によって実施した。本稿で使用するデータは、主に次の 3 つである。第一に、社団法人コンピュータエンターテインメント協会（Computer Entertainment Supplier's Association、以下「CESA」と称する）経済産業省委託平成 18 年度サービス産業人材育成事業「ゲーム産業における開発者人材育成事業」実行委員会（以下、「平成 18 年度実行委員会」と称する）が 2006 年 12 月 22 日～2007 年 2 月 6 日にかけて、CESA 会員企業 100 社の人事部長に対して実施した「ゲーム産業における経営戦略と人材マネジメントに関する総合調

査」（質問紙調査）の有効データ 30 票（有効回収率 30%）【研究 1】、第二に、CESA 平成 18 年度実行委員会が 2006 年 12 月 7 日および 2006 年 12 月 8 日に、CESA 会員企業のうち国際的なビジネスを展開するゲーム開発・販売会社（大企業）3 社の人事部長に対して実施した「ゲーム産業における経営戦略と人材マネジメントに関する総合調査」（インタビュー調査）【研究 2】、第三に、筆者が 2007 年 11 月～2008 年 3 月に、（商業的ヒット作品を生み出している）高業績プロデューサー14 名に対して実施した「ゲーム産業におけるプロデューサーのキャリア発達に関する調査」（インタビュー調査）【研究 3】である（表 4.1-01）。

表 4.1-01　調査の方法とデータの概要

方法	No.	調査テーマ	調査対象	有効データ	調査実施時期
質問紙調査	1	ゲーム産業における経営戦略と人材マネジメントに関する総合調査	CESA 会員企業の人事部長	30 票 （回収率 30%）	2006/12/22～ 2007/2/6
インタビュー調査	2	ゲーム産業における経営戦略と人材マネジメントに関する総合調査	CESA 会員企業のうち、国際的なビジネスを展開するゲーム開発・販売会社の人事部長	3 社	2006/12/7 2006/12/8
	3	ゲーム産業におけるプロデューサーのキャリア発達に関する調査	現在の所属企業における高業績（商業的ヒット作品を生み出している）プロデューサー	14 名	2007/11～ 2008/3

表 4.1-02　回答企業の概要【研究 1】

	設立年	資本金(百万円)	平均年収	平均年齢	平均勤続年数	従業員数
度数	30	29	22	27	24	29
平均値	1,988.87	165,819	4,829,477	31.28	5.09	2,983.41
標準偏差	14.00	865,064	780,080	2.02	2.31	13.113.43
最小値	1,951.00	10	3,400,000	27.00	1.50	19.00
最大値	2,006.00	4,663,256	6,370,000	34.90	10.00	71,000.00

表 4.1-03　調査企業の概要【研究 2】

No.	設立年	資本金	平均年収	平均年齢	従業員数（連結）
X	1,978	9,090,810,000	6,370,000	33.2	1,006
Y	2,006	26,000,000,000	N/A	N/A	N/A
Z	1,980	7,825,782,540	6,170,000	32.5	3,130

表 4.1-04　調査対象者の概要【研究 3】

	年齢	性別		最終学歴			専攻		現在の所属企業規模	
		男	女	大学	専門学校	高等学校	理系	文系	大企業	中小企業
度数	14	13	1	9	4	1	7	7	7	7
平均値	38.57	－	－	－	－	－	－	－	－	－
標準偏差	2.02	－	－	－	－	－	－	－	－	－

これらの調査は、プラットホームの世代交代が行われる時期に実施されており、より実態に即したデータを提供しており、変化の現象をリアルに捉えることができる点が最大の特徴である。

なお、本稿では、出典データに関して、【研究 2】ならびに【研究 3】は明記し、【研究 1】については割愛する。調査対象の概要は、表 4.1-02、表 4.1-03、表 4.1-04 を参照されたい。

4.2　経営環境の変化

ゲーム産業を取り巻く経営環境はどのように変化し、どのような経営戦略が策定されているのであろうか。ここでは、事業領域・プラットホーム別出荷タイトル数・業績の変化、経営環境の変化に対する認識、ゲーム産業における経営戦略をもとに考察を行う。

4.2.1　事業領域

ゲーム産業の事業領域をみると、「家庭用ゲームソフトウェア事業」（90.0%）が最も多く、「モバイル・コンテンツ事業」（70.0%）、「オンラインゲーム事業」（60.0%）と続く。また、ゲーム関連事業として、「パチンコ・パチスロ事業」（36.7%）、「キャラクタービジネス事業」（30.0%）、「アミューズメント機器事業」（23.3%）、「映像事業」（23.3%）、「音楽事業」（16.7%）、「出版事業」（13.3%）、「トイホビー事業」（13.3%）、「アミューズメント施設事業」（10.0%）が挙げられ、ゲーム産業における事業領域の拡大が示されている。

事業領域を企業規模別にみると、「オンラインゲーム事業」への参入は、中小企業（33.3%）よりも大企業（86.7%）が圧倒的に多い結果を示している。また、「アミューズメント機器事業」（大企業 33.3%、中小企業 13.3%）、「映像事業」（大企業 33.3%、中小企業 13.3%）、「音楽事業」（大企業 26.7%、中小企業 6.7%）をみると、大企業は中小企業と比べてそれぞれ 20%多く参入しており、事業領域の拡大が顕著である。

各事業領域間の相関（$p < 0.01$）をみると、①「出版事業」と「トイホビー事業」（$r = .712$）、「映像事業」（$r = .711$）、「音楽事業」（$r = .614$）、②「キャラクタービジネス事

業」と「映像事業」（r =.671）で強い正の相関が観察される。これらの事業領域においては、ワンソース・マルチユース等による事業間のシナジー（相乗効果）を期待していることが窺える。

4.2.2　プラットホーム別年間出荷タイトル数

プラットホーム別に、2006 年度の年間出荷タイトル数の平均値をみると、「携帯電話機（docomo、au、SoftBank など）」（22.2 本）が最も多く、「（旧型）家庭用ゲーム機（GameCube、PlayStation2、XBOX など）」（9.0 本）、「携帯型ゲーム機（GameBoyAdvance、NintendoDS、PSP など）」（6.6 本）と続く。しかし、「（現行）家庭用ゲーム機（Wii、PlayStation3、XBOX360 など）」（1.3 本）、「PC」（1.2 本）、「業務用ゲーム機」（0.9 本）の 2006 年度の年間出荷タイトル数は、極めて少ない。これは、プラットホーム交代と携帯型ゲームソフト市場の拡大という環境変化が大きく影響している結果を示すものである。

また、企業規模別に 2006 年度の年間出荷タイトル数の平均値をみると、とくに「（旧型）家庭用ゲーム機」（大企業 14.0 本、中小企業 4.8 本）、「携帯型ゲーム機（大企業 11.5 本、中小企業 2.6 本）」となっており、大企業と中小企業の格差は顕著である（格差はそれぞれ 34.3 本、9.2 本、8.9 本）。これは、企業規模における開発力の格差を顕著に示すものである。

4.2.3　2006 年度の業績（対前年度比）

2006 年度における回答企業の業績を前年度と比較してみると、43.3%の企業で「増益」となっている。しかし、企業規模別に業績をみると、中小企業では「減益」が 0.0%に対して、大企業では 35.7%が「減益」となっており、業績の低下が著しく、なかでも、「家庭用ゲームソフトウェア事業」、「オンラインゲーム事業」、「モバイル・コンテンツ事業」（いずれも 26.7%減）の業績悪化が目立っている。

4.2.4　経営環境の変化に対する認識

如上のとおり、事業領域、年間出荷タイトル数、業績の変化にみられるように、ゲームビジネスは大きな転換期を迎えている。そこで、(1)政治動向、(2)経済動向、(3)社会動向、(4)技術革新という 4 つの観点から、経営環境の変化に対する認識について尋ねた【研究 2】。その結果、以下のとおり整理された（表 4.1-05）。

表 4.1-05　経営環境の変化に対する認識

(1)政治動向	(2)経済動向
・税制優遇への期待 ・ネットワークビジネスにおける課題解決	・ゲームユーザ層の拡大・多様化 ・ゲームビジネスからコンテンツビジネスへの発展 ・ビジネスチャンスの拡大
(3)社会動向	(4)技術革新
・ゲームに対する意識変化 ・ゲームの社会的認知向上の必要性 ・レーティング制度への対応	・ネットワークやプラットホームの進化 ・技術力強化の必要性 ・技術のエンターテインメントなどへの応用

(1)　政治動向

我が国ゲーム産業が、国際競争優位に立てるような環境整備が急務の課題である。ゲーム開発会社は、とくに、技術開発や人材育成に関する税制優遇支援策を政府に期待している。

また、ゲームのネットワーク化等によって、著作権問題やユーザー間のトラブルなど、新たな問題が生じてきている。これらの問題解決は、コンテンツの保護・流通という観点から、企業努力だけでは困難な側面もあるため、政府の支援が期待されている。

(2)　経済動向

これまでは、その時代を代表するプラットホームやコアゲーマーをターゲットにしてゲームを開発し、そのなかでヒットしたタイトルについて、ローカライズしてワールドワイドに順次展開していくことで、ある程度の収益も確保できた。非常にシンプルな戦略であった。しかし、今後は、ターゲットをより明確に定め、効率的に資金を投入して、タイトルのポートフォリオを構築していくことが求められている。また、これまでは、ゲームを事業の中心に据えてきたが、今後は、ゲームのみならず関連領域との融合により、コンテンツという切り口で、顧客ニーズを追求していく必要がある。このように、新たなビジネスモデルの構築が求められていることが、多くのゲーム開発会社で指摘されている。

近年、これまで積極的にゲームをプレイしてこなかった女性や高齢者などの層が拡大しつつあり、また、ゲーム機の機能性も高度化・多様化していることから、ゲームビジネスの裾野は飛躍的に拡大してきている。しかも、これまでゲーム開発会社で蓄積されてきたエンターテインメント・テクノロジーは、あらゆる事業領域との応用可能性を秘めており、ゲームビジネスからコンテンツビジネスへの発展も期待される。そのため、多くの企業は、ビジネスチャンスに恵まれている時期にあるとの認識に立っている。

(3)　社会動向

これまでは、“ゲーム脳”という言説等、ゲームに対する負の側面ばかりが強調されて

きたが、現在では、教育や福祉、日常生活における問題解決などにゲームが応用されることによって、ゲームへの親和性が増し、ゲームに対する社会的意識が少しずつ変化してきている。一方で、レーティング制度への対応、レーティング制度の社会的な理解形成などは、ゲーム開発会社の社会的責任という観点からもより重要となってきていている。そのため、ゲーム開発会社のみならず、業界全体で、ゲームの社会的認知をより一層向上させていくための啓蒙・広報活動（コミュニケーション戦略）が重要になってきている。

(4) 技術革新

ゲームのネットワーク化やプラットホームの進化は、これまでのビジネスモデルのみならず、開発環境にも大きな変化をもたらしている。このような変化に対応したゲーム開発において、品質向上と開発効率を両立するためには、従来とは異なる開発プロセスの構築が求められている。そのため、ゲーム開発会社は、とりわけ技術開発に対して資源を集中投下し、ツールやライブラリ、ミドルウェア、開発プロセスの強化などをより一層推進していかなければならないという認識に立っている。

4.2.5 ゲーム産業における経営戦略

経営戦略上の重視項目（5 年前と 2006 年度との比較）について尋ねた結果をみると、「開発者の育成」（これまで 33.3%→今後 76.7%）が最も重視されるとともに、「シリーズ・リメイク化ゲームの開発」（同 63.3%→60.0%）から「オリジナルゲームの開発」（同 63.3%→70.0%）へとシフト・チェンジを始めていることが分かる。また、「マルチプラットフォームに対応したゲーム開発」（同 33.3%→60.0%）、「ゲーム開発のアウトソーシング」（同 50.0%→60.0%）、「戦略的提携」（同 33.3%→56.7%）、「潜在顧客・新規顧客層の拡大」（同 26.7%→53.3%）が重視され、新たなビジネスモデルの構築と安定的な収益の確保を模索していることが窺える。

さらに、「これまで」と「今後」との相関分析を行ったところ、いくつかの項目において正の相関が観察された。とりわけ強い相関関係（P＜0.01）がみられたのは次の 4 点である。第一に、これまで「シリーズ化・リメイク化ゲームの開発」を行ってきた企業は、今後「ゲーム開発のアウトソーシング」を行う意向を示している（r =.917）。第二に、これまで「ゲーム開発のアウトソーシング」を行ってきた企業は、今後も「ゲームのアウトソーシング」を行う予定である（r =.632）。第三に、これまで「ゲームのメディアミックス展開」を行ってきた企業は、今後も「ゲームのメディアミックス展開」を行う意向を示している（r =.790）。第四に、これまで「経営多角化」を行ってきた企業は、今後も「経営多角化」を行う予定である（r =.780）。

このように、国内ゲーム市場の縮小や開発費の高騰、国際競争の激化などゲーム産業

を取り巻く環境の変化は、経営戦略にも影響を与えている。如上の経営戦略を達成するためには、とりわけビジネス環境の変化に対応した開発者の育成・確保が必要不可欠となっていることを、多くのゲーム開発企業が認識している。しかしながら、企業のコアコンピタンス（競争相手が模倣しにくい競争力の源泉となる中核的な能力）を見極め、何を企業の中に残すべきかを検討することが必要である。とりわけ人材マネジメントの観点からは、後述する人材ポートフォリオの考え方が重要となるであろう。

4.3 開発者の獲得

開発者の獲得は、どのように行われており、どのような課題があるのだろうか。開発者の確保、雇用形態別雇用量の増減、人材ポートフォリオ、採用フローの事例、採用上の課題から考察してみよう。

4.3.1 開発者の確保

2006 年度における回答企業の開発者の充足状況についてみると、「不足」（70.0%）が「適正」（23.3%）を大きく上回っており、開発者は不足の状況にある。藤原（2006）では、2006 年 1 月に、国内ゲーム会社の人材充足状況について調査しているが、本調査と同様に「不足」（66.7%）が「適正」（33.3%）を大幅超過しており、ゲーム産業は慢性的な人材不足の状況に陥っている。

これまでは、「ゲーム業界経験者の採用」（93.3%）と「新規学卒者の採用」（70.0%）により外部労働市場から人材確保を行うとともに、「内部開発者に対する OJT」（60.0%）を中心とした企業内人材育成を実施して内部労働市場から、必要な人材を確保してきた。また、「外部委託（アウトソーシング）」（73.3%）も多くの企業で実施してきた。

今後の意向についてみてみると、外部労働市場からの確保については、「新規学卒者の採用」（70.0%）をこれまで通り実施しつつも、「ゲーム業界経験者の採用」を 13.3%減らし（これまで 93.3%→今後 80.0%）、「IT 業界」や「映画業界」など関連業界より広く人材調達を行うことが窺える。一方、内部労働市場からの確保については、「内部開発者に対する OJT」を 16.7％減らし（同 60.0%→43.3％）、「キャリア開発支援」（同 6.7%→26.7%）や「Off-JT」（同 3.3%→16.7%）をそれぞれ増やし、OJT 偏重の人材育成から脱却し、多様な人材育成策を実施していくことが示されている。

4.3.2　雇用形態別雇用量の増減

雇用形態別の雇用量の増減（5 年前と 2006 年度との比較）についてみると、「正社員」、「契約社員」、「パート」、「アルバイト」、「派遣社員」のいずれも増加傾向にある（表 4.1-06）。なかでも、「正社員」の増加が最も著しく、5%以上増加している企業は 59.9%に達する。次いで 5%以上増加しているのは、「契約社員」（50.0%）、「アルバイト」（36.6%）、「派遣社員」（33.3%）となっている。しかし、「パート」を雇用している企業は少なく（56.7%）、半数にも満たない。

表 4.1-06　雇用形態別雇用量の増減

		20%以上減少	10～19%減少	5～9%減少	±5%内で増減	5～9%増加	10～19%増加	20%以上増加	該当なし(未雇用)	不明
正社員		3.3%	0.0%	0.0%	26.7%	13.3%	3.3%	43.3%	3.3%	6.7%
非正規社員	契約社員	3.3%	0.0%	0.0%	23.3%	16.7%	3.3%	30.0%	10.0%	13.3%
	パート	0.0%	0.0%	0.0%	3.3%	3.3%	0.0%	6.7%	56.7%	30.0%
	アルバイト	6.7%	0.0%	6.7%	20.0%	10.0%	3.3%	23.3%	10.0%	20.0%
	派遣社員	3.3%	0.0%	3.3%	26.7%	6.7%	3.3%	23.3%	16.7%	16.7%

4.3.3　人材ポートフォリオ

先述のとおり、ゲーム産業を取り巻く経営環境の急激な変化を背景として、正社員を中心とした雇用システムだけでは、人的資源の効率的な活用は不可能である。また、経営戦略に基づき、人材の能力を最大限に活用するために、戦略的人材マネジメント（SHRM: Strategic Human Resource Management）が求められている。そこで、戦略的人材マネジメントを行う上で、人材ポートフォリオは重要な考え方の一つとなってきている。一般に、ポートフォリオとは、金融資産などの組み合わせを意味し、その最適な組み合わせの選択が問題となる。人材ポートフォリオとは、この発想を人材マネジメントに適用し、人材を「類型化してバランス良く管理していく」手法である（内田,2006）。

人材ポートフォリオが流布する以前に、日本経営者団体連盟（現・日本経済団体連合会）（1995）による「雇用ポートフォリオ」があった。雇用ポートフォリオとは、3 つの層で人材を考えている。その 3 つとは、①長期蓄積能力活用型グループ、②高度専門能力活用型グループ、③雇用柔軟型グループである。①は期間の定めのない雇用契約、②と③は有機雇用契約であり、採用、育成、評価、処遇もグループ毎で異なる。この類型化の最大の目的は人件費の削減であり、結果として雇用柔軟型グループが急速に拡大しつつある。企業特殊的な技能の伝承という観点からみれば、このような雇用ポートフォリオの考え方は限界がある。また、企業との雇用関係にない派遣社員やインディペンデント・コントラクター、請負などのアウトソーシングなどが除外されている点も不十分である。本来は、戦略達成への貢献のあり方で人材を類型化するべきであろう。換言すれば、戦略達成

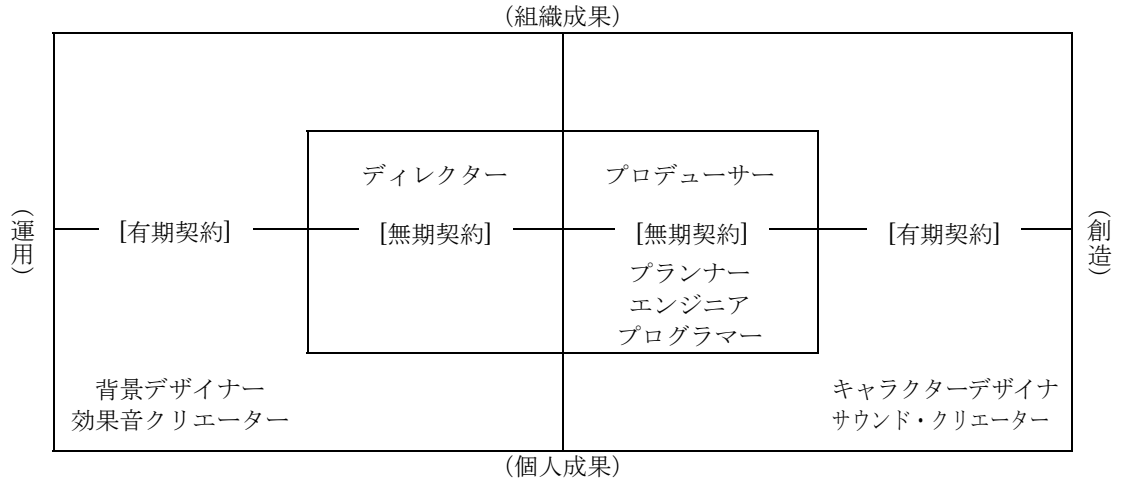
のためにどのような役割を期待されるのか。それによって人材を類型化するべきであろう。

そこで、波田野・守島・鳥取部（2000）に倣って、運用 VS 創造、個人成果 VS 組織成果という 2 軸から、非正規社員の役割について尋ねた。とりわけ「契約社員」に期待される役割は多岐にわたっているが、「個人成果に責任を持つ」ことが最も多くなっている（66.7%）。したがって、契約社員は、個人の持つ才能を十分に発揮することが期待されていると言える。また、「派遣社員」と「アルバイト」は、「運用的な業務遂行」としての役割が期待されている（それぞれ 40.0%、43.3%）。

また、開発者の活用に対する考え方を職種別にみると、「ノウハウ蓄積・長期雇用型」として活用されている職種は、「プロデューサー」（80.0%）、「ディレクター」（76.7%）、「プログラマー」（66.7%）、「ネットワークエンジニア」（70.0%）、「プランナー」（73.3%）であり、リーダーシップやマネジメント、高度な専門性が求められる職種が目立つ。一方で、「才能重視・短期雇用型」として活用されている職種は、「キャラクターデザイナ」（23.4%）、「背景デザイナー」（23.4%）、「効果音クリエーター」（23.4%）、「サウンド・クリエーター」（20.0%）であり、グラフィックスやサウンドなどのアート領域、個人の才能に依存する職種が該当している。

このように、雇用形態や職種によって、人材ポートフォリオが構築されていることが確認された（図 4.1-01）。しかし、経営戦略と人材ポートフォリオの構築の関連性については、明らかにすることができなかった。

図 4.1-01　ゲーム産業における人材ポートフォリオ



出典：波田野・守島・鳥取部（2000）p.29 のモデルをもとに筆者作成。

各象限に人材が固定されるのではなく、動態的なモデルとして捉えることに留意が必要である。

4.3.4　開発者採用フローの事例

人材採用においては、新卒採用ならびに中途採用のいずれも職種別採用が多くなっており、最終的には各部門の意思決定によって、採用者を確定している。しかし、採用のフローは、新卒採用と中途採用とで若干異なっている【研究 2】。

新卒採用の場合、まず、自社ホームページや求人ポータルサイト、大学や専門学校への求人票送付などによって、募集告知を行う。その後、エントリーシートなどをもとに書類選考を行い、合格者に対して筆記試験を課し、2 回程度の面接を実施して選抜を行っていく。その過程において、適性検査と、職種によっては課題・作品評価を行い、総合的に評価し、採用者の決定を行っている。

一方、中途採用の場合、自社ホームページや実績のある求人ポータルサイト、求人誌や新聞などによって募集を行う。その後、履歴書や職務経歴書などをもとに書類選考を行い、合格者に対して、2 回程度の面接と、職種によっては課題・作品評価を行い、総合的に評価して、採用者を決定している。

4.3.5　開発者採用上の問題点

新卒採用ならびに中途採用のいずれにおいても、「何時間もかけて面接を行い、作品をみたとしても、個人の資質や能力を十分に把握することが困難である」ことが、人材採用上の問題として指摘されている。とくに、グラフィック・デザイナーやサウンドなど、クリエイティブ・センスが問われるような職種においては、現状の採用フローで優秀な人材を見極めることが非常に難しくなっている【研究 2】。

そこで、中途採用者の募集については、近年、紹介会社の活用や知人を媒介した紹介が多くなってきており、通常の採用方法で採用するよりも、ある程度スキルが保証された人材を確保することが可能となっている。また、新卒採用においても、研修生として試用期間を設け、優秀な人材をスクリーニングして採用する方法を採り入れている企業もある。しかし、インターンシップは受入側の負担の問題により、積極的に実施されていない場合が多い。具体的には、開発現場が多忙のため、インターンを受け入れる余力が残っていないことが挙げられている。他方で、インターンを受け入れても、その期間が短いため、指導して多少仕事を覚えた頃に、インターンシップが終了してしまい、教育コストの回収ができないという欠点も指摘されている。

これまで中途採用者のほとんどがゲーム業界経験者であったが、ゲームの高度な映像表現技術やオンライン技術などの要請によって、ハリウッドなどの海外映像業界経験者や IT 業界などからも人材確保を行うようになってきている。ゲームビジネスの拡大に伴って、今後、多様な業界経験者、高度な専門的な知識やスキルを有する人材の確保がより一層求められている。

4.4　開発者の育成とキャリアディベロップメント

開発者の育成は、どのように行われており、どのような課題があるのだろうか。開発者の育成に対する考え方、キャリアディベロップメント、育成上の問題点をもとに考察してみよう。

4.4.1　開発者の育成に対する考え方

開発者の育成に対する考え方について尋ねてみると（表 4.1-07）、「開発者のキャリア開発は、OJT が中心である」（82.7%）、「キャリア開発の諸施策は開発現場主導で行われている」（51.7%）、「本社人事部は OJT の計画や実行には全く関与していない」（48.2%）という傾向がみられる。しかし、「開発者のキャリア開発：企業責任 VS 個人責任」（48.3%）、「教育投資：すべての開発者 VS 特定の開発者」（41.4%）は、どちらともいえないという回答が最も多くなっている。「キャリア開発：プロフェッショナル育成 VS ゼネラリスト育成」については、どちらともいえない（48.3%）と「プロフェッショナルとしてのキャリア開発を重視している」（48.2%）が拮抗している。

表 4.1-07　開発者の育成に対する考え方

A	Aに近い	ややAに近い	どちらともいえない	ややBに近い	Bに近い	B
開発者のキャリア開発は OJT が中心である	51.7%	31.0%	6.9%	10.3%	0.0%	開発者のキャリア開発は Off-JT の割合が大きい
開発者のキャリア開発は会社が責任を持って支援を行う	13.8%	17.2%	48.3%	17.2%	3.4%	開発者のキャリア開発は個人の自己責任に任せている
教育投資は全ての開発者に対して公平に行っている	10.3%	24.1%	41.4%	24.1%	0.0%	教育投資は特定の開発者に対して重点的に行っている
プロフェッショナルとしてのキャリア開発を重視している	17.2%	31.0%	48.3%	0.0%	3.4%	ゼネラリストとしてのキャリア開発を重視している
本社人事部は OJT の計画や実行に関与している	3.4%	27.6%	20.7%	31.0%	17.2%	本社人事部は OJT の計画や実行には全く関与していない
キャリア開発の諸施策は本社人事部主導で行っている	3.4%	20.7%	24.1%	24.1%	27.6%	キャリア開発の諸施策は開発現場主導で行われている

4.4.2　開発者のキャリアディベロップメント

(1)　開発者の能力開発

開発者の能力開発（OJT、Off-JT、キャリア開発支援）の実施について、これまでと今後の意向を尋ねた結果をみると、OJT は、今後「十分実施」する企業が 16.7%増加している（これまで 33.3%→今後 50.0%）。「ある程度実施」する企業を含めると、全体として 93.3%が今後 OJT を実施する予定である。また、Off-JT は、今後「ある程度実施」する企業が 30.0%増加し（同 43.3%→73.3%）、「十分実施」する企業を含めると、全体として 76.6%の企業が今後 Off-JT を実施する意向を示している。そして、キャリア開発支援は、今後「ある程度実施」する企業が 33.4%増加し（同 33.3%→66.7%）、「十分実施」する企業を含めると、全体として 70.0%の企業が今後キャリア開発支援を実施する予定である。

このように、開発者の能力開発は、これまで OJT を中心に行われてきたが、今後は Off-JT やキャリア開発支援も重視し、多様な能力開発が組み合わされて実施されていくことが示唆される。

(2)　OJT の内実―成長を促す経験

先述の通り、開発者の能力開発は、OJT（On the Job Training）を重視するという回答が最も多く、93.3%（十分実施、ある程度実施）を示していた。しかしながら、「本社人事部は OJT の計画や実行には全く関与していない」（48.2%）が多く、「キャリア開発の諸施策は開発現場主導で行われている」（51.7%）。OJT は、本社人事部は非関与で、現場任せの状況にある。OJT とは便利なことばで、単なる現場での仕事経験として放任を覆い隠している可能性もある。

このような問題を正面から取り上げたのが MacCall（1998）である。彼は、リーダーシップ開発の観点から、成長を促す経験を 4 つのカテゴリーに分類し、OJT の内実を可視化した。それは、課題、他の人とのつながり（特に自分よりも高い地位の人）、修羅場と失敗、その他である。そして、それらのカテゴリーに内在される 16 種類の経験についても明示化した。第一に「課題」については、初期の仕事経験、最初の管理経験、ゼロからのスタート、立て直し、プロジェクト・タスクフォース、視野の変化、ラインからスタッフへの異動、第二に「他の人とのつながり」については、ロールモデル、価値観、第三に「修羅場」については、事業の失敗とミス、降格・昇進を逃す・惨めな仕事、部下の業績の問題、既定路線からの逸脱、個人的なトラウマ、第四に「その他」については、コースワーク（公式の研修プログラム）、個人的な問題（仕事以外の経験）である。

そこで、上記の研究成果を参考にしつつ、ゲーム産業におけるプロデューサーが回顧的に意味づけている成長を促す経験について分析した結果、課題、人脈、修羅場、日常生活という 4 つのカテゴリーが抽出された【研究 3】。

第一に「課題」については、先例のない業務へのチャレンジを挙げる者が最も多く、その結果、達成感を獲得している。ゲーム産業は、ハイリスク、ハイリターンのビジネスのため、ヒットを続けているタイトルやジャンルをシリーズとしてリリースあるいはリメイクする傾向にある（米倉・生稲,2005）。そのため、新しいタイトルやジャンルをつくり出すことはリスクを負うことにもなる。しかしながら、未開拓のゲームを開発していくことにより、新しいアイデアや技術の応用方法などを学習することにもつながるのである。

――「なんでも初めてやることはおもろいってことですね。このジャンル、このシリーズが受けているから同じ物を作りましょうっていうよりも、あまり誰もやらない、やられていないジャンルをやろうと思ったら、作り手としては面白いんじゃないかなとすごく学べました。」（37 歳、男性、専門学校卒業）

また、ゲームのタイトル開発を一本任せられて仕事の幅が広がったり、先述のとおり人事異動などによって役割が変化することによって、視野が広くなったという者も 4 名存在する。視野の変化は、課題を克服していくなかで、つまり行為によって獲得されるものであり、極めて自覚的な概念である。自己の成長において最も重要な経験と言えるであろう。

――「（タイトルを）一本まあ、あの仕上げたことによって、こう、何か仕事の幅が広がったような気がします。」（38 歳、男性、大学卒業）

その他、他社や異業種との協働経験や、海外での協働経験は、仕事の進め方の違いを学習し、後の仕事に活かされている。企業組織内部により人材育成がなされることは事実であるが、企業組織外部との協働経験は、社内の慣習や慣行、言語や思考などの差異を確認することができ、人材育成のみならず、企業組織の変革にも寄与するものである。また、海外での協働経験は、ゲーム産業を国際的なビジネスとして発展させていく上で有益な示唆を与えている。

――「ゲーム業界は言葉が全然違うんですね、会社によって。他の会社さんとこう話をすると、時々お互いに分からないっていう。（中略）でも映画系とか向こう（ハリウッド）では当然それはもう当たり前のようにやって。どこに行ってもだから、みんな仕事はできるんですよね。」（40 歳、男性、専門学校卒業）

短期間における成果達成という経験は、小橋（1996）が日本のゲーム産業におけるゲーム開発工程の特徴を「自己犠牲的プロセスモデル」と名付けたように、短期間に過剰な負担を強いられるものであり、自己犠牲的側面が強調されている。自己犠牲的プロセスモデルとは、曖昧なイメージの交換によって、難易度、ゲームバランス、論理と曖昧さのバ

ランス、操作性というゲームの面白さを左右するような実質的な構成要素が組み立てられていくが、心理的バイアスが介在することにより、イメージの修正、発展な統一（完成）がなされるという、日本のゲーム産業における開発工程の特徴を示す。実際、「かなりつらかった」と過酷な状況を物語っているが、プロデューサーとしてよい経験となったと回顧している。

――「30 人ぐらいのチームで、30 タイトルぐらいを 1 年ちょっとくらいでラインナップとして揃えろって。かなり複数の会社を一度に使うっていう経験はなかったんですよ。制作委託を出しながら、その中の部分委託を他へやらしてっていう作り方のコーディネートまでやりつつ、内部の制作もありつつ。そういうやり方はやったことなかったんで、プロデューサー的には、いい経験になりましたね。まあかなりつらかったですけど。」（42 歳、男性、専門学校卒業）

第二に、「人脈」については、主体的に技能を習得することによって周囲からも承認を得られるようになり、一人前として認められるという相互作用がみられる。周囲から認められることにより、言語的説得という自己効力感の獲得にもつながっている。

――「自分自身がグラフィックをやって勉強しながら、進めていって、結果的にまあ当時いたスタッフのメンバーにも後から追いかけた状態ではあったんですけど、内部から認めてもらえるような状態になって、（中略）そのステージとか背景のグラフィックの大半は私が 1 人で作った。」（37 歳、男性、大学卒業）

また、ロールモデルとなる人との出会いがキャリアに影響を与えており、弱い紐帯よりも強い紐帯を持つ者が多い結果となっている。強い紐帯を具体的に挙げれば、創業者、社長、ゲームデザイナーの上司、プログラマーの同僚のように、自分より目上の仕事上の関係者が多くなっている。一方、弱い紐帯を具体的に挙げれば、他社のゲームデザイナー、大学時代のサークルの先輩、前職の OB が挙げられ、仕事以外の者からもキャリアに影響を与えられている。いずれの者も反面教師的に学ぶというよりは、尊敬の念を抱き模範的に学習しているという点が特徴として挙げられる。

――「ゲーム制作にはすごい関わってきはるんですよ。大体大きな方針だけ言って、『やっとけ』とか言うんじゃなくて、文字チェックとかまでしはるんですよ。細かいとこの修正とかも自分でやりはって。シナリオとかも全部自分で書きはって。印象に残っているのは、知識が広いというか深い。インプットの量、半端じゃないなと思いましたけどね。すごいろいろ本も読んではるし。」（35 歳、男性、大学卒業）

――「演劇時代で教わった先輩っていうのが一番大きかったと思いますね。僕に例えば演出を教えてくれたり、人を喜ばせたりするにはここが重要だとか、そういうこと教えてくれた先輩の方々がすごい影

響与えましたね。」（42 歳男性、大学卒業）

第三に、「修羅場」については、失敗経験と追い詰められるような経験が挙げられる。失敗は、経営環境の厳しい企業においては制約されるものであるが、同時に、絶好の学習の機会でもある。ゲーム開発の現場においては、「自己犠牲的プロセスモデル」が示唆するように、「つくってみたら、企画時に想定していたものと全然違う、面白さの要素が全然違う」ということはよくあることである。また、ゲーム産業では同時並行的にラインを走らせて、いくつかのゲームを開発して、リスクヘッジが行われている。したがって、日本のゲーム産業においては、福島・萩野（2005）が指摘するように、ある程度「失敗が許容される環境」にあるように思われる。また、追い詰められるような経験とは、倒産寸前の危機に遭遇した際に、それを立て直すために、あらゆる問題を同時並行的に解決しなければならない経験である。そのような場面では、短期間の間に、ヒト・カネ・モノ・情報などの経営資源を有効に活用しなければならない。したがって、追い詰められるような経験は、失敗経験と同様に、短期に学習が促される、成長を促す経験と言えることができる。

――「自分がディレクターしてプロデューサーして関わった大作がボツになって、1 年半まで開発期間がかかったかなっていうぐらいですかね。何かプロジェクトを進める時には、明確なビジョンを持ってから進めないとだめっていうことを学びました。」（35 歳、男性、大学卒業）

第四に、「日常生活」については、遊び経験、家族とのふれあい、学生時代に得た経験が挙げられる。遊び経験とは、ゲーム、漫画、アニメ、映画などの遊びが日常生活に埋め込まれていることを指す。これらは、仕事としてではなく、あくまでも遊びとして意味づけられていることが特筆すべき点である。ゲーム産業への関心醸成という初心が一層高められ、消費者の立場として、遊び経験を通じて実際のゲーム開発に応用されているのである。

――「タイトルっていうのは奇跡の連続で物が出来上がっているわけですから、言い換えると我々当たり前のように多分奇跡をコントロールできないとだめだと思ってる。ですから、奇跡を起こす方法、逆転の方法論っていうのは、多分ちっちゃい時から読んで育ってる漫画に多分教えてもらったんじゃないかなと思うんです。日常よりも非日常が描かれてることの方が多いじゃないですか、漫画って。そんなばかなって言うようなことこそが多分、逆転の発想だと思うんですよ。できるわけないじゃんっていうのが漫画じゃないですか。」（37 歳、男性、大学卒業）

また、家族とのふれあいとは、具体的には子どもがいるからこそ、子どもに与えたいもの、感じて欲しいもの、次世代に伝えたいものが浮かんでくるという概念である。主要な

消費者である子どもの目線に立ってゲームをつくって、一緒に遊びたいという願望は、子どもを持つゲーム開発者の極みではなかろうか。ゲーム産業におけるキャリア発達を捉える上では、仕事と同様に家族という存在は大きなものと思われる。

――「わたしは今子どもが 2 人いるんですけれども、子どもが 2 人いるからこそ、やっぱり子どもに与えたいもの、子どもに感じて欲しいもの、次世代に伝えたいものっていうのが浮かんでくるんですよね。一般的にはゲームがすごい高機能化、高度化してなかなかおじいちゃんおばあちゃんができるようなゲームってないですよね。でも子どもに目を向けると子どもが楽しめるものって、裏を返せば全員楽しめるんですよ。例えば、ショッピングセンターに行った時に子どもを連れているけれども、何か子どもと親が一緒にね、触れ合って遊べるようなものがあったらもっといいのにと思うわけですね。そういうものが何かないかしらっていうのがその開発の原点になる場合がよくあります。子どもがいなかったらそもそもそういう気分は生まれてこないじゃないですか。」（42 歳、男性、大学卒業）

学生時代に得た知識の有効性は、学校教育の機能的側面と自己の認識の側面による 2 軸によって 4 つの象限から捉えることができる（表 4.1-08）。

表 4.1-08　学校教育の知識の有効性：四つの見方

認識 ＼ 機能	有効である(+)	無効である(−)
有効だと思う(+)	実質説	陰謀説
無効だと思う(−)	隠蔽説	空洞説

出所：矢野（2001）p.114

例えば、学校教育の職業的意義を主観的に評価している（学校教育の知識が機能・認識ともに有効だとする「実質説」を支持する）者は、学校で学んだ知識をゲーム産業におけるプロデュースにも応用ができているという認識を持っている。一般に、実質説を支持するのは、専門学校とされているが、学歴によって差はみられなかった。とりわけ数学の知識は、プログラミングだけではなく、マーケティングや論理的思考による説得など、プロデュース業務においても役に立っている。また、視角情報伝達や図面設計は、イメージを実際の絵に描くだけでなく、考えを視角化して相手に伝えるというコミュニケーションの場においても応用されている。このように、実質説においては、学校教育で学んだ知識が直接的に役に立っているということが特筆すべき点として挙げられる。

――「絵を描いて表現するイコール、ユーザーさんには口で説明することはできませんので、結局見て、完成していただいたものを見て、そのまますぐ直感で分かっていただかないといけないので、そういう意味ではその大学で学んだ情報伝達というか、ゲーム画面を構築する上では非常に役に立ちました。」（38 歳、男性、大学卒業）

以上のとおり、プロデューサーの成長を促す経験について、課題、人脈、修羅場、日常生活という 4 つのカテゴリーと、先例のない業務へのチャレンジ、視野の変化、他社・異業種・海外との協働経験、短期間における成果達成、主体的な技能形成と周囲からの承認、強い紐帯を持つロールモデルとの出会いとキャリアへの影響、弱い紐帯を持つロールモデルとの出会いとキャリアへの影響、失敗経験、追い詰められるような経験、遊び経験、家族とのふれあい、学生時代に得た経験という 12 種類の経験が抽出された。

(3) Off-JT の具体的な取り組み

Off-JT の具体的な取り組みについてみると、「国内研修機関・研究会への派遣」が最も多く（66.7%）、その内訳の中心は「CEDEC（CESA Game Developers Conference）」（60.0%）への派遣となっている。CEDEC は、国内最大規模のゲーム開発者の人材交流や情報交換の場となっており、多くの期待が寄せられていることが示唆される。また、66.7%の企業が「専門誌の閲覧」機会を提供し、とくに「週刊ファミ通」によってゲーム業界の動向や最新のゲームソフトに関する情報提供に資している（66.7%）。その他、「新入社員研修」と「社内勉強会」（いずれも 53.3%）が中心に行われている。海外研修については、年々規模を拡大している「GDC（Game Developers Conference）」への派遣が23.3%の企業で行われている。

しかし、企業規模別にみると、中小企業よりも大企業において、Off-JT の実施率が高い。例えば、「GDC」への派遣は、中小企業が 6.7%に対して、大企業では 42.9%であり、大きな乖離がみられる。同様に、「SIGGRAPH」への派遣についても、21.9 ポイントの格差がある（大企業 42.9%、中小企業 6.7%）。また、「語学研修（英語）」は、大企業が35.7%に対して、中小企業では全く実施されていない（0.0%）状況である。その他、「新入社員研修」（大企業 71.4%、中小企業 40.0%）、「職種別研修」（同 35.7%、13.3%）、「階層別研修」（同 42.9%、26.7%）においても、大企業と比較すると中小企業では人事制度として定着していない企業が多い。

(4) キャリア開発支援の具体的な取り組み

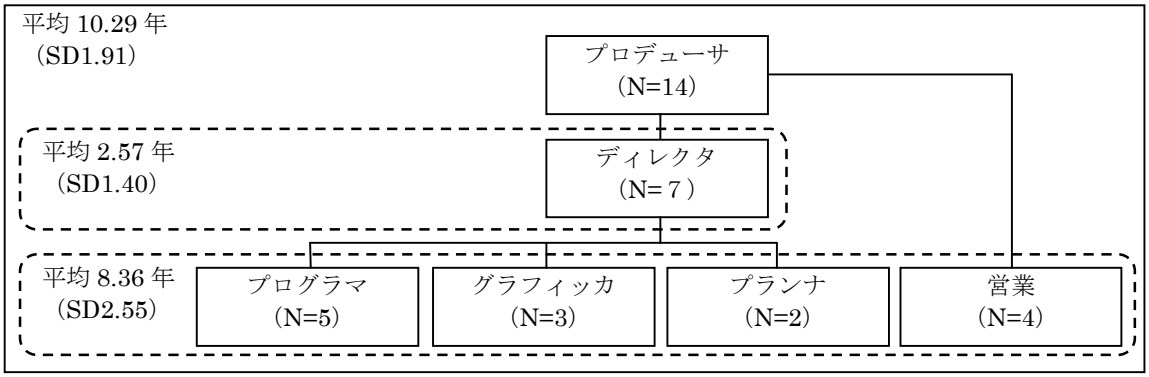
キャリア開発支援についてみると、「特別な支援はしていない」企業が最も多い（36.7%）が、「社内公募制の導入」（26.7%）や「自己申告制度の導入」（23.3%）など、開発者の主体的（自律的）なキャリア開発を支援する取り組みもみられる。しかし、「キャリア開発に関する情報提供」の取り組みは最も少ない（10.0%）。

(5) キャリアパターン

プロデューサーに至るまでのキャリアパターンを職種の観点からみると、①プログラマー、グラフィッカー、プランナーという開発職経由のキャリア（10 名）と、②営業職経由のキャリア（4 名）という、2 軸のキャリアパターンが析出された【研究 3】。

まず、開発職経由のキャリアパスをみると、プログラマー、グラフィッカー、プランナーという専門職として初期キャリアを形成した後、7 名がディレクターを経て、プロデューサーに到達している点が特徴として挙げられる（図 4.1-02）。また、10 名中 9 名が学校卒業後に、ゲーム産業に参入し、その後もゲーム産業に属していることから、企業組織内部においてプロデューサー育成が行われていることが示唆される。開発職経由のプロデューサーは、すべての者がゲーム開発の周辺から中核にまで携わった経験を持っており、ゲーム開発の過程を理解している。ゲーム開発の現場を理解していることは、とくに現場の状況を把握してスケジュールを管理するなど、プロジェクト全体を統括していく上で重要である。さらに、ディレクター経験者は、俯瞰的立場から現場を指揮する経験を有するため、ゲーム開発者のそれぞれの役割をより深く理解した上で、プロジェクト全体を統括することができるのである。

図 4.1-02　ゲーム産業参入からプロデューサーに至るまでのキャリアパターン



次に、営業職経由のキャリアパスをみると、営業職として初期キャリアを形成した後、ディレクターを経ずにプロデューサーに到達している。営業職経由のプロデューサーは、自らがゲーム開発の中核的な経験を持たないが、営業職において、プロデューサーに必要とされるコミュニケーション能力や管理的な能力を獲得している。そのため、ゲーム開発の経験を持たなくても、円滑な人間関係を構築しながら、自らの役割を現場に埋め込み、プロデューサーを担っていることが示唆される。

(6)　キャリアラダー

図 4.1-03 は、等級と職分のマトリクスで構成されるキャリアラダーの事例を示したものである【研究 2】。等級と職分のマトリクス、つまり、この事例は、職能資格制度（担当職務の内容や社内における相対的地位ではなく、職務を遂行するための能力［職務遂行能力］に応じて従業員の格付けを行う制度）と職務等級制度（職務に対して難易度や重要度をもとに格付けを行う制度）との折衷・混合型の人事制度と捉えることができる。平野

（2006）に従えば、J（日本）型の組織モードが A（米国）型の組織モードを取り入れながら、進化 J 型に移行した人事管理とみることもできる。

等級は、最上位 6 等級～最下位 1 等級の 6 段階により構成されている。各等級によって、その責任範囲は異なる。「部長クラス」は組織全体の利益責任、「課長クラス」は部下の監督責任、「チームリーダークラス」は各職務におけるリーダーとしての責任を有する。「一般社員クラス」は下積み期間として位置づけられている。

次に、職分についてみてみよう。4～1 等級については、「コアクリエーター」（プログラマー、プランナー）、「スペシャルクリエーター」（グラフィックデザイナ、サウンド）、「スタッフ」（人事、財務、広報、営業担当など）の 3 類型を、6～5 等級では、「エキスパート」（専門能力を発揮する人材）と「マネジメント」（組織目標を達成する人材）の 2 類型を定義している。さらに、各職分の各等級ごとに職務要件が定義されており、グループウェア上で社員がいつでも閲覧できるようになっている。そのため、社員は、職務要件を指針として、キャリア目標を計画することが可能となっている。

図 4.1-03　キャリアラダーの事例

<table>
<tr><td>6</td><td>部長クラス</td><td>利益責任</td><td colspan="2">エキスパート
（専門能力を発揮）</td><td>マネージメント
（組織目標達成）</td></tr>
<tr><td>5</td><td>課長クラス</td><td rowspan="2">監督責任</td><td rowspan="4">コア
クリエーター</td><td rowspan="4">スペシャル
クリエーター</td><td rowspan="4">スタッフ</td></tr>
<tr><td>4</td><td>主任クラス</td></tr>
<tr><td>3</td><td>チームリーダークラス</td><td>職務責任</td></tr>
<tr><td>2</td><td rowspan="2">一般社員クラス</td><td rowspan="2">下積み期間</td></tr>
<tr><td>1</td></tr>
</table>

4.4.3　開発者の育成上の問題点

開発者の採用における問題としては、「人材要件に合致するような人材が集まらない」ことが最も多く、80.0%にも達している。次いで、「求人活動を行っても人材が集まらない」ことが顕著になっている（40.0%）。開発者の育成に関する問題点としては、「教育する人材が社内に不足している」（60.0%）が最も多く、次いで、「育成する時間がない」（40.0%）と続いている。

企業規模別にみると、中小企業ではとりわけ「人材要件に合致するような人材が集まらない」（86.7%）、「教育する人材が社内に不足している」（78.6%）が顕著であり、開発者の採用や育成に関する問題が多岐にわたっている。

4.5　開発者の評価

開発者の評価は、どのように行われており、どのような課題があるのだろうか。評価に対する考え方、評価制度の事例、評価上の問題点をもとに考察してみよう。

4.5.1　開発者の評価に対する考え方

開発者の評価に対する考え方について尋ねた項目についてみると（表 4.1-09）、大半の企業において、「過去の成功や実績に基づいた意見や提案が評価される」（55.5%）傾向にある。「評価基準：公開 VS 非公開」については、どちらともいえないが最も多くなっている（48.3%）が、「評価結果・評価理由を本人に十分開示している」（75.8%）状況にある。しかし、「評価に関する苦情処理の仕組み：あり VS なし」はどちらともいえないが最も多くなっている（41.4%）。評価の透明性は担保されているが、客観性、納得性の確保については課題が残されている。

表 4.1-09　開発者の評価に対する考え方

A	Aに近い	ややAに近い	どちらともいえない	ややBに近い	Bに近い	B
過去の成功や実績に基づいた意見や提案が評価される	18.5%	37.0%	22.2%	14.8%	7.4%	過去の成功や実績にとらわれない意見や提案が評価される
評価基準を開発者に公開している	24.1%	13.8%	48.3%	6.9%	6.9%	評価基準を開発者に公開していない
評価結果・評価理由を本人に十分開示している	24.1%	51.7%	13.8%	6.9%	3.4%	評価結果・評価理由を本人に開示していない
評価に関する苦情処理の仕組みがある	13.8%	20.7%	41.4%	20.7%	3.4%	評価に関する苦情処理の仕組みがない

4.5.2　開発者の評価制度の事例

インタビュー調査によって明らかにされた開発者の評価制度は以下のとおりである【研究 2】。

X 社では、目標管理制度（MBO: Management By Objectives）による人事評価制度が整備されている。ここでの評価対象は、「能力」と「業績」の 2 点となっている。能力は、職務要件で定義された能力を発揮できたかどうかについて、また、業績は、直属の上司と面談を行い、設定した目標を達成できたかどうかについて評価している。一次評価は、課長クラスが行い、二次評価は、部長クラスが行っており、2 段階の多段階評価を経た後、人事部において評価決定を行っている。評価結果は、8 段階の格差で示される。その後、直属の上司との面談によって評価結果をフィードバックし、今後の課題解決などについてのアドバイスを受けている。万が一、評価結果に納得がいかない場合は、人事部で個別に対応をしている。

Z 社では、評価対象は、「能力」と「チーム力」、「勤務姿勢」の 3 点となっている。評価基準はフィードバックの際に、口頭で説明されている。評価のフローは、現場直属のリーダーが査定する一次評価、職種別リーダーが査定する二次評価、職種間リーダーが査定する三次評価という 3 段階の多段階評価を経て、開発責任者が最終決定している。評価結果は、5 段階の格差で提示される。その後、面談によって、評価結果を本人にフィードバックし、20 数段階ある開発スキルの達成度合いなどについて説明が行われる。

4.5.3 開発者の評価上の問題点

開発タイトルの成果による評価には、いくつかの問題点が指摘されている。第一に、「短期的な評価と長期的な人材の育成が必ずしも合致しないこと」、第二に、「開発者個々人が自らプロジェクトを選べないこと」、第三に、「タイトル間の格差が拡大しているため、ネガティブな面が強まっていること」である。このように、とくに評価の納得性という点で大きな問題を抱えている。

また、「評価自体が絶対評価でありながら、予算や人事的な観点から相対評価にならざるをえない側面がある」、「一人の評価に対して、多数の被評価者が集中している（評価者が不足している)」ことから、評価の客観性の担保も課題となっている。

さらに、「会社規模が現状維持の場合、新しいゲームソフトのメイン的な立場での仕事など、面白い仕事を与えるという処遇が出来ない」ことも指摘されており、OJT を中心とした人材育成策は、脆弱な状況に置かれている。

上記の質問紙調査と同様に、インタビュー調査においても以下のとおり同じような問題点が指摘されている【研究 2】。

(1) 評価期間の長期化

開発期間の長期化やオンライン化によるビジネスモデルの変化に伴って、これまでと比べて、プロジェクトに対する評価期間が長期化したり、あらゆる指標での数値目標の設定が必要となってきている。そのため、年度単位での目標設定やプロジェクト評価を行うことが難しくなっており、金銭的処遇のあり方にも影響を及ぼしている。

(2) 評価制度の未整備

評価制度の未整備により、職務要件や評価の方法が明確に規定されていない企業もある。そのため、評価者の主観に依存する可能性が高くなっており、評価の正当性を担保できない懸念がある。

4.6　開発者の処遇

開発者の処遇は、どのように行われており、どのような課題があるのだろうか。開発者の処遇に対する考え方、賃金制度改革、処遇制度の事例、処遇上の問題点をもとに考察してみよう。

4.6.1　開発者の処遇に対する考え方

開発者の処遇に対する考え方について尋ねた項目についてみると（表 4.1-10)、多くの企業において、「給与制度は、成果や業績的な要素が大きく」なっていることが分かる（86.2%)。しかし、「評価結果に基づく降格人事は行われていない」（51.7%)、また、「評価結果に基づく給与ダウンは発生していない」（48.3%)。これは、開発者個人に対する評価のみならず、開発チームやプロジェクト単位での評価も行われていることから、開発者の外発的動機の低下を招かないような措置であろう。

そして、「高業績者を報奨金以外の給与以外の報酬で処遇している」（64.3%)、「高業績者を昇進や昇格等のポストで処遇している」（55.1%）ことから、高業績開発者の内発的動機の維持または向上に向けた人事施策がとられていることが示唆される。

表 4.1-10　開発者の処遇に対する考え方

A	Aに近い	ややAに近い	どちらともいえない	ややBに近い	Bに近い	B
給与制度は成果や業績的な要素が大きい	51.7%	34.5%	13.8%	0.0%	0.0%	給与制度は年功的な要素が大きい
高業績者を報奨金等の給与以外の報酬で処遇している	37.9%	17.2%	13.8%	24.1%	6.9%	高業績者を報奨金等の給与以外の報酬で処遇していない
高業績者を昇進や昇格等のポストで処遇している	17.9%	46.4%	17.9%	10.7%	7.1%	高業績者を昇進や昇格等のポストで処遇していない
評価結果に基づく給与ダウンは頻繁に発生している	3.4%	20.7%	27.6%	13.8%	34.5%	評価結果に基づく給与ダウンは発生していない
評価結果に基づく降格人事は頻繁に行われている	0.0%	17.2%	31.0%	20.7%	31.0%	評価結果に基づく降格人事は行われていない

4.6.2　開発者の賃金制度改革

基本給については、これまで「年俸制の導入」（56.7%）、「年齢給の縮小・廃止」（50.0%）、「職務給・役割給の導入」（46.7%）、「業績給・成果給の導入」（43.3%）、「能力給部分の拡大」（40.0%）などの賃金制度改革が行われている。年功的な部分は縮小ないし廃止の方向に向かっており、業績に準じた仕事ベースの給与体系への変更の傾向がみられる。

賞与・一時金については、これまで「企業業績と賞与の連動強化」（46.7%）、「個人業績と連動する部分の拡大」（33.3%）、「開発タイトル報奨金制度の導入」（30.0%）、「ストックオプションの導入」（30.0%）など、企業業績と個人業績とに連動して決定している。そして、今後の方針としては、引き続き、企業業績や個人業績との連動を強めるとともに、「開発チーム別業績賞与の導入」（26.7%）、「開発タイトル報奨金の格差拡大」（16.7%）など、開発チームやプロジェクトの成績との連動が検討されるようになっている。

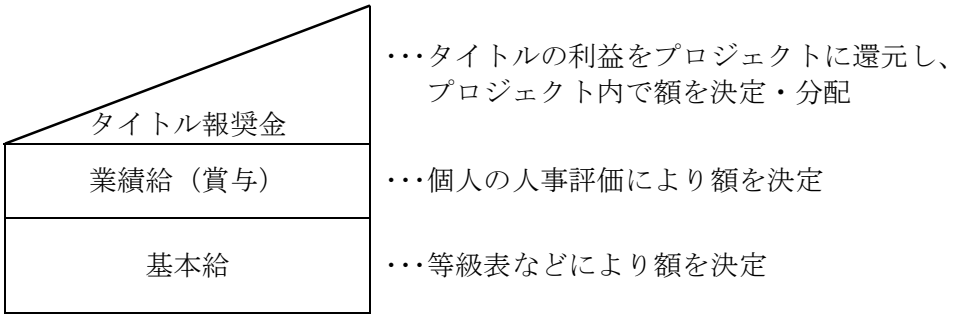
4.6.3　開発者の処遇制度の事例

インタビュー調査によって明らかにされた開発者の処遇制度の事例について、以下のとおり概観してみよう【研究 2】。

(1)　金銭的処遇

給与体系は、単純には、①基本給、②業績給（賞与）、③タイトル報奨金の 3 つで構成されている（図 4.1-04）。基本給は、等級表などで規定された基準額に基づいて支給されている。また、業績給（賞与）は、人事評価の結果に基づいて支給されている。そして、タイトル報奨金は、プロジェクトにおける開発タイトルの利益に応じて支給されている。開発タイトルの売上が好調の場合、タイトル報奨金だけで数千万円支給されることもある。

図 4.1-04　ゲーム会社の給与体系



(2) 褒賞制度

ヒットタイトルを開発したプロジェクトに対しては、賞状と記念品が授与され、祝賀パーティが開催されている。

(3) 魅力的な仕事の提供

成果目標達成者に対して、やりたい仕事や役割を与えたり、所属したいプロジェクトへの人事異動を行っている。

(4) リフレッシュ休暇

プロジェクト終了後、2 週間～1 ヶ月間のリフレッシュ休暇を与えている。

4.6.4 開発者の処遇上の問題点

インタビュー調査によって明らかにされた開発者の処遇上の問題について、以下に言及する【研究 2】。

(1) プロジェクト間の格差拡大と開発者間の公平性の確保

タイトル報奨金だけで、数千万円もの格差がつく場合があり、ヒットタイトルを開発している部門とそれ以外の部門との格差が拡大している。そのため、ヒットタイトルを抱えるプロジェクトに所属する高業績者と低業績者、ヒットタイトルを抱えていないプロジェクトに所属している社員との間で不公平感が存在する。また、開発期間の長期化やオンライン化によるビジネスモデルの変化に伴って、これまでのような年度単位での金銭的処遇が難しくなってきている。

(2) 高業績者の処遇

高業績者ほど責任が重く、多忙な勤務状況となっている。そのため、高業績者は、プロジェクトが終了しても、リフレッシュ休暇を取得することが難しく、内発的動機の維持、向上のみならず、人材育成や雇用管理においても大きな問題を抱えている。

4.7 要約と結論

如上のとおり、本稿では、ゲーム産業における人材マネジメントの現状と課題、具体的には、①開発者の獲得、②開発者の育成（キャリアディベロップメント）、③開発者の評価、④開発者の処遇について考察を行ってきた。以下に、改めて要約をまとめ、結論を提示し、最後に、残された課題について言及する。

4.7.1 要約

ゲーム産業を取り巻く環境は、政治・経済・社会・技術的観点からみて変容してきており、経営戦略や人材マネジメントに大きく影響を与えている。このようなゲーム産業の転換期における経営戦略としては、経営資源の根幹となる「開発者の育成」が最も重視されている。ゲーム産業における競争優位の源泉は、ヒト・モノ・カネ・情報という経営資源のなかでもヒト（開発者）であるという認識が高まっている。また、コアゲーマー層をターゲットにしたゲーム開発の時代から、潜在顧客や新規顧客層をターゲットにしたゲーム開発の時代に突入し、経営戦略上、市場を的確に把握しうるマーケティングも重視されるようになってきている。加えて、ゲーム産業では、これまでシリーズ化またはリメイク化されたゲーム開発によって安定的な収益を確保してきたが、今後はオリジナルゲームを開発し、マルチプラットフォーム化戦略を重視する、新たなビジネスモデルの開発傾向がみられる。

開発者の獲得は、これまでゲーム業界経験者や新規学卒者の採用を中心に行われてきたが、今後は IT 業界や映画業界等関連業界より多様な人材を確保していく意向が示されている。しかし、人材が集まらない、人材が定着しない等、慢性的な人手不足の状況に置かれている。そのため、これまで、ホームページや求人ポータルサイトなどによって募集が行われてきたが、中途採用においては紹介会社の活用や知人を媒介した紹介によって、新卒採用においては研修生として研修期間を設け、優秀な人材をスクリーニングして採用する方法を取り入れている企業もある。

開発者の育成は、これまで OJT を中心に行われてきたが、今後は Off-JT やキャリア開発支援も重視し、開発者の主体的（自律的）なキャリア開発を支援していく意向が示されている。開発者のプロデューサー到達までのキャリアパターンは、①開発職（プログラマ、グラフィッカ、プランナ）経由と、②営業職経由の 2 軸が抽出された。とくに、開発職として初期キャリアを歩んだ後（平均 8.36 年）、ディレクターを経て（平均 2.57 年）、プロデューサーに至るキャリアパターン（初職から起算して平均 10.29 年）は、企業組織内部におけるプロデューサーの典型的なキャリア発達過程であることが示唆される。開発者のキャリアラダーは、各職種を数年経験後、専門職と管理職に分岐していくが、具体的な人材要件の定義、将来的なキャリアパスモデルの提示等が課題となっている。マクロ的にみれば、開発者の転職行動・採用活動は、業界内でほぼ完結しており、ゲーム産業全体で開発者の育成が行われてきたといえる。

開発者の評価は、2～3 名による多段階評価が行われ、評価結果・評価理由の開示（フィードバック）が多くの企業で実施されているが、評価基準の公開や評価に関する苦情処理の仕組みが未整備の企業もあり、評価の客観性や納得性が担保されているとは言い難い。また、開発期間の長期化、ネットワーク化等によるビジネスモデルの変化に伴って、従来の評価制度では運用が難しくなってきている。とくに、評価の客観性、納得性の確保につ

いては課題が残されている。

開発者の処遇は、これまで主に基本給について成果主義的な賃金制度改革が行われてきたが、今後は賞与・一時金について個人業績やチーム業績との連動を強化していく意向が示されている。しかし、プロジェクト間（タイトル報奨金）で大きな格差が生じることもあり、利益の分配という観点から処遇上の課題を抱えている。

4.7.2 結論

ゲーム産業の人材マネジメントは、人事部と開発現場との連携により行われているが、人事部よりもむしろ開発現場の権限が強くなっている（図 4.1-05）。

図 4.1-05 人材マネジメントの役割分担の現状

人事部		開発現場
人材募集・採用	獲得	開発職応募者への面接・採用者決定
全社員の育成（主に Off-JT、キャリア開発支援）	育成	開発者の育成（主に OJT）
評価制度の策定、評価結果の調整	評価	開発者の評価
給与計算、規定整備、福利厚生充実	処遇	開発者の人材配置・金銭的処遇

とくに、採用者の決定や、最終的な評価においては、開発現場のイニシアティブが強く、人事部は事務的な役割に留まっているケースが多い。この背景には、プロジェクトを単位とした開発組織における人間関係のマネジメント（構築・維持）が、企業の業績向上にも大きく寄与していることが考えられる。また、職種別・雇用形態別にみると、人材ポートフォリオが形成されており、開発プロセスに応じた人材の組み合わせが行われている。現状においては、経営戦略と人材マネジメントが連動しているとは言い難い状況にあるが、ゲーム産業を取り巻く環境が劇的に変化していくなかで、今後、両者の連動強化は必須となってくるものと思われる。

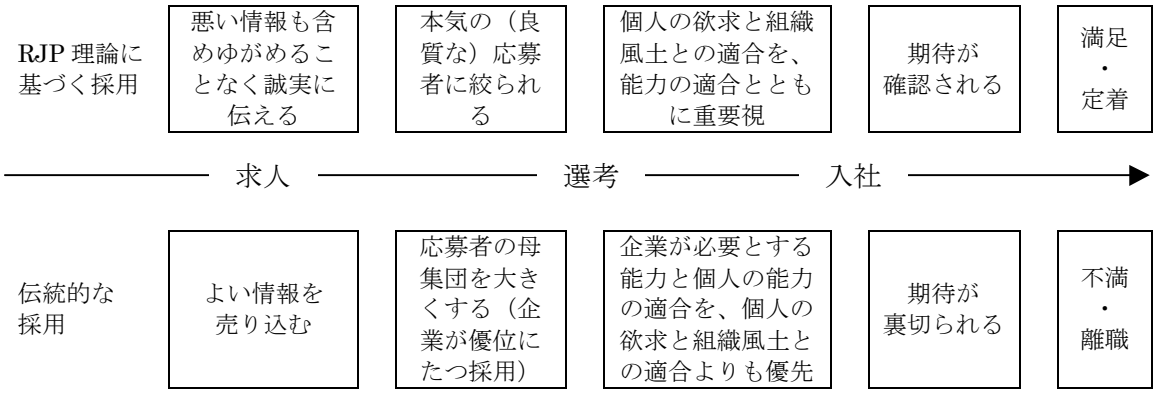
Ulrich（1997）は、人事部の役割について、何をするのか、できるのか（doable）ではなく、どのような価値や結果がもたらせるのか（deliverable）という発想の転換が必要であることを指摘している。人事部は経営戦略を達成するために、開発現場にどのような価値を提供できるのか、検討すべきであろう。

例えば、ゲーム産業は、開発者の獲得や育成において大きな課題を残しているが、Wanous（1992）による RJP（Realistic Job Preview）を導入し、優秀な人材の確保や定着に結実させることも一つの deliverable であろう。RJP とは、入社前に、よい面だけでなく悪い面などを含め、できる限り現実に即した仕事の情報を、求職者に提供する採用方法である。伝統的な採用方法は、企業や仕事のよりよい情報を魅力的に提供することによ

り、応募者や内定者の確保を行ってきたが、RJP は、リアリズムに基づいてありのままの情報を提供することにより、応募者や内定者の獲得よりもむしろ人材の定着を目的としている（図 4.1-06）。

RJP には、①ワクチン効果（過剰期待を事前に緩和し入社後の幻滅感を和らげる効果）、②役割明確化効果（入社後の役割期待をより明確かつ現実的なものにする効果）、③スクリーニング効果（自己選択、自己決定を導く効果）、④コミットメント効果（入った組織への愛着や一本化の度合いを高める効果）という 4 つの効果があることが指摘されている。

図 4.1-06　RJP 理論に基づく採用と伝統的な採用との比較



出所：堀田（2007）p.62

雇用のミスマッチを解消し、人材の定着とモティベーションの向上を図っていくためには、新たな採用システムの開発が必要である。繰り返しになるが、ゲーム産業における競争優位の源泉は、ヒト・モノ・カネ・情報という経営資源のなかでもヒト（開発者）である。ゲーム産業の転換期にある現在、deliverable 発想に立った人材マネジメントのあり方が求められていると言えよう。

また、上記と関連する事項であるが、人材マネジメントを「情報」の観点から捉え直すことも有効である。平野（2006）は、「情報システム」、「人事管理特性」、「組織モード」という鍵概念をもとに、日本の大企業ホワイトカラーを対象に、ケーススタディーと定量的な実証研究を行い、日本型人事管理の進化プロセスとその要因、機能的合理性などについて明らかにしている。その一つの結論として、「本社人事部は社内の人材に関して正確な情報をすべて手に入れることはできない。組織モードは、このことから生じる人事情報の費用を解決するように進化的に修正される。」（p.240）と、人事情報の非対称性と粘着性の原理を導き出している。また、「マネジメント人材とエキスパート人材の二極化した群別管理は必然的に進展するから、それに対応した新たな人事施策を準備しなければならない。その合理性の所在は人事情報の費用問題にある。具体的にはマネジメント人材の人事部個別管理強化であり、エキスパート人材に対するキャリア自律支援である。」（p.246）と、実践的含意を提示している。

ゲーム開発者のキャリアラダーにおいて、マネジメント人材とエキスパート人材に分岐している事実発見から、上記の知見はゲーム産業においても十分に適用できるものと思われる。また、開発現場の管理者がもつゲーム開発者の詳細な人事情報を人事部に開示しない（人材を抱え込む）ことによる「人事情報の非対称性費用」と、開発現場に埋め込まれた人事情報を収集・活用するためにかかる「人事情報の粘着性費用」という 2 つの「人事情報の費用」が、「組織モード」の合理性を決定する。この原理に従えば、日本のゲーム産業の人事管理は、J 型組織モードと A 型組織モードを折衷・混合させた進化 J 型へと移行しているということができる。

今後、より一層国際競争が激化するゲーム産業の人材マネジメントを検討していくにあたって、上記の理論的モデルは大変示唆に富むものである。日本のゲーム産業が競争優位性を維持・向上していくためには、産学協同によるさらなる処方箋の開発が求められている。

4.8 残された課題

今回は、 開発者の獲得・育成・評価・処遇に焦点を当てつつも、総合的な人材マネジメント調査に留まった。しかし、このような研究の蓄積は遅れている。例えば、日本のゲームソフト会社 9 社に半構造化面接を行い、人材マネジメントの実態を明らかにした小橋（1998)、日本のゲームソフト会社 14 社にインタビュー調査を行い、開発に関連したマネジメント（内製・外製および開発者の雇用方針）を明らかにした新宅（2000)、生稲（2003a)、日本のゲームソフト会社に郵送によるアンケート調査（有効回答 85 社）を行い、開発者の採用・育成・報酬制度を明らかにした新宅・田中・生稲（2000)、生稲（2003b）など、管見の限り数少ない蓄積となっている。したがって、近年のゲーム産業における人材マネジメントの実態を把握する上では、本調査は意義があったものと思われる。

今後の課題としては、第一に、本調査で最大の課題として挙げられていた「開発者の育成」に焦点を絞り、深く掘り下げた調査が必要である。例えば、OJT には、一言では表現できない豊かな仕事経験が埋め込まれている。OJT を体系的な能力開発プログラムとして企業内に定着させていくためにも、一皮むける経験（Quantum Leap Experience）の研究など、OJT の内実を正面から取り上げて研究していくことが有効であるように思われる。一皮むける経験の研究とは、リーダーシップ開発から生まれた研究であり、イベント（過去の仕事経験において、一皮むけたと感じられる具体的な出来事）とレッスン（イベントからの教訓）を収集し、コーディングを行い、クロス集計により、経験と教訓の体系化を試みるものである（金井・古野,2001)。また、この研究では、インタビュー・プロセスが学習の機会となる副次的な効果を持つ。つまり、内省を促され、

気づきを高められるのである。したがって、このような科学的なアプローチによって、どのようなイベント（仕事）がどのようなレッスン（教訓）を生み出すのかを明らかにすることで、より効果的な人材育成が可能になるのではないだろうか。

第二に、業界内には“プログラマー30(35)歳定年説”（プログラマーとして働くことができるのは 30(35)歳まで）という定説をどのように読み取るべきかという課題が残されている。今回の調査でも、平均勤続年数の短さ（5.09 年）、平均年齢の若さ（31.28 歳）が確認された。これは Arthur ら（1996）が指摘するようなバウンダリーレス・キャリア（境界のないキャリア）としての新しい働き方なのか、それとも人材の未定着を人材マネジメントあるいはキャリアディベロップメントにおける問題として捉えるべきか。このような現状を鑑み、ゲーム産業における人材マネジメントやキャリアディベロップメントに関する研究の蓄積が求められているように思う。

参考文献

Arthur, M.B. and D.M. Rousseau（1996）*"The Boundaryless Career as a New Employment Principle"*, The Boundaryless Career: A New Employment Principle for a New Organizational Era, pp.3-20, Oxford University Press.

David Ulrich（1997）Human Resource Champions, President and Fellows of Harvard College.（=1997, 梅津祐良訳『MBA の人材戦略』日本能率マネジメントセンター）

藤井大児（2005）「ゲームソフト開発における戦略オプションの選択」『岡山大学経済学会雑誌』37(1)、pp.19-34

藤原正仁（2006）「コンテンツ産業における人材育成の現状と課題」馬場章・藤原正仁『コンテンツ分野における人材育成に関する調査研究報告書』、pp.107-136、東京大学大学院情報学環

———（2007）「我が国のゲーム開発会社の人材マネジメント」社団法人コンピュータエンターテインメント協会『ゲーム産業における開発者人材育成事業報告書』、pp.45-80

福島さやか・荻野進介（2005）「どう育てるか、プロデューサー型人材」『Works』No.69、pp.39-43、リクルートワークス研究所

福島真人（2001）『暗黙知の解剖：認知と社会のインターフェース』金子書房

波田野匡章・守島基博・鳥取部真己（2000）「戦略的 HRM を生み出す『人材ポートフォリオ』」『Works』No.40、pp.2-29、リクルートワークス研究所

平野光俊（2006）『日本型人事管理：進化型の発生プロセスと機能性』中央経済社

堀田聰子（2007）「採用時点におけるミスマッチを軽減する採用のあり方：RJP (Realistic Job Preview) を手がかりにして」『日本労働研究雑誌』2007 年 10 月号 (No.567) 、pp.60-75、労働政策研究・研修機構

生稲史彦（2003a）「ソフトビジネスにおける企業像」新宅純二郎・田中辰雄・柳川範之

編『ゲーム産業の経済分析―コンテンツ産業の構造と戦略』pp.167-205、東洋経済新報社
生稲史彦（2003b）「ソフト開発の内製・外製とパフォーマンス」新宅純二郎・田中辰雄・柳川範之編『ゲーム産業の経済分析：コンテンツ産業の構造と戦略』pp.207-223、東洋経済新報社
生稲史彦・新宅純二郎・田中辰雄（1999）「家庭用ゲームソフトにおける開発戦略の比較：開発者抱え込み戦略と外部制作者活用戦略」東京大学ディスカッションペーパー、CIRJE-J-11.
金井壽宏・古野庸一（2001）「『一皮むける経験』とリーダーシップ開発：知的競争力の源泉としてのミドルの育成」『一橋ビジネスレビュー』Vol.49, No. 1、pp.48-67
小橋麗香（1993）「家庭用テレビゲームソフト産業の戦略と組織」『BUSINESS INSIGHT』、No.1(3)、pp.74-90
―――（1996）『日本の家庭用テレビゲーム産業：その構造と組織過程』神戸大学大学院経営学研究科博士論文
―――（1998）「日本のゲームソフト会社の人材マネジメント」『国際研究論叢』12(4)、pp.1-22
McCall, Jr. M.W.（1998）High Fryers: Developing the Next Generation of Leaders, Harvard Business School Press.（=2002, 金井壽宏監訳『ハイ・フライヤー：次世代リーダーの育成法』プレジデント社）
日本経営者団体連盟（1995）『新時代の「日本的経営」』日本経営者団体連盟
新宅純二郎（2000）「ゲームソフト開発における開発者マネジメントと企業成果に関する研究」http://www.gbrc.jp/content/old/PDF/20000914.PDF
新宅純二郎・田中辰雄・生稲史彦（2000）「家庭用ビデオゲーム開発企業に関する実態調査：製品戦略、製品開発、人的資源管理における 3 つの企業類型」東京大学 ITME Discussion Paper、No.47
田中辰雄（2003）「ハード・ソフト間のネットワーク外部性の実証」新宅純二郎・田中辰雄・柳川範之編『ゲーム産業の経済分析』pp.41-70、東洋経済新報社
内田康彦（2006）「何を企業の中に残すべきか？」大久保幸夫編著・リクルートワークス研究所協力『正社員時代の終焉：多様な働き手のマネジメント手法を求めて』日経BP、pp.71-103
Wanous, J. P.（1992）Organizational Entry: Recruitment, Selection, Orientation, and Socialization of Newcomers, Addison-Wesley.
矢野眞和（2001）『教育社会の設計』東京大学出版会
米倉誠一郎・生稲史彦（2005）「日本のゲームソフト産業：シリーズ戦略の罠」『一橋ビジネスレビュー』Vol.53, No.3、pp.52-69

4.9　ゲーム関連技術教育についてのヒアリング

4.9.1　株式会社セガ

ゲーム関連技術について、株式会社セガ庄司氏、高橋氏、康氏にヒアリングを行った。

株式会社セガ CS 研究開発本部 CS R&D 推進部　部長　庄司　卓
1987 年入社。プログラマーとして開発業務に従事した後、セガサターン、ドリームキャストの開発サポート業務を行う。現在は、技術導入、技術情報管理など開発業務のバックアップを主として広範囲な開発サポートを行っている。

株式会社セガ 第二 CS 研究開発部第二プログラムセクション　セクションマネージャー　高橋敦俊
1991 年入社、メガドライブ、セガサターン、ドリームキャスト、PS2 などのプログラミングを手がける。各ゲームのメインプログラマーを経て、現在、プログラマーのマネージャーを担当している。また、新入社員研修は入社 2 年目より担当している。

株式会社セガ CS R&D 推進部　システムサポートセクション　主任　康　日準
2006 年入社、3D ツールの機能サポートや、イントラ制作管理、海外メーカー情報管理、フォントや素材管理等で開発サポートを行っている。以前は酒類・飲料系の会社でキャンペーンやワインのオンラインサイト担当など異業種からの転身。

株式会社セガにおけるゲーム関連技術教育

―― 御社におけるゲーム関連の技術教育について、どのようなプログラムがございますか？

庄司　まず、新人教育に関してですが、教育プログラムは 3 段階に分かれています。4 月の前半は各部署の部長を招聘して、弊社の事業内容ついての説明を 1 週間ほど行います。その後に、1 週間だけなのですが新入社員の企画、デザイナー、プログラマーを全員集めまして、共通でゲーム作りをする上で必要な知識を学んでもらいます。他にはムービーやネットワーク講座、企画ですとプレゼンの書き方、これは職種に関わらず全員が一回経験することになっています。
デザインの職種の人間も、やはり企画やプログラムについて知っておいてほしいので、グラフィックス開発のワークフローなどの説明も最初の 1 週間で行います。これは一か所に集めての座学形式です。
そして、次の段階は約 2 週間程度、ここでは企画・デザインとセクションが分かれて各担当者を決め、そのセクションに特化した内容の講義を行っています。企画職は主に企画

書の書き方やプレゼンテーションの仕方が中心になります。デザイナーは、2D/3DCG ツールがインストールされた PC を一人一台与えられ、実際に触りながら学習していきます。プログラマーに関しても同じく PC を全員に配って、ゲーム作りの基本的なプログラムの知識を順番に教えていきます。一通り網羅できるような内容を 2 週間に詰め込んでいます。
ここまでで、集合研修と呼ばれる全員を集めて一緒に行う講義は終わります。ここから先は各部署の配属が決まっているので、そこで引き続き研修を行います。ここ数年弊社で行っているのは「ミニゲームを各部署に行ったら作る」という研修で、これを 1 か月程やります。
5 月一杯かけて各部署の企画、デザイナー、プログラマーが小さなプロジェクトチームを作り、先輩達に色々と方法を教えてもらいながら 1 か月かけてゲームを作ります。その合同発表会を 5 月末頃に行い、この 2 ヶ月間で新人教育のプログラムは終了となります。

―― 製作するミニゲームは、特定のハードウェアや、カードゲームやボードゲームなど、何か具体的にお題はございますか？

高橋 毎年違うのですが、だいたいは PC 上で DirectX を使ってゲームを作ることが多いです。内容に関しては部署毎に先輩達がカードゲームやシューティングゲームといったお題を設定しています。期間は 1 か月しかないですから、できるものは限られています。その中で、できる範囲のものを先輩がテーマとして与えることがほとんどです。このミニゲーム制作は、弊社では十数年続いている新人研修になります。

高橋 年度によって 5 月で終わる年もあれば、発表会を 6 月末に設定して 2 か月ぐらいミニゲーム制作をすることもあります。

―― ミニゲーム製作はチーム制になると思うのですが、大体何人ぐらいになるのでしょうか？

高橋 大体、企画 1 名、デザイン 2 名、プログラム 2 名ぐらいが平均的です。それよりも人数が少ない場合、セクション毎に 1 人ずつの 3 人で行うこともあります。それは各部署に配属された人数によって変わります。

―― 教育担当は、社内の方がメインなのでしょうか？

庄司 ここ数年のやり方ですが、各部署から教育担当者を立ててもらい、更に「エルダー」と呼ばれる先輩社員を新人一人一人に担当してもらっています。その人達の中で打ち

合わせを行い、各講義の担当者を部署ごとに割り振ります。そして各部署の担当者がその部署から適任者を選んで講義をお願いする流れになっています。

―― 外部の新人研修を担当されている会社に委託することや外部から招聘講師を呼ぶことはされていないのでしょうか？

庄司 検討したことはあるのですが、ゲームメーカーにフィットした講義をしてくれる業者が全然いません。オーダーメイドで頼んだこともあるのですが、ちょっとゲーム業界への理解度がこちらの求めるものに達していなかったため、検討はしたものの結局お断りしました。

高橋 過去に、とある企業さんにお願いをして社員全員を連れて 1 週間程プログラム講習などを 1 回行いましたがそれきりでした。やはり内容的にゲーム作りとちょっとマッチしないものがありまして……。なぜそれを行ったかというと、その頃「C 言語」というプログラミング言語がようやく出てきた時で「皆が知らないので基礎を教えてもらおう」という理由もありました。今はさすがにそういうこともないので、社内で調達しています。

―― もし外部でやるとしても、いわゆるマナー研修くらいでしょうか。あるいはそれも社内で行っているのでしょうか？

庄司 研修期間内に、短期ですがマナー研修を行っています。それは開発だけではなくて各事業部も全員集めた研修で、これは業者さんにお願いしています。

―― 技術分野に関しては、ほぼ全て社内でまかなっていらっしゃるのでしょうか？

康 はい、専用のテキストが社内で用意されています。

高橋 本当は、どこかの企業さんがやってくれればいいのですが。社内研修にかける労力が結構多いので。

―― テキストで用意されているとのことですが、改訂は頻繁に行われているのでしょうか？

高橋 ある程度基礎的なところですので、毎年大きく変わることはないです。もちろん少しずつは変えているのですが、担当になった講師の方がある程度内容を見て調整しています。

康　去年の資料を基に合わないところを削除して、新しい内容を上書きしています。例えばグラフィックツールの 3D STUDIO MAX などはバージョンが変わってきますから、そのオペレーショングラフィックの写真の入れ替え作業は担当の人がメインで行っています。

――　教員担当の方は 1 年間のうち、どのぐらいの力を新人研修にあてていらっしゃるのでしょうか？

高橋　大体 1 か月程前から準備に入って講義内容と資料を揃えて、という作業ですから。1 か月弱ぐらいの準備期間があります。もちろん専任ではないので普段の仕事をしつつ行っていくことになります。

――　教員担当の皆さんでの指針会議などは行われますか？

高橋　それは最初に行います。大体の方針が決まった時点で 1 度資料を全員の方から出してもらいます。後は各部署のマネージャーの人達に内容を見てもらい「今年はこれで教えます」と確認した上で再度もう 1 度更新をして最終的なテキストにして教える流れになります。

――　教員担当の人数はどのくらいでしょうか？

康　15～16 人ぐらいだと思います。

高橋　毎年同じわけではないのですが、ある程度何年か実施して次に講師をする方達にバトンタッチしています。

高橋　講師以外にも、新人達の PC の場所を一人一人まわってサポートをする人も何人かいるので、講師以外にももう少し人数はいます。

高橋　エルダーは入社 2 年目や 3 年目の人がいますが、教えることによって分かることもたくさんあります。そういった 2～3 年目のスタッフへの研修も兼ねてエルダーを選出しています。

――　ゲーム開発については、各社それぞれ独特な部分があるとお伺いしているのですが、そういったところは何かプログラムをご用意されていますか？

高橋　いえ、中途採用で入ってくる方は即戦力ですので、現場で覚えてもらいます。

康　それが一番効率いいでしょう。

高橋　弊社では、中途採用というよりも、派遣やアルバイト、業務委託という形式が多く、ほとんど即戦力として働いていただいています。そういった面で技術教育などは特に行っていません。全てプロジェクトに入って学んでもらっています。

――　中途採用の方向けの教育プログラムを行ったほうがいいのではないか、といったニーズが社内から上がってきたことはございますか？

高橋　おそらく、中途採用で入った人にヒアリングするとそういった希望は出てくると思います。マネージャークラスからは上がったことがないです。

康　他の研修に比べるとニーズが多くなさそうな気もします。デザイナーでしたら弊社の場合、3D ツールを何種類か使っているのですが、例えば 3D STUDIO MAX をプロジェクトで使うならそのツールを使える人をメインに採用しますので、何かを教えるといったことはしていません。別のプロジェクトに行って、他のツールを使わなくてはいけなくなると話は別になるのですが。

社内での社員教育プログラムについて

――　社内の社員教育のプログラムはどういったことをされているのでしょうか？

庄司　一昨年にある技術において、外部の業者さんを使おうとしました。社内の人間では教えられることは限られてくるので、ちょっと違った視点で自分達の足りないところをと考えました。ゲーム業界のプログラミングは特殊ですから、スタンダードを学ばないままゲームメーカーに入って来る人が多数います。情報処理の資格を取得している人間も少ないです。一般的な技術に関する設計手法やプログラミング技術などを習得していない人間が多いのではないかという想定に至り、そういった業界向けのセミナーで組み込み式の設計手法を勉強してもらう試みを一度しました。しかし、ちょっとタイミングが合わず、1回でやめてしまいました。対象は入社 3～4 年目ぐらいの社員を対象に行いましたが、受けた人間の中でも、もともとできる人間は「だるかった」という意見がありましたし、成長過程にある人間はたとえば設計手法の方法について「一般的にはどういう考え方をするのかを知ることができて大変役に立ちました」という意見もありました。なので、今後どうあるべきか検討しているところです。ゲームとは面白くなかったら作り直しの「スクラップ＆ビルド」ですから、設計の概念が非常に希薄です。一般にアプリケーションを作る場合は設計書通りに作らないと駄目ですが、ゲームの場合、設計書通りに作っても面白くなかったら終わりですから。その設計をするための発想をいかに切り換えてできるかのオ

ブジェクト思考のような感覚で「これが駄目だったらこう組み合わせて設計しなおしたらどうか」といった勉強をしてもらおうと試みています。
他には開発者向けのセミナーや CESA の主催するセミナー、Game Developers Conference、SIGGRAPH といったセミナーになるべく行ってもらうようにしています。Game Developers Conference と SIGGRAPH は海外のカンファレンスですから英語ができなくてはいけないのでかなり人を選びます。しかし、英語ができてそういったジャンルの勉強をしている社員が多いので、そこに出張させてフィードバックを行う方法をとっています。

―― 海外から戻ってきてから、社内で勉強会を行ったりするのでしょうか？

庄司　はい。その時のテーマにもよりますが小さい部屋でやることもありますし、100 人ぐらい集まれるところで発表をすることもあります。

康　後はレポートを社内で見られるようにしています。

高橋　他に、社内勉強会については、現場に近いところでニーズが上がってくることがあります。例えば「新しい技術を導入したいので知っているプログラマーに講師をしてほしい」と現場レベルで人数を集めて行うことを各部署で行っています。そういったことに対応していったほうが現実的で即効性も良いです。組織立ててやると腰が重くなってしまいますから。

高橋　組織として行うと社員から「あれは行かないと駄目ですか？」と聞かれてしまいますので、現場のニーズに合わせてという具合です。

―― 社内勉強会については、その開催規模や回数、あるいは参加人数を会社側で把握されていますか？

庄司　費用がかかるような公な講習会は認識していますが、細かなものまで含めて全ては把握していません。

高橋　ただ、他の部署でこういう講習をしたのならば、こちらの部署でも教えてほしいというオーダーがあって、その講師にもう一度やってもらうこともあります。しかし、大体はその場で出てその場で終了することが多いです。

―― 社内勉強会を取りまとめて、体系的にマニュアルを作ろう、あるいは、勉強会の内容を社内のネットワークにアップロードしようといった動きはございますか？

康　はい。勉強会の資料は、誰でも見られるようにしてあります。

庄司　そういった情報は、現在なるべく CESA に集中するようにしています。私も CESA の技術委員なので、社内の人間をアサインして、積極的に講演させるようにしています。

開発技術の社内向けマニュアルやリファレンスガイドについて

―― 特定の開発技術等について社内向けのマニュアルやリファレンスガイドを作られていますか？

康　公にできるもの、皆に見せていいものはイントラネットに集めていますので、そういう意味ではドキュメント寄りのリファレンスガイドになります。デザイナー向けのシェーダー作成の内容などもあります。一般的な技術で、ゲームで使えるものばかりを集めているところもあります。

―― このイントラネットの管理はどなたが担当しておられますか？

康　ウェブとドキュメントを管理している人間が数名います。記載されている情報は載せる前に確認をしまして、誤りがあった場合は随時修正しています。

庄司　ドキュメントを公開するまではかなり時間をかけています。一度書き上がってから公開するまでは少なくとも 1 か月は。投稿者から「これは絶対正しい、これが自分のやり方だ」といわれた時はすぐに公開することもありますが、それは実際に投稿者が知っているものについて書いている場合です。「シェーダーはこのように作るもの」と詳しい人が直接書く場合もありますので、その場合は書きながら載せています。

―― 開発事例については一部の方だけ見られるのでしょうか？

康　これはセキュリティレベルをひとつ高くしています。開発に特化しているものですので。また、実際の作り方は担当部署に確認してから載せています。

―― イントラネットのドキュメントは大体一日どのぐらいの社員の方がチェックされるのでしょうか？

康　ものにもよるのですが、たとえばこれはデザイナーが新人向けに「実際にこういうのを注意しながら作るといい」というのをまとめたものなのですが、ログを見た感じでは多くの社員に毎日見られています。

庄司　今のところは IP 制限プラス ID とパスワードで、開発職しかアクセスはできません。一般の営業職の方はアクセスできないようにしています。

――　この Web ページの立ち上がりはいつ頃からですか？

康　ベースとなるものは 2003 年～2004 年あたりからあったのですが、今の形になったのは 2005 年からです。

――　現在の形式はどういった経緯でできたのでしょうか？

庄司　今までは現場主導で行っていたのですが「これは公にしてもいいのでは？」、「これ載せられないの？」という声を聞いたことがありまして。ドリームキャストのセポートセンターを運営していた際に、ひとつのサイトを皆が見に来ていて、そういった集中して見る場所があること自体は皆認識がありました。その上で、マルチプラットフォーム化、弊社のソフトウェアデベロッパー化に伴って情報集中に対する違和感がなくなったのだと思います。もともと、運営するサイドとしてはやるべきだと思ってやっていましたし、開発の人もそれほど嫌ではなかったようですね。

高橋　プロジェクトがひとつ終わるとそこで技術が変わるので、それを他のチームに入るとお披露目したい人もいれば、そのチームの内容を知りたいという話もあります。それを、こういうところに情報共有させることは自然な流れでした。

――　イントラネットにアップロードされた CG などの素材を複数タイトルで使用することはあるのでしょうか？

康　この議論は他社さんでもされていると思うのですが「あの世界観用に作ったものがこっちで流用できるっていうのは、効率はいいかもしれないけれどちょっと変じゃないの？」と。私も再利用できればいいという考えですが、使えるとしたら「あるプロジェクトで出ていたあの郵便ポストを使いたい」というリクエストは聞いて右から左に「こういうリクエストがきているから流してあげてね」と、個別に取引するところをコントロールしたいなと考えています。現時点では素材を丸々取り放題みたいなシステムはありません。ただし「サンプルゲームやプロトタイプを作る分にはいいのではないか」という意見もあ

りますので、現時点では公開はしていませんが一応土台は作っています。それぞれ素材をデータベース化していますので。

庄司　データベース化して、全てを共有できるようにするのはいいのですが、そのシステムを作って公開する。それ自体が制作に使えなくても勉強になるのかなと。もちろん昔のプロジェクトや他の部署が作ったプロジェクト、モデルデータなども流用できるのですが運用が非常に大変です。

康　素材の管理タグを誰がつけるのか、ということ。それらのルール付けなども「これは公開できるけどこれは公開できない」といったことや、例えばライセンスがついているものだったら「画像として見てはいいけどデータとしては取得してはいけない」など、そこは非常に複雑です。

――　今後の教育プログラムをどのように行っていく予定でしょうか？

康　新人研修プログラムに関しては、ここ数年やっと上手く行き始めましたのであまり大きな変更は考えていません。もちろん新しい技術などが入ってきたら、それをカリキュラムに組み込んでいきますが、方針としては変えずにいこうと思っています。

――　社員教育プログラムが一旦止まっているというお話がありましたが 2009 年はどのようにしていくのでしょうか？

庄司　中堅の底上げをしていきたいです。私の勝手な思い込みかもしれないですが、やはり設計手法を。コードを書くのはいいのですが、設計の部分が……。

高橋　現場から出てきているニーズとしては、プロジェクト・マネージメントが弱い部分があります。

――　制作工程管理はどのように教育していくのでしょうか？

高橋　仕事のワークフローの部分を見るのは独自なところがあります。必要に応じてその場で組み立てていたりして、後は一般的なマネージメント手法をチームとして導入しようとしているところもあったのですがなかなかマッチしませんでした。例えばツールですと MS プロジェクトで全ての行程を管理しようとしても、やはり上手く動かなかったりします。また、なかなか教育化できないところなので、それを教育プログラムに入れられるようになることができればいいなと、常々話し合っています。

庄司　ボトルネックは分かっているのですが、なかなかどうしようもないところで。先程も申しましたが、面白くなかったら進めてはいけない。それで逆戻りするとなると「どこまで戻るの？」、「これだったら作り直したほうが早い」と現場のメンバーはよく言います。そのジャッジや、マネージメントの仕方は私も勉強したい……皆途方に暮れているところです。

――　マネージメントの難しさは制作過程や工程にあるのでしょうか。

庄司　元々の開発フローの設計も間違っているのではないかと思っています。ですからプログラミング設計ではなくて開発手法の設計、大前提の問題です。もちろん、弊社の中でも上手くいっているプロジェクトもありますが、それを真似するとしたら、例えば年間で 10 本のタイトルを 3 本ぐらいにしなくてはいけないので……それは現実的ではないですね。

康　日本でゲームのプロジェクト・マネージメントを教えられる人は数人しかいないような気がします。つまり、昨今の大型プロジェクトを最初にスケジュールを組んで、きちんと最後までやり通した人ぐらいしか分かっていない。さらにそれを他者に教えられる人が何人いるのかということに繋がります。しかもその人は普通のソフトプロジェクトではなくて、ゲームというエンターテイメントでないといけません。

庄司　この手の話を何年も同じことを言っていますが、プロジェクト・マネージメントを誰に学ぶか、それは組み込み機器メーカーか映画業界しかないのです。たとえばハリウッドは完全に分業制です。隣で何やっているかわからないが、いつの間にかできているといった具合です。おそらく、そのやり方が正しいだろうとは思っていますが、それをいかに日本人の性格に合わせて持ってこられるかが問題ですね。
これを先人に学ぶわけではないですが。誰に頼んでもきっと答えは出てこないことも、これで半分ぐらいは答えが出るので、どうしようかなというのが今年のテーマです。私個人もそうですし、会社もそうでしょうし、ゲーム業界もきっとそうだと思います。

語学教育について

――　社内での語学教育についてはどういったものがございますか？

庄司　TOEIC の試験は弊社の人事部がまとめて申請をして、社内で奨励はしています。ただし、試験だけで勉強会のようなことはしていません。過去に何度か外国人の職員が何人か集めて部活動のように行ったことがありましたが、今は行っていません。

―― 即戦力として、海外の方をスカウトするようなことは特にはされていないのでしょうか？

庄司　やっている部署もあります。当然マネージャーが面接して判断しています。

教育プログラムの今後について

―― 御社のゲーム関連の技術教育は歴史的に見てどのように変化をしてきましたか？また、これからどういった形に変わっていくのでしょうか？

高橋　新人教育の内容は毎年バージョンアップしていますが、基本的にはミニゲームを作ることと皆で集まって技術教育をする、これは私が入社した 17 年前ぐらいからずっと実施しています。基本的にはその柱は変わらず今まで続いてきました。ですので、これを軸にして進めばいいのかなと思っています。また、新人教育プログラムは毎年使っている資料については蓄積があります。それを使って、2 年目 3 年目ぐらいの若手にもう一度教育するプログラムを組みたい、という話は現場で聞いたりしますので、そのへんも検討してみたいとは考えています。

庄司　他社さんも新人教育プログラムについては、似たようなやり方をしていると思いますので、どこかで教育プログラムの比較をしたいですね。

―― 社員教育プログラムについて今後、複数年にかけて何かできればよいなというようなイメージはあるのでしょうか？

庄司　新人教育は期間も、その間にやるべきことも決まっていますから、ある程度はつかめていると思います。ただ、何年もかけて社員全員のスキルアップに集中していく場合、例えば上からのバックアップ、それを運営する組織が必要になってくる。継続的な教育はしっかりとした地盤があってこそできるものだと思います。それに関して一生懸命な企業であれば多分実施していると思いますし、毎年の業績によって揺らいでいくようなところはできないのかなと。なるべく継続的にやる条件としてその地盤を組みたいと考えています。教育で一番注目しているのはプロジェクト管理で、しかもこれは管理職やプロジェクトマネージャーレベルの手法を身に付けることですね。

4.9.2　株式会社バンダイナムコゲームス

株式会社バンダイナムコゲームス　執行役員　人材開発担当　社長室　人材開発推進局
市川　秀久
1991 年ナムコ入社、業務用ゲームのセールスマンとして入社。2001 年からは製品開発の部署で主にマーケティングの推進に関わる仕事に携わる。2006 年 4 月に執行役員(家庭用ゲーム事業担当)、2008 年 4 月～執行役員人材開発担当。

株式会社バンダイナムコゲームス　コンテンツ制作本部　制作統括ディビジョン　制作管理部　クリエーターHRD 課　マネージャー　八木　幸彦
1987 年ナムコにプログラマーとして新卒入社。業務用・家庭用のゲームプログラマーを経て、プログラマーのマネージャーをやりつつ数タイトルゲームディレクターを経験、今年から現所属（新規部署）に異動、現在に至る。

株式会社バンダイナムコゲームスにおける教育プログラム
――　御社におけるゲームの関連技術教育にはどのようなプログラムがございますか？

市川　私が会社全体の人材開発について担当していまして、八木が主にコンテンツ制作本部のクリエーターの人材開発について担当しています。ですので、専門化が進めば進むほど八木の領域になり、新人はまず一旦人事部や私ども人材開発推進局で引き受けるというシステムになっています。主にマーケティングを学ぶ素養を最初に身につけようというコンセプトです。職種は問わず、基本的なマナーやビジネススキルなどから入り、半年間ぐらいは毎週金曜日にマーケティングや考える技術……これは問題解決思考というのですが……例えばマーケティングを学び、それを研ぎすますような能力をつけてもらいます。マーケティング系の教育というのはプログラマーやデザイナーは直接的に影響を与えないことが多いです。企画職においてはチームの動かし方から商品の仕上げ方、売り方まで、教育は初級レベルですが広い範囲をカバーしているので、大きく影響を与えられているのではと思っています。プログラマーやデザイナーなどは、半年間の基礎的な研修の途中からそれぞれのセクションで教育を開始する形になります。

八木　私どものコンテンツ制作本部という部署は、家庭用のゲーム中心に制作を行っているところです。他にクリエーターは別の部署に何人かいるのですが、配属になってしまうとそこで分かれてしまうので、私がこれからお話するのはコンテンツ制作本部に限ったことになります。
まず、新人研修のカリキュラムはきっちり半年間取っています。主にゲームがどうやって

作られるかというのを徹底的に叩き込むということで、座学で基本的な流れを教え、その後実践に入ります。

特定技術の教育プログラムについて

―― 特定の開発技術について教育プログラムや勉強会はございますか？

八木　プログラマー職からお話ししますと、基本的にソフト開発の業種なので元々プログラムの素養がある学生を採用しています。それに磨きをかける形で「C++」を中心にソフトを組む基礎を学んでもらっています。ひとつ変わっているという点では、コンシューマーなのでハードをいじる機会がほとんどないのですが、ハードの研修も行っているという点です。「ワンチップマイコン」を秋葉原で買ってきてもらい、それで一からアセンブラで組んで作成してもってくる、そういう研修内容も含みながらハードの知識を少しずつ植え込み、ソフトを作ってもらうのがソフトエンジニアの研修です。ビジュアル職に関しては専門のツールがたくさんありますので「Maya」や「Photoshop」を習熟させるというのが一番必要な要素なので、そこを重点的に研修しています。企画職に関しては発想法であるとか、出てきた企画をきちんと紙にまとめる、文章にして人に伝えられるような形にしていく、そういった作業をしてもらった上で、その後全員をまとめて合同研修という形で一個作品を作ってもらっています。それがきちんとしたゲームの形になればいいのですが、ならなくても全然問題が無く、コミュニケーションを取りながら作品を作り上げていき、実践を兼ねてのコミュニケーションの研修になると。これらはすべて半年間で行っています。

―― ソフトが実際にできているかできていないかの割合は、どのくらいでしょうか？

八木　完成という定義が難しいですが。例えばアクションゲームでボタンを押し走ってジャンプしてゴールをするくらいのものは誰でも作れます。それを完成というのであれば完成まではいきます。ただ、そこに面白さがしっかりと入っているかいないかを考えると少し難しいかもしれません。

―― 新人教育のカリキュラムに関しては、例えばバンダイ色が強いとかナムコ色が強いといったことはございますか？

市川　ナムコ色です。旧バンダイは技術者がいませんので。コンテンツ制作本部自体に旧バンダイの人間はほとんどいません。教育は、クリエーターHRD 課と人材開発推進局でお互い協力しあって行っています。なにせ人数が多いものですから。

―― そういった中で他に専門のスタッフや教育専門の方は、何人かいらっしゃるのでしょうか？

八木　職種の上の人が仕切ってやってくれていますが、我々のように専門という形ではないです。

―― 例えば新人研修において外からどなたか講師をお呼びするとか、最近は新人研修を外注で請け負うような企業がありますが、そういったところをご利用になられたことはございますか？

市川　技術以外について、例えばマーケティング系は一部外部講師を呼んでいます。あるいは外部の研修で十分スキルが高まった社内講師に頼んで行ってもらうケースが多いです。技術系の講師は外部に頼むことはほとんどありません。

―― それはニーズに合った講師がいないことが理由でしょうか？

八木　それもありますし、後は費用対効果を考えて中の人間で十分まかなえるレベルという理由もあります。

市川　ただ、今後それが必要になるかどうかは検討の余地があると思います。今までやったことがないということと、効果がどのぐらいあるのか正直計れないので基本 OJT（On the Job Training）という形式をとっています。新人の時に一通り教えた後はプロジェクトに入って、実作業を行いながらスキルアップしてもらうのが基本スタンスです。

―― 中途採用された方の教育は実際に業務を行いながらになるのでしょうか？

市川　元々、中途採用の人はある程度、技術力を確認して入ってもらうので、特に技術的な教育は施していません。中途採用の社員は、一応必要最低限の開発技術ではなくて全般的な知識を二日か三日ぐらいかけて研修しています。3 か月から半年間に 1 度、中途採用した人を集めて行っています。

―― その技術力の確認は、部署の長の方が面接の段階でシートを見てチェックされるのでしょうか？

市川　雇用の形態によるのですが、正式な職員として採用する場合はある程度の作品を提出してもらっています。

八木　あと、プログラマー系の方は筆記試験をやります。

――　実際に何か定期的な社内セミナーや、あるいは「毎週何曜日にこういった授業をやります」といった取り組みは行っているのでしょうか？

八木　一昨年ぐらいから試みているのですが 3～4 か月に一回ぐらい外部から人を呼んでいます。弊社の大会議室にプログラマーを 200 人くらい集め、外部の方を呼んで講演をしていただいています。最初にやってもらったのはテストエンジニアといって、ゲーム業界ではわりと手をつけない職種でした。一般的な組み込みのソフトウェアなどをきちんと行っている場所の、そういった手法を上手く会社で採用できないかということで。社内にそういった職種をつくるわけでないのですがプログラマーに今やっているゲームをこれからどうしたいかということを考えさせるために、そういう講演を行う試みはしています。参加率も 8～9 割は参加しています。

――　終わった後に、レポートの提出を義務付けていますか？

八木　それはそれぞれの部署にお任せしています。レポート提出をやっている部署もあるかもしれません。また、ほかには細かい勉強会は行っています。例えば自分の作品で培ってきた AI に関して「20 人ぐらい入る会議室で発表会をやります」とメールで流す、場合によってはそこに 40 人ぐらい集まって立ち見のときもあります。開催ペースは把握しきれていないですが、月に 1～2 回は開催しているのを見かけます。

――　それはプログラマーの方が自発的に行っているのでしょうか？

市川　はい。後は講師としてクリエーターが例えば「CEDEC」など外部に出向くことがありまして、それも本人には勉強になっているとは思います。

――　御社の社内的なスタンスとしては、本業のゲームをつくる仕事の中でそれを圧迫しない限りは、ある程度自由裁量なのでしょうか？

市川　可能であるなら外部に対して自分の考えやスキルを紹介するのは問題ないです。当然、どこで何を話すということは管理しています。

――　スタッフ間で何か取り組むことはございますか？

八木　10 人ぐらいの規模で、自分の部署の中で勉強会を行っているところは多いです。

ただ、どうしても横のつながりが少ないので、それをこれから仕掛けていこうというのが私のこれらの試みです。

―― 勉強会のレポートや、マニュアルなどの共有化は行っていますか？

八木 はい、クリエーターはレポート・マニュアルが苦手な人種なので、そこは一番馴れさせなくてはいけないところで、会社的に義務付けてもよいと思っているぐらいです。また、新人研修のマニュアルは完備されています。

―― 「今後は横の繋がりを」というお話ですが、具体的にはどんなアプローチをお考えですか？

八木 それは難しいことだと思っています。私の部署は今期から設立されました。横から言わないと動いてくれない方達を相手にしているのですが、なるべく私が働きかけるようには実はしたくないです。自分からやりださないと成長しないので自分からやりたい、という気持ちにさせるように仕掛けようということを今考えています。やらされている感があるとどうしても伸びませんし、クリエーターというのは本業と勉強を平行しながら行わなければならないので……。実践しながらスキルを上げていくのが本当はいいことだと思います。勉強会というのはあくまで横のサポート的なことになりますので大々的にこちらが構えていても彼らはついてこないです。なので、本当の製品を作っている隙間でどうしても自分でクオリティを上げたいという時に、何をしたらいいかというのを補助できるような形で何とかできればと考えています。

市川 それは全ての研修について言えることで、集合研修はあまり大きな効果が期待できません。会社としてはやった感があるのですが、実はあまり効果がない。要は自分でまずやってみて困ったという経験がないと、学ぼうという気が起こらないのです。先に教えておいてもすぐ抜けてしまいます。だから集合研修は基本的なところを教えるべきだと思うのです。それ以上は本人のニーズが発生しないといくら教えても効果が薄いです。より良いやり方というのはこれからじっくり作戦を練っていかないといけません。

―― 今期お二方の部署が立ち上がりとなったそうですが、それまではそういった部署は存在しなかったのでしょうか？

市川 人材開発というセクションはあったのですが、それはいわゆる汎用的な教育で技術的なことは一切タッチしませんでした。部署ごとに特に技術的な部分についてはある程度教育は任せるという具合でした。旧ナムコはそういう意味では、それほど胸を張って言え

ないのですが、総じて歴史的には高い技術力を誇っていたので、先輩から後輩に受け継がれていくことでその優勢を保っていたというところがあります。だいぶ辞めてしまった人もいますが。

八木　私はプログラムのセクションに前期までいましたので、そこでずっと横を繋いでまとめる役をやっていました。そういう意味ではセクションはなかったのですがプログラムに関しての教育というか啓蒙ということはずっとやっていました。

――　教育プログラムの成果が出てくるのは、ずいぶん先になると思いますが……。

市川　多分、なんらかの方法を工夫しないと具体的な成果として目に見えにくいと思います。ただ、必要な投資であることは間違いないわけで、これを否定したら、おそらく企業体としてはお終いだと思います。

――　今回、お二方の部署が立ち上がるに至った経緯は、例えば内部からこういう部署が必要という声があったとか、あるいは「外の企業でこういうことやっているから会社でもやってみよう」といったものがあったのでしょうか。

市川　やはりヒット作がなかなか生まれにくくなっているので、そこの壁を突き破るような人材を多く育てようという経営方針の中で、この二つの部署が生まれました。コンテンツ制作本部が技術者集団です。マーケティングというのは、基本的にどのセクションでも通用すると思います。私たちは、考え方に影響を与える部分を担当して、ここではもっぱら技術的な部分を担当しています。

――　社外セミナーへの参加を促進はされていますか？

八木　ツールを使うような講習会への参加は積極的に促進しています。「Game Developers Conference」であるとか、そういった場所には積極的に行ってもらっています。また、セミナーの情報はそれぞれのセクションの役職者に流しています。その中で、スタッフから希望が出るものに関しては本当に実になるものであれば行ってもらうようにしています。

教育プログラムの今後について

――　今後の教育プログラムの理想形とはどんなものでしょうか？

八木　先ほども触れましたが、クリエーターはマニュアルやレポートを書くのが苦手です。

ですがそこが一番やらなくてはいけないところです。ゲームはひとつひとつ違う作品なので標準みたいなものは作れないのですが、ノウハウは作れると思っています。そういうところを、きちんと全ての職種の人たちにやってもらいたいと思っています。プログラマーは小さい勉強会を行っていますので、それをもう少し活性化させれば大丈夫だと思っています。ビジュアルに関しては一番難しいところで、どうしてもプロジェクトに入ってしまうと、ビジュアル職というのは特定のもの専門になってしまいます。例えばモデルを作る人、背景を作る人、そこにスキルが集中してしまいます。ビジュアル職として全体を見渡せる人を育てにくい環境になっているので、そこをどう打開できるのかを考えています。どうしてもひとつの専門になると、それはいち作業員になってしまう。私たちはそういう風に社員を育てたくはありませんので。

市川　今はどちらかというとプロジェクト単位で動いてしまっていることが多いですから、例えばプロジェクトが終わって次のプロジェクトが始まるまでに、各自で個人的な研究、自分の技術を高める研究は行っていても、それがバラバラに行われている。今後はプロジェクトに入っている間は、当然そこで一所懸命やってもらうのですが、プロジェクトの合間に発生する技術研究を職種全体の底上げに使おうという構想は持っています。ですから同じ職種間での繋がりをできるだけ促進させるような組織の組み方、あるいは運営の仕方を工夫しています。完全に標準化するのは難しいのですが、皆が同じように利益を享受できるような、それぞれのパートでの研究をある程度繋がりをもって、やってもらえるような方法を採用したいと考えています。
これは組織の方針などにも関わることなので、これから方法を詰めていきたいなとは思っています。現場の問題意識として各自の研究がバラバラに行われ、繋がりがなくてプロジェクトが始まってしまうと、その技術研究が中断してしまいます。それを誰かが受け継いで職種のグループとして完成させるのが一番望ましいので、その点が現在の課題だと思っています。

――　開発者や、現場からの「このままでは良くないのでは」という意見がきっかけなのでしょうか？

市川　技術の蓄積やノウハウの共有みたいなものはずっと課題として言われてきたので、そこも工夫しなくてはいけないと思います。研究をやっていることを、みんなが知らないといけないし、その成果を十分に伝えればそれを引き継いでさらにレベルアップさせていける、だからみんなで完成させるようになってほしいです。職種単位での連携については、上手くいく職種と上手くいかない職種があると思います。

語学教育について

―― 語学教育についてはいかがでしょうか？

市川 こちらは個人のスキルに依存していて、本人が英会話を学びたいと言えば必要に応じて許可するかもしれませんが、ほとんどそういう要望は上がっていないです。とはいえ、海外が非常に重要な市場ということで、会社としては戦略を打ち出しています。欧米のゲーム開発者たちの優れた技術力と戦うために基本的な部分について学ぶ一歩としてまず語学を習得しないことにはと思っていますので、それは来年度の大きな課題です。2009 年度はこれらのテコ入れをしようと考えています。

―― 御社に海外のカンファレンスを理解できたり技術書を読めたりする方はいらっしゃるのでしょうか？

市川 大部分は読めないと思います。英語ができないので。私も先週本屋に行って海外のゲーム制作技術の本を見てきたのですが、この差は凄いと思います。ちょっと話がそれるかもしれませんが、日本語版の技術書はあるにはあるのですがやはり古い。英語版は、どんどん最先端のものが出版されていて、それを読めないと日本のクリエーターは正直厳しい。読めないと欧米の技術力にはもう対抗できないと思います。なぜならどんどん優れた考え方や、技術が出版されていくからです。

八木 海外出張の結果として外国語に興味を持ち、結果喋れるようになったというケースはありますが。

市川 今ほとんどクリエーターが海外に行っていないので。そこを今、戦略的に詰めようと考えていて、成長段階と前向きに捉えていただければ幸いです。

八木 一応人事部的には TOEIC 何点以上で資格取得奨励金のような制度があります。また、英語を勉強する時の費用の一部を負担する制度もありますので、人事的には補助はやっています。

ゲーム関連教育の歴史について

―― 30 年ぐらい前ですと人事教育という概念すら企業には無かったのではないかと思いますが……。

市川 きちんと組織的に教育をしていこうという流れになったのは、ここ 1～2 年のことです。もちろん技術者の中では、後輩に受け継いでいくのは大事だと考えはずっとありましたが。

八木　コンテンツ制作本部の前の、CS 開発部という家庭用の部署の時には、新人研修は特に力を入れていました。そこで 92 年ぐらいから始めて現在に至るまでずっと継続してやっております。新人研修に関しては先ほどもお話しましたが、最初に教えなくてはいけないところで、研修を受ける側も欲しているところなので、そこだけは強いと思います。

市川　昔はゲーム作りそのものが勉強というか、作れば売れた時代でしたから。ナムコ時代ですがポリゴンの技術も弊社がおそらくはじめてゲームで使った会社だと思います。新技術を自主的に取り入れて「これをやってみよう」といった、そういった技術的にも新しいものをみんなで学ぶ中でレベルアップしてきました。

――　体系化された技術を取り入れるよりも新しいものに取り組んでいこうと。

市川　そうです。常に新しいものをみんなでやってみようという風潮がずっとあったので、そういう時は自然にみんなの技術力がアップしていきました。
そういった意味では近年業務用が頭打ちになっているのが厳しいです。会社の場合、業務用で新しいものを色々自由に考えられる領域があったのですが……新しいゲーム基板などですね。現在人気の高い「戦場の絆」もあれは先に筐体ができたということがありまして、そこにゲーム企画が入ってきて非常に優秀な機械に仕上がりました。「コレができたら面白いね」という技術重視なところがずっとあったので、それを業務用で自由に追い求められている時は非常にいい感じで回っていたと思います。

八木　弊社に技術部というセクションがあるのですが、ここは新しいことを研究しています。ソフト的に新しいことというのは、これから先だと CPU が早くなったり RAM を多く積めたりできれば、実現できるという事が山ほどあります。それを先に研究していますので、あとはそれを実現できる環境にいつなるかというタイミングや、今の環境で擬似的にそれを実現する方法はどうしたらいいのかを研究しています。それらの結果は他のエンジニア職にも発信し情報共有しています。

教育システムの変化について

――　御社の中での教育はどのように変わっていくのか、構想はございますか？

市川　技術も含めて 1 年目、3 年目、5 年目、奇数年度研修コースを開催しようと思っています。
これはどちらかというと技術そのものというよりも、マネージメント力をそこで底上げすることで結果的に教育に寄与すると思っています。技術的なものも、マネージャーの力が上がれば教育を活性化させることができるというイメージをもっているので、そういう意

味でも技術系のほうも優秀なマネージャーを育てていこうと、それで 1、3、5、7 年という奇数年度研修を構想している訳です。単なるプレイヤーから組織力を上げるというところの考え方、組織力を上げていくことで組織全体として成果、実績を上げるというところを身につけてもらいたくて、結果的にはそれが技術力のアップに繋がるという考え方です。なぜならば単純な集合研修では十分な底上げができないからです。マネージャーの力を上げて、組織的にいつも OJT を活性化させる方向にしたほうが効果があるだろうと思います。だから今後も「このプログラム技術を教えます」といった集合研修はあまり考えていないです。必要な人が集まってやるというのはいいと思うのですが、人事とかこういうセクションが音頭を取ってみんなで一斉に教えるというのはあまり効果が期待できないと思います。

―― その研修は、新人であろうと中途採用であろうと全員が行うのでしょうか？

市川　そうです、まず優秀なマネージャーを育てるというのが中期構想です。やはり今はまだまだ十分にマネージャーが育っていない。それゆえに教育全体の技術的レベルを引き上げる組織の運営といったものが、若干上手くまわっていない感じがしています。

―― そこに気がついたきっかけ、あるいは今まで不足していなかった時期というのは、何か具体的に大体これぐらいの時期というのはございますか？

市川　最近なかなかヒット作ができない、ソフトが売れなくなってきた、という状況があります。その原因を分析してみると、企画が出しにくいとか、企画発想力が不足しているとか、企画を練る時間が足りないのではないかとか、そのようなことが原因の一部として挙げられました。結局そういうことを引き起こしているのが何かというと、いままでにわりと場当たり的な行動をしてきてしまったからなのではないか、と。それで組織、人材の質を高めてヒット作を出すのを促す仕組みが必要だということで、一番何が必要かといったら優秀なマネージャーを育てることしかないと気がついたわけです。

―― その体制についてのアプローチは現在どうなっているのでしょうか？

市川　正直申し上げてクリエーターがゲーム作りに一生懸命で、自分で手を動かすことが続いています。だから元々マネージャーが育ちにくいのです。

―― 生涯現場といった考えでしょうか？

市川　はい。作りたい気持ちだけが先行してしまう、中にはマネージャー的な素養がある

人でも、ゲームを作っていたい人がほとんどなので……。ですが、部下も含めて組織全体を鍛えて、素晴らしいゲームを排出するという喜びを覚えてもらいたいということです。一言でいうと一人が頑張り、にっちもさっちもいかなくなってきている、そこが歴史的な変遷だと思います。今は組織的に頑張らないと、いいゲームにならない時代になってきています。

八木　総勢が 70～80 人ぐらいの組織であれば、おそらく何とかなると思います。私どもの部署は 800 人いますから個々人がバラバラに動かれると何もできない。市川が言うとおりにマネージメントできる人を育てないと、せっかくのパワーが全然活かせないのです。

市川　特にゲームはそういったところが難しいです。例えばガンシューティングゲームでいうと、敵の出現位置が右にもう 10 センチずれていると面白いなど、そういった少しのことでまったく変わってきます。そうすると一人だけ優秀でも完璧に面白いゲームをつくるのは難しくて、それぞれが意識を高く持ち、高い技術で連携していかないといいゲームが完成しません。絵や音楽など、ひとつが欠けてもダメになってしまうかもしれない。ゲームは今やすごく難しいプロダクトだと思います。

――　来期の新入社員の方へ教育のプログラムは始まっているのでしょうか？

市川　はい、この間組み立てましたが、技術的なほうはたぶんもうすぐ……

八木　今はじめています。

――　では、4 月の一期から研修があるのでしょうか？

市川　来年は大体 7 月 8 月ぐらいまでに一通り仕上げようかと。

八木　やりすぎかもしれないのですが、内定者に本を渡したりもしています。

――　採用されている人数は、年間通して振れ幅はどれくらいでしょうか？

市川　新卒は、ここのところは大体 50 名前後で推移しています。中途に関しては必要に応じてといった具合です。

――　プロジェクト単位なので難しいと思いますが、例えば一年間全体を見た時にこの職種の中途採用者が多いといった傾向は見られますか？

市川　足らない職種がいつの時代でもあるので、今はプログラマーが足りないです。昔はビジュアルが足りないと言っていた時にビジュアルをたくさん採ったこともあります。

――　それは、総量として足りないのか、それとも欠員が出て足りないのか、どちらでしょうか？

八木　時代によって違うと思います。例えばビジュアルが足りない時は、ツールでカバーができなくて、手作業をしてもらわないと困るといった時があるのですが、時代が進んでいくにつれてツールとかでカバーできるようになると、必要ではなくなったりします。今はネットワークゲームにどんどん広がっていっているので、ネットワーク系の技術者も含めてプログラマーが全然足りないです。やはり時代によって変わってくると思います。

――　今は全体的にマネージメントスキルを持っている方が足りないのでしょうか？

市川　それが急務だと思います。それとプログラマーです。

――　先程、社内の方が外部の講師として参加されるお話を伺いましたが、それはCEDECのような場所の他に、学校もあるのでしょうか？

市川　はい、専門学校とか大学とか。学校は都内近郊だけでなく地方もあり、年間契約している学校もあります。毎月一回行くとか。ジャンルとしてはゲーム企画に関することが一番多いです。CEDECは比較的技術よりです。

――　日本は実際に先輩の背中を見て学ぶ風潮がある気がします。

市川　ゲームは草分け的存在です。何かを学んでやるみたいな雰囲気じゃなくて、作りながら説明していくものだという文化があるじゃないですか。だから、そこなのだと思います。あとゲーム特有の複雑さや、わかりにくさが教育の意味付けを妨げているというのがありますよね。結構ゲーム業界から教育機関に、ゲームクリエーターが育つプログラムが欲しいという話がそこかしこでされているみたいなのですが、未だにコレというのに出会ったという話は聞いたことがないですね。

4.9.3　フロム・ソフトウェアにおける社内ゲーム AI セミナーの紹介

三宅　陽一郎
株式会社フロム・ソフトウェア　技術部

(1)　はじめに

ここで「デジタルゲーム AI」という言葉を、デジタルゲームで用いられる AI 技術とゲーム・キャラクターに使用される技術という意味で使うことにする。

人工知能という学問がそうであるように、デジタルゲーム AI はその全体像が捉えにくい分野である。また、デジタルゲーム AI は広大な分野であり、また、日々急成長を遂げている分野であるから、キャッチアップするだけでも、一定の労力を強いられる。そこで、フロム・ソフトウェアでは、毎週一回一時間、デジタルゲーム AI のセミナーを社内で実施している（以下「AI セミナー」）。このセミナーは全社へ向けた自由参加のセミナーであり、2005 年から 4 年間、現在までほぼ毎週開催し、開催回数は 140 回を超えている。ここでは、このような社内セミナーが持つ短期な役割、長期的な意義、実施方法、継続のノウハウについて説明する。

(2)　社内セミナーが持つ意味

日本のゲーム開発企業において、社内技術セミナーは大きく 5 つの意味を持っている。

(a)　多忙な開発者に新しい技術を提供すること。
(b)　社内に継続して技術を積み上げるデータベース機能。
(c)　英語圏が形成する技術圏のレベルにキャッチアップする。
(d)　社内で問題となっている技術的課題を討論する場を提供する。
(e)　新入社員教育。

である。以下、各項目について順番に解説する。

(a)　多忙の開発者に新しい技術を提供すること

米ではプログラマーの中でも、専門領域がさらに細分化しており最近では「AI プログラマー」という職種が明確に確立した。一方、日本では他の開発ラインに「ヘルプで入る」という言葉もあるように「手が空けばあくほどどんな仕事でも引き受ける」という、区分を明確にしない文化があり。これは、技術学習に十分な時間が取れない、という事態

のみならず、専門領域を確定しないために、一つの分野に特化して学習できない、というアンチ・スペシャリストの現象を引き起こす（逆にオールラウンダー指向を育む傾向にある）。そこで、一つの分野に特化して、定期的に継続して行われるゲーム技術セミナーは、ある特定の分野の技術に特化して積み上げることで、モチーフを拡散させることなく、その特定の分野に特化して社内に継続した学習を促し、セミナー参加者に序々に高いレベルの技術へと到達する足場を提供するものである。

(b) 社内に継続して技術を積み上げるデータベース機能

社内セミナーについての実施情報は、全社的に公開し、常に仕事で出席できないメンバーに対しても、情報を送り続けなければならない。かつ、社内でセミナーを行う限り、セミナー用に作成した文書や情報は、全て社内で共有するべきである。これは、1 回、1 回だけを考えれば、微々たる情報であるが、セミナーが 10 回、20 回、…と、積み重なるに従い、膨大な情報となって相乗的に価値を持ち始めるからである。

セミナーの時間は限られているため、セミナーは、セミナー自身の開催の他に「メンバーが後で、自学自習することができる情報環境を作る」という、もう一つミッションを想定して行わなければならない。また、学習環境を提供することにより、セミナーに出席できなかったメンバーや、後から入社して来た社員に対しても永続的な学習環境を提供することになり、会社の情報財産として積み上がって行く。また、些細な点ではあるが、長期欠席したメンバーでもいつでも再参加できる雰囲気を持ち、その旨を明確に伝えることも大切である。参加メンバーによっては、不参加が長いことに引けを感じて出席しなくなることも少なくないからである。

AI セミナーでは、以下のような情報を毎回、蓄積している。

① セミナーのログ（実施日、セミナー見出し）
② セミナーで配布するセミナー内容をまとめた A4、1 枚以上の WORD 文書
③ 技術キーワード
④ 参考にした資料リスト、WEB 文書であればリンク

これらは、社内サイトを通じて共有され、誰でも利用できる形であることが望ましい。WORD 文書は本報告書に添付できないが、①～④の 130 回分のリストを本章の巻末に記載しておく。

この中で「技術キーワード」は特に重要である。AI セミナーでは、発表者の最低限のタスクとして、「一つの技術ワードを取り上げて解説する」ことを課している。逆に言えば、タスクの縛りはこれしかなく後は自由である。準備の時間がなければ、5 分程度の解説で後はディスカッションにしてもよい。こういった枷を最小限にしておくことも、参加障壁を下げるために必要である。

また、技術キーワードを用意することは、そのセミナー回の方向を簡略に指し示すと同時に、社員が WEB を通して、技術を検索する助けとなる。また、参加者側には、

「出席メンバーはセミナーで、最低、一つの専門用語を覚え、理解して帰る」

という原則を課している。この原則は、長期的に見て大きな意味を持っている。これによって、これまで、開発の現場であやふやな言葉で指し示しながら会話して来たことに対し、明快な概念と言葉を与え、やがてそういった専門用語が開発の現場で用いられるようになり、次第にレベルの高い議論が、開発者間で形成されるようになる。こういったキーワードが、50、100 と積みあがって行けば、もはや、それなしには議論できない高度な内容でさえ、日常的に会話されるようになり、開発のレベルの本質的な底上げを促すことになる。それは、一端導入されさえすれば、その企業にとって強固な基礎となるのである。

(c) 英語圏が形成する技術圏のレベルにキャッチアップする

ゲーム開発者は必ずしも英語の読解、ヒアリングに長けているわけではない。むしろ、技術文書や英語論文から技術を吸収することを必ずしも得意としているわけではない。英語力の向上には継続的な社員研修が別途必要であるが、AI セミナーにおいてそういった英語学習の効果を前提とすることは出来ない。ただ、技術分野を限定すれば複雑な文章があれば、一週間に 4～5 ページの英語記事を読むことによって少しずつ向上する。また、セミナーは、参加メンバーの持ち回りであるから、必ずしも毎週そういった時間を強いられる読解が課されるわけではない。

セミナー各回の担当者が英語記事の読解をすることは、社内全体で英語読解の労力を緩和すると共に、英語圏からの技術の流入を促し、社内の技術レベルの向上に大きく貢献する。また、これは殆ど国内に文書のないデジタルゲーム AI 分野では特に重要なことである。

紹介した英語記事・論文は、できるだけ簡単な要約とキーワードを用意し、社内全体に対して公開することが望ましい。そうすることで、多忙な各開発者が論文全体を読まなくても、「必要なときに必要な情報を持ち出せる、インデックスと理解の助け」を提供することが出来るからである。

技術セミナーで過度で英語力の向上は期待してはならない。英語力の向上は、毎日の鍛錬が必須であり、語学セミナーは別に開催することが望ましい。1 ヶ月に一度程度の読解では大きく向上を期待することは出来ない。しかし、英語文書を読解して解説するというサイクルによってモチベーションを与えることが出来る。また、それ以上に英語圏の技術蓄積の文化に触れることが大切なことである。また、英語圏の文献のネットワークの広がりに触れることは、如何に日本がゲーム技術に対しては情報後進国であることを実感することになり意識の変化を促す。

(d) 社内で問題となっている技術的課題を討論する場を提供する

社内セミナーは、その内容が何であれ、社内全体に対して開かれていなければならない。そこに、業務上仕切られた部署の垣根を持ち込むべきではない。むしろ、そういった垣根を超えて自由に議論できる場を提供することで、社内で新しい位置と価値を持つことになる。

また、社内でタイムリーな話題をセミナーの議題として上げることは非常に有用である。また、そういった話題は人をセミナーに引き付けるものである。例えば、ある開発ラインで「パス検索」が問題になっていれば、すぐに「パス検索」の話題を取り上げればよい。そのトピックに関する技術文書を用意し解説し議論することで、知識を吸収し議論のレベルを上げながら討論することが可能になる。

例えば、「敵 AI の攻撃の挙動」が問題になっていたとする。その場合、その問題に対する解決に近い技術文書・論文を用意し、その方法が解決につながるにせよ、そうでないにせよ、議論に変化をつけながら、セミナーをくり返すことで、序々に解決への糸口を見つける。そえらが、困難な問題である場合は、半年に渡って、同じテーマに関するセミナーを断続的に行う場合もある。そういった場合でも、新しい知識を学びながら討論することは、議論を空転させないためにも非常に効果の大きいことである。そして、開発者というものは、ある課題を抱え込んで学んでいるときが、最もよく知識を吸収するものなのである。

また、社内に対しては、そういった議論が行われていることを、最大限アピールすることである。100 人いれば、解決を思い浮かぶ人間が一人はいるものである。そして、こういった議論の記録を公開することである。そういった議論の記録は、同じ議論を繰り返す無駄を避け、また、新人は先人が議論した終わりから新しく議論を始める出発点を得ることが出来る。それは、確かに社内の歴史として積み上げられる実績なのである。

(e) 新入社員教育

大学などで、高い専門教育を受けて来たとしても、そういった知識を直接、ゲーム開発に役立てる機会は非常に少ない。ただ、右も左もわからない、新入社員の時機は、最も好奇心が高く、知識が吸収しやすい時機の一つである。実際に、開発ラインへ配属されると、2、3 年は、業務を覚えることに忙しく、そういった好奇心や柔軟性は、仕事への適応と共に一旦は失われてしまう場合が多い。しかし、ある程度のベテランになって、大局を見ることが出来るようになると、再び、高い技術に対する好奇心が復活する傾向がある。

ただ、次世代機の時代は、本格的な学術知識が必要とされる時代になりつつあり、教育機関で育んだ新入社員の高い知識を継続して発展させる環境も必要となりつつある。そういった知識をコーディングで活かす道を用意することも、企業としては必要である。

特定分野の技術のセミナーでは、そういった新人に出身の教育機関から継続した学習と興味の継続の場を与え、新人の持っている知識がゲーム開発へ如何に利用できるかを議論する場を与えるという意味で有用である。そういった議論を通して、学術知識から、ゲームへ特化した技術へ応用するための分岐点が次第に見え始めるのである。

また、会社としても、そういった新人が持っているリソースを社内で共有し吸収して行くことは大きな価値を持つことである。教育機関で育まれた、論文読解能力や、ディスカッションの能力、英語の能力を途切れさせることなく継続して教育できるという効用もある。一端、こういった継続が切れてしまうと、復帰できない場合も少なくないからである。

多忙なゲーム会社の業務はある側面で新人から多くの学習と発展の機会を奪っていることを理解しておく必要があり、これを緩和することが必要である。

(f) まとめ

以上、5 つの項目について解説して来たが、各項目の解説を通して挙げた、社内セミナーを実施する際のポイントについて列挙しておく。

① 社内セミナーは、社会に広く公開すること。特に、デジタルゲーム AI は、総合的な分野であるから、プログラマーだけでなく、企画職やデザイナーにも、職種を超えて参加できるセミナーを目指すことが必要である。
② セミナーの内容は、出来る限り形にして残して行くこと。出来れば、社内サイトから常にアクセスできる形で残して行くことが望ましい。また、検索しやすい形で積み上げることで、セミナーの副産物が永続的な価値を持つことになる。
③ 英語の文書にも積極的に挑戦すること。これは、英語圏が形成する技術を継続的に社内に取り入れ、技術レベルの向上・キャッチアップに繋がる。

④ 社内の開発で問題となっていることを、タイムリーに社内セミナーのテーマとして選択すること。これは社内におけるセミナーの価値を高めることになる。
⑤ セミナーは、社内の部著の垣根を超えて議論できる場とした雰囲気作りをすること。そうすることで、セミナーは社内で特別な意味と機能を持つことになる。

社内セミナーには、短期的な効用と、長期的な効用がある。短期的な効用は、必要な情報を開発者に与えること、長期的には、社内の技術ポテンシャルを上げることである。

新しい技術の流入がなければ、技術者というのは次第に自分の得意とする領域に閉じこもって行く傾向にある。新しい技術は新しい空気であり、そういった空気の中には、新しい時代のエッセンスが詰まっている。社内の開発者に常にそういう空気を送り込んでおくことは、技術に対する挑戦的な意欲を高めることに繋がる。

また、社内セミナーは、会社の理解があってこそ実施できる。しかし、こういった効果自体を、形として提示することは難しい。であるから、共有できる情報は可能な限り、利用しやすい目に見える形で積み上げる必要がある。さらに、社内の批判などには耳を傾け、それぞれの会社にあった形のセミナーを次第に形成して行くことが必要である。

積み上げられた知識は圧倒的な存在感を持つ。100 回積みあがったセミナーというのは、もはや他の追随を許さない分量と質を獲得する。それは、開発者、個々人の力となり、同時に会社の力となる。継続されるセミナーこそは長い時間をかけて、その会社を決定的に変えて行く力を持つのである。その成果がある塊り（クラスター）となるまでの時間が一番苦しい。しかし、一端大きな塊となって知識群は壮観である。その存在の意義を否定することは出来ない。参考に、巻末のセミナーリストを参照して頂きたい。

(3) 実施方法

この章では具体的な、AI セミナーの実施方法を解説する。まず、各回のセミナー担当者の実施手順を解説する。

① **テーマを選択する**　自分が業務で課題としている問題や、AI で該当分野がなければ、書籍や WEB から技術記事・論文を選択する。コツとしては、できるだけ、ゲームタイトルで実装された技術の解説を選ぶことである。そうすれば、単なる研究途中の内容ではなく、ゲーム開発の中で用いられることで洗練された技術を選択することが出来る。また、開発への応用への道も遠くないため、セミナー参加者からもそう不評を得ることもない。ただ、一方で研究段階の技術解説も、開発者が強く

惹かれる内容であれば取り上げることもいいだろう。開発者が惹かれるからには、そこには何かしらのこれからのゲーム開発に必要な要素が含まれているものである。また、大切なのは、その回のセミナー担当者の最低限の仕事を決めておくことである。そうすれば、時間が取れないときでも、とりあえずセミナーを開催することが出来る。AI セミナーの最小限の仕事は**「新しい（セミナーでこれまでに取り上げたことのない）AI の専門言葉用語を取り上げ、その概念を解説すること」**である。もちろん、それ以上の解説があればあるほどよい。ただ、キーワードの解説さえあれば、ディスカッションを行うことが出来、参加者にとっても、毎回一つのキーワードを覚えて帰ることが出来る。

② **調査・読解**　論文や文書を読み、A4、1～4 枚程度で、内容をまとめる。開発者は多忙であるから、この工程はあまりこだわらず最小限の要約が作成できればよい。資料は解説の道具であり、大切なのはまず、その担当者が内容を理解し他人に解説できる準備をすることである。

③ **アナウンス・公開**　社内サイトや、セミナーML で、セミナーの実施をアナウンスする。セミナーの参加人数にはこだわらない方がよい。セミナーを行う心構えとしては、

(A) まず準備を通して自分の勉強となることで、十分に報われたと考えること。逆に、そういった感触を持てるまで準備することが望ましい（これは、セミナーの参加者が誰もいなかった場合の心理的保険にもなる）。

(B) 仲間の開発の役に立つことに歓びを覚えること。 自分の興味のみならず、あの開発者には、こういう情報が大切なのでは、と考えて準備することは、めぐり巡って自分の興味と技術の幅を広げてくれる。

という二つである。この二つは一見、相反しているが、こういった二つの方向を持つことが、（特に主催者にとって）セミナーの継続には必要である。セミナーは、最低、自分と聴いてくれる人、2 名いれば実施できるので、事前に参加できる人には連絡してもらうか、友人に個人的に参加してもらうなど、最低限の参加者を確保することもよいだろう。それでも、誰も来なかった場合は、（A） と思って、次回を期すのがよい。筆者も誰も来なかったセミナー回を数回経験している。ゲーム開発企業は締め切り前には、多忙になる傾向がある。そういった場合でも、準備で勉強できたと思って、自分の場合は、その文書はそのまま公開して、次の回は全く別の内容を準備する。準備すること自体に価値を見出すことは継続のために必要なこ

とである。そして継続するで、一つの大きな価値をセミナーは獲得することになる。

④ **資料作成**　参加人数分の要約のコピー、参考にした論文のコピーを用意する。PPT やムービーがあれば、事前に共有フォルダに上げておくことが望ましい。事前に資料を上げれば上げるほど、参加者は増えるし、参加者から見た参加の障壁が低くなる。あらゆる手段を尽くして参加者の参加障壁を低くすることは、資料作成に限らず重要なことである。これは、実は業務において技術導入を進める場合と、全く同じ問題を含んでおり、セミナーを通じて、そういった感触をつかむことも有用である。特に、技術支援部に属している人間には有用だろう。職種の違う人々に、技術を噛み砕いて解説する訓練は回数を重ねれば重ねるほどよい。

⑤ **セミナー実施**　自分を含めて 2 人以上なら実施すること。主催者は一人の同志を持つことが出来き、楽しく開催していれば、自然と参加者を引き付けるものである。

⑥ **セミナー後の情報アップ**　セミナー後、社内 WEB に資料をアップする。ログを更新して、追加情報があれば、セミナーML に流す。議論した内容は社内サイトなどに簡単に記しておくとよい。

(4) セミナーのテーマの選択

各回のセミナーの選択は、セミナーの担当者にとって、最も大きく困難な課題である。このテーマの選択は、セミナーの初期であればあるほど、難しい。なぜなら、セミナーを始めた時期というのは、まだその分野の全体像が見えていないので、一体、自分が選択するテーマが、どれほど重要な話題であるかを判断できないからである。CEDEC や GDC の後であれば、そこからピックアップすることがよいだろう。右も左もわからなければ、その年の講演資料を追って行くだけでも、1 年のセミナーの題材に事欠くことない。

① 最初のステップ

そこで、20～30 回を超えるまでは、素直に、ゲーム開発技術のカンファレンス資料や、その分野の代表的な教科書に掲載されている記事や、参考文献の論文を題材にする方がよいだろう。また、教科書や論文、資料に掲載されている資料のリンクは重宝する。そういった**リンク集を社内サイトに、綺麗に分類して掲載することで社内の情報環境に貢献することが出来る**。次の章に、ゲーム AI の代表的な文献を挙げておく。

② 第 1 のステップ

次にセミナーを重ねて行くと、自分自身でテーマを見つけ、論文や記事を見つけ出す

段階になる。これは一つの論文や記事に依存するのではなく、「自分が抱えている課題から出発して情報を集めて解決する」という方向である。

③　第 2 のステップ

それを超えれば、次は、あるテーマについて、自分で深く考えるところから始めるのがいいだろう。例えば、自分で「集団 AI の制御の仕方」のアイデアを考えてみる。そして、自分が考えたアイデアと似たアイデアを考えた人がいないか文献を調べてみる。もし、誰もいなければ、文書に頼らず、自説をセミナーの議論の題材として提供する。似たようか論文があれば、自分のアイデアとどう違うか、また、その論文から、さらに自分のアイデアを発展できないか、さらに考えてみる。

④　第 3 のステップ

50～60 回を超えれば、その分野の全体像が見えて来るので、準備は格段に楽になる。同時に、テーマを取り上げつくした観に捉われるのもこの時期である。こういった段階になると、出来るだけ、社外のセミナーなどで、同じ分野のエキスパートたちの議論を聞き、実際にそういった人たちと議論してみることが大きな効果を持つ。自分が今まで気がつかなかった領域や、同じテーマに対しても違った視点を得ることが出来る。大勢の人の前で講演してみるのもよい。自分の意見に対する賛同者や反対者を得ることが出来る。

⑤　第 4 のステップ

60 回以降は、いわば、「2 周目」である。これまでの準備よりも、一層深い考察を経て、他人の仕事を吸収しながら、自分のアイデアを発展させて行くのがよいだろう。オリジナルのアイデアを一つでもセミナーで発表することを心掛けるとよい。それこそ、開発の現場で実際に裏づけされた専門性の高いアイデアを提供する力である。

⑥　第 5 のステップ

100 回を超えたら、一度、その分野全体像を描いて説明してみるとよい。1 回目のセミナーに比べたら、自分がどれだけ大きな分野を歩いて来たかがわかる。それと同時に、未踏野の分野があることがわかる。同時に、この頃になると新しく入って来た人が、これまでのセミナーにキャッチアップ出来る量を超えてしまうので、新人用のセミナーを開いてみるのもよい。

⑦　第 6 ステップ

150 回を超えたら…？　筆者もわからない。現在勉強中である。

(5) 資料

以下に、セミナーのテーマ選択に役立つ資料を紹介する。

(a) 論文・記事が集められた書籍

[1] Steven Rabin 編. “AI Game Programming Wisodom”, Charles River Media (2002)

現在 1～4 巻まで刊行され、続刊が予定されている。翻訳はされていない。

[2] Mark DeLoura（著), 川西 裕幸（訳), 狩野 智英（訳）“Game Programming Gems”,ボーンデジタル(2001)

現在 1～7 巻までが刊行され、順次翻訳されている。各巻に AI の章が用意されている。

(b) 論文・記事が集められた WEB サイト

[3] Jeff Orkin 氏のサイト： http://web.media.mit.edu/~jorkin/

F.E.A.R. でゴール指向アクション・プランニングの手法を導入して、ゲーム AI を大きく発展させた Jeff Orkin 氏のサイトには、ゲーム AI に関する論文、発表資料が豊富である。

[4] Craig Reynolds, “Game Research and Technology”,
http://www.red3d.com/cwr/games/

ゲーム AI 全分野にまたがる情報リンク集。Craig Reynolds は群制御のパイオニアとして有名。このサイトには、他にも重要な情報が掲載されている。

Steering Behaviors For Autonomous Characters
http://www.red3d.com/cwr/steer/

[5] BUNGIE PUBLICATIONS, http://www.bungie.net/Inside/publications.aspx

HALO シリーズを通して、賢く，面白い AI を両立し、常に高品質の AI を開発し続け来た BUNGIE の技術資料公開のサイト。Damian Isla 氏の資料はゲーム AI の基礎である。

[6] CGF-AI, http://www.cgf-ai.com/links.html

AI のコンサルタントとして KILLZONE の AI などを共同で開発。
リンクにはゲーム AI の情報多数。また、彼ら自身の仕事も発表している。
http://www.cgf-ai.com/cgfai.html

(c) 教科書

[7] Mat Buckland (著)、 松田 晃一 (翻訳).

“実例で学ぶゲーム AI プログラミング”,オライリー・ジャパン (2008)

「スクリプティング AI」「有限状態機械」「群制御」「グラフ検索」「プランニング」など、ゲーム AI の中核的な概念を、ゲーム開発者が実際に使用することを念頭に、概念の解説は明晰、ソースコードはエレガントで高品質、実用の為の注意点は的確、入門書としても専門書としても高い完成度と自由度を誇っている。また、随所に、実際に技術が応用されている商業ゲームタイトル名や画像が紹介されており、ゲーム開発者に向けて応用を促そうとする作者の懸命な意図が感じられる。さらに、各章ごとに、デモプログラムが用意されており、ソースコードのポイントが解説される。全てのコードはインターネット上で誰でもダウンロードすることが可能であり、プログラマーはこのサンプルコードから直ちに実際的な開発へ向けて出発することが出来る。

Programming Game AI by Example - Resource Page (Wordware Publishing)
http://www.wordware.com/files/ai/

特に「プランニング」を扱った最終章は、ゲーム AI の書籍としては最初のまとまった解説である．この同時期に、アクション・プランニング（キャラクターの一連の行動を目的に応じて自動的に計画する技術）を応用した最初のゲーム『F.E.A.R.』（2004, Monolith Productions）がリリースされ、この技術が現在、次世代ゲーム AI の最も重要な技術になりつつあることを思うと、その先見の明には驚かされるばかりである。実際、私はこの最終章とサンプルデモを発展させて、ChromeHounds (2006, FromSoftware)の AI のシステムを完成させた．Mat Buckland は ai-junkie.com というサイト内で、さらに広範な人工知能技術のデジタルゲームへの応用を展開している．唯一の問題点は、彼の著作が 2004 年であり、この 5 年でゲーム AI 技術は劇的な発展を遂げたために、最新のゲーム AI 技術にキャッチアップするには、これを基礎に最新のトピックについて学習する必要がある。

[8] John B.Ahlquist,Jr., JeannieNovak, “Game Artificial Intelligence”, Delmar Pub. (2007)

プログラマー以外でも読めるトピックなゲーム AI 事例が紹介されている。

(d) ゲーム AI 連続セミナー資料

IGDA 日本において長久氏と三宅が、2006 年～2007 年、一年に渡って実施したゲーム AI をテーマにした連続セミナー。PPT 資料の中に、ゲーム AI の情報への膨大なリンクがある（以下[9]～[28]のうち、igda.jp の資料は URL は変更の可能性があり。ただ、そ

の場合も、IGDA 日本のサイト内に置かれる予定でトップページから辿ることができる予定。）

[9]ゲーム AI 連続セミナー第 1 回「Killzone における NPC の動的な制御方法」
［資料］ http://www.igda.jp/modules/mydownloads/visit.php?cid=2&lid=60

[10]ゲーム AI 連続セミナー第 2 回「F.E.A.R におけるゴール指向型アクション・プランニング」
［資料］ http://www.geocities.jp/mnagaku2000/igda/gate.html?gameai2doc.zip

[11]ゲーム AI 連続セミナー第 3 回「Chrome Hounds におけるチーム AI」
［資料］ http://www.geocities.jp/mnagaku2000/igda/gate.html?gameai3.zip

[12]ゲーム AI 連続セミナー第 3 回補足資料
［資料］ http://www.geocities.jp/mnagaku2000/igda/gate.html?cedec2006r41.zip

[13]ゲーム AI 連続セミナー第 4 回「Halo2 における HFSM(階層型有限状態マシン) 」
［資料］ http://www.geocities.jp/mnagaku2000/igda/gate.html?gameai4.zip

[14] ゲーム AI 連続セミナー第 5 回「NERO における学習と進化」
［資料］
http://www.geocities.jp/mnagaku2000/igda/gate.html?IGDA_AI_Semi_5th_2007Nov.zip

[15] ゲーム AI 連続セミナー最終回「次世代ゲームにおける自動生成技術」
［資料］ http://page.freett.com/gameboy/gate.html?gameai6.zip

[16]「エージェント・アーキテクチャーから作るキャラクターAI」 in CEDEC2007
［資料］ http://www.igda.jp/modules/mydownloads/visit.php?cid=2&lid=78
第 1～4 回のまとめとして活用できる。

[17] CEDEC2007T07「ゲーム AI を語る」資料(発言概要付)
［資料］ http://www.geocities.jp/mnagaku2000/igda/gate.html?t07_0928.zip

(e) 講演資料

[18] デジタルコンテンツ協会：“デジタルコンテンツ制作の先端技術応用に関する調査研究報告書”, http://www.dcaj.org/report/2007/ix1_07.html
2007 年までのゲーム AI の主要な仕事をまとめている（第 3 章)。

[19] Spore におけるゲーム AI 技術とプロシージャル(DiGRA Japan 第 14 回月例研究会)
http://www.igda.jp/modules/mydownloads/visit.php?cid=2&lid=77

[20] GDC2009 報告会資料
http://www.digrajapan.org/modules/news/article.php?storyid=257

[21] GDC2008 報告会資料
http://www.igda.jp/modules/mydownloads/visit.php?cid=2&lid=74

[22] 森川　幸人「ゲームと AI はホントに相性がいいのか？」
http://www.muumuu.com/other/cedec2008/index.html

(f) 論文

[23] 三宅　陽一郎　「デジタルゲームにおける人工知能技術の応用」、人工知能学会誌 Vol.23, No.1 （2008 年 1 月）
参考文献にゲーム AI の主要な参考文献をリストしている。

(g) ニュースサイト

[24] generation5　http://www.generation5.org/
人工生命や進化的アルゴリズムのニュースが豊富である。

(h) ゲーム AI の歴史

[25] 本報告書ゲーム開発技術ロードマップ「ゲーム AI の歴史」参照

これ以外にも、たくさんの優れた資料がある。特に米ではゲーム開発技術のコースが大学にあるため、そうしたサイトを探してみるのもよいであろう。また、最近では、動画資料が充実していて、google の Tech talks は技術情報の宝庫になっている。

(i) 大学のゲーム開発技術のコース

[26] Ken Forbus：395-22 Lectures ノースウエスト大学
http://www.cs.northwestern.edu/~forbus/c95-gd/lectures/

(j) 動画サイト

[27] Tech talks at google, http://research.google.com/video.html

(k) IGDA 日本　ゲーム AI 専門部会

メイリングリスト（定期的に情報を発信）と、mixi のコミュニティサイト「ゲーム AI コミュニティ」があり、筆者が世話役をしている。

[28] ゲーム AI コミュニティ（mixi）
http://mixi.jp/view_community.pl?id=3385132

(6) AI セミナーの実施リスト

最後に、参考に、AI セミナーの 135 回のログのリストを掲載しておく。実際の WEB ページでは、各セミナーで使用したファイルや資料へのリンクが張られている。数回を単位として、フェーズ毎に区切られている。初期の頃は、このフェーズの終わりに、セミナー自身について反省会と今後の方向についてミーティング「Review & Perspective」を行っていた。

表 3.2-08　AI セミナーの実施リスト

期	No.	テーマ	日付	キーワード
第17期	134	キャラクターの向きを考慮したパス経路変形について	2/11	キャラクターの向きを考慮したパス検索
	133	Company of Heroes における分隊 AI 制御	1/28	分隊制御
	132	Civilization 4 の AI	1/21	スクリプトによる UGC の AI
	131	錯視のメカニズム	2009/1/7	動的錯視
	130	囲碁における目の判定方法について	12/10	囲碁における目の判定方法
	129	遺伝的プログラミングのゲーム AI への応用	12/3	遺伝的プログラミング
	128	囲碁 AI　～基礎からモンテカルロ木検索まで～	11/19	モンテカルロ木検索
第16期	127	CEDEC 2008 における物理、モデル検証論、開発方法論	11/5	速度表現と LCP 法
	126	CEDEC 2008 と TGS 2008 におけるアバター技術まとめ	10/22	拡張現実(augmented reality)
	125	CEDEC 2008 における A.I. 系セッションまとめ(3)	10/1	全般
	124	CEDEC 2008 における A.I. 系セッションまとめ(2)	9/24	ファジー、ゲーム理論、ゲーム木、静的評価

期	No.	テーマ	日付	キーワード
	123	CEDEC 2008 における A.I. 系セッションまとめ(1)	9/17	ニューラルネット、群知能、GA、創発
	122	ゲーム開発のためのプロシージャル技術の応用	9/3	プロシージャル
第15期	121	セルオートマトンのゲームへの応用（試論）	8/6	セルオートマトン
	120	エージェントによる街生成	7/30	エージェントによる街生成
	119	街生成アルゴリズム	7/23	L-system
	118	Spore における自動生成アニメーション	7/16	プロシージャル・アニメーション
	117	Age of Empire1,2 における地形生成と自動解析	7/9	地形解析
	116	Spore エディターで遊ぼう！	7/2	ロシージャル・コンテンツ・ジェネレーション
第14期	115	ゲームと社会と人工知能（I）	6/18	個としての人工知能
	114	計算幾何の諸相	5/28	計算幾何
	113	Working Memory と長期記憶	5/21	Working Memory
	112	認識モデル ACT の源流	5/14	ACT モデル
	111	SOAR アーキテクチャー	5/7	SOAR アーキテクチャー
	110	マルチエージェントにおける学習	4/23	マルチエージェントにおける学習
	109	エージェント	4/16	エージェント
第13期	108	Halo3 の環境デザイン	4/2	レベルデザインと AI
	107	Halo3 におけるタスク配分	3/26	タスク配分システム
	106	Assassin's Creed における群集制御	3/19	群集制御
	105	Far Cry 2 におけるコンテンツ自動生成	3/12	Procedural Contents Generation
	104	Spore における Procedural Music	3/5	Procedural Music
	103	検索手法と AI	2/13	OPEN リスト
	102	ルールベースシステム（2）	2/6	後ろ向き推論
	101	ルールベースシステム（1）	1/30	前向き推論
	100	セマンティックネットからフレームへ	1/16	フレーム
	99	セマンティックネットの実装と概念	2008/1/9	セマンティックネット
第12期	98	Crysis における AI の手法	12/26	スマートオブジェクト
	97	次世代ゲームにおける自動生成技術	12/12	自動生成
	96	Spore におけるプロシージャル技術	12/5	プロシージャル
	95	Far cry Instincts における AI 製作工程	11/28	アンカーシステム
	94	人工知能と人工生命	10/31	人工生命
第11期	93	ニューラルネットとゲーム AI	10/25	ニューラルネット
	92	NERO におけるニューラルネットを一から組み立てる	10/18	誤差伝播法
	91	NERO におけるニューラルネットと遺伝的アルゴリズム	10/10	rtNEAT
第10期	90	エージェント・アーキテクチャーにおける情報環	9/26	情報環
	89	MIT メディアラボの AI 技術	9/12	Perception Tree
	88	協調のメカニズム	9/5	AI の同期
	87	ラクガキ王国のアルゴリズム	8/22	WYGIWYG（ウィジウイグ）
	86	アクションゲームの AI	8/8	アクション・プランニング
	85	ゲーム AI の双つの相	8/1	ゲームシステム AI
	84	ボードゲームの解析アルゴリズムと 3D アクションゲームの AI	7/25	モデリング
	83	ゲーム空間における適応と進化	7/18	ゲーム空間における進化
	82	生命のコーディング	7/11	コドンのコーディング

期	No.	テーマ	日付	キーワード
	81	日米の AI の差異を考える	7/4	ゲーム AI のリアリティーと演出
第9期	80	有限状態機械	6/20	有限状態機械
	79	BDI アーキテクチャー	6/13	BDI アーキテクチャー
	78	Synthetic Creature	6/6	Synthetic Creature
	77	ノンバーバル・コミュニケーション	5/30	ノンバーバル・コミュニケーション
	76	群知能とレベルデザイン	5/23	群知能
	75	階層型ダンジョン自動生成	5/16	階層型ダンジョン自動生成
	74	クロムハウンズにおけるチーム AI	5/2	マルチエージェント
	73	黒板モデル	4/25	黒板モデル
	72	Halo2 における階層化パスプランニング		階層化パスプランニング
	71	アストロノーカにおける遺伝的アルゴリズム	4/18	遺伝的アルゴリズム
	70	Forza Motorsport 2 におけるベイジアンネットワーク	4/11	ベイジアンネットワーク
	69	ザコ敵の群制御を考える	4/4	集団における知能
第8期	68	ロボカップにおける人工知能	3/28	ロールによるマルチエージェント
	67	サブサンプション制御	3/14	サブサンプション・アーキテクチャー
	66	レベルデザイン	3/8	レベルデザイン
	65	AI の基本システム	2/28	円柱を基本とする世界表現
	64	人工知能が拓くオンラインゲームの可能性	2/21	マルチエージェント
	63	プランニングを考える	2/14	行動の創発
	62	F.E.A.R.におけるゴール指向型アクション・プランニング	2/7	プランニング
	61	世界表現を考える	1/31	モーションプランニング
	60	データマイニング	1/24	データマイニング
	59	進化と共生の仕組み	2007/ 1/17	共進化
第7期	58	Review & Perspective 7	12/20	人工知能の発想
	57	Killzone における NPC の動的な制御方法	12/13	戦略的動的位置検索
	56	Halo2 における AI	12/6	知識表現
	55	意志決定とシミュレーション	11/29	意思決定
	54	行動の表現	11/22	行為の表現
	53	内部モデル	11/15	内部モデル
	52	ゲームにおける記憶の扱い方について	11/8	ゲームにおける記憶
	51	知能と時間	11/1	知能と時間
第6期	50	Review & Perspective 6	10/25	
	49	AI Middleware	10/18	オブジェクト行為の管理
	48	直感	10/11	直感
	47	確率表現のゲーム AI への応用	10/4	事後確率
	46	世界表現を記述し、AI を形成する	9/27	依存グラフ
	45	「世界表現」に注目する	9/20	世界表現
	44	インタラクティブドラマ「Facade」	9/13	ドラママネージャー
	43	「再構成」から知能を捉える	9/6	再構成
	42	自律型エージェント	8/23	自律型エージェント
	41	パターンランゲージ	8/9	パターンランゲージ
	40	第 3 者としての AI	8/2	第 3 者としての AI

期	No.	テーマ	日付	キーワード
	39	感性情報処理	7/26	感性情報処理
第5期	38	Review & Perspective 5	7/19	
	37	Chrome Hounds AI 解説 －仲間としての AI を考える－	7/12	協調とコミュニケーション
	36	戦略的パス検索について	7/5	パス検索
	35	チーム AI のプログラム	6/28	チーム AI
	34	自動生成	6/21	自動生成
	33	音楽生成と AI	6/14	パターン抽出
	32	スクリプトでどこまで AI を表現できるか	6/7	戦術性と複雑性
	31	プログラマーでない人の為のスクリプト言語	5/31	スクリプト言語の設計
	30	「人間にしかできないゲーム」をコンピュータにやらせるために	5/24	ジェスチュアゲーム、接待プレイ、Bot の判別
	29	ミドルウェアの紹介	5/17	リアルタイムモーション
	28	「人間的な AI」とは何か（ディスカッション）	5/10	予測、記憶、自律性
	27	AI を用いた緊張感の演出	4/26	自動難易度調整
第4期	26	Review & Perspective 4	4/19	
	25	最新海外 RPG に見る社会生活系 AI	4/12	社会生活系 AI
	24	AI@GDC	4/5	AI ラウンドテーブル（GDC）
	23	エキスパートシステムを用いた観光計画支援システム	3/29	エキスパートシステム
	22	自然言語インターフェイス	3/22	自然言語インターフェイス
	21	AI のためのレーストラック表現	3/15	表現
第3期	20	Review & Perspective 3	3/8	
	19	チャンス発見	3/1	チャンス発見
	18	Killzone における AI	2/22	位置評価システム
	17	プレイヤーの思考のモデル化による適応学習	2/15	適応学習
	16	リアルタイムプランニングのためのエージェントアーキテクチャ	2/8	エージェントプランニング
	15	Navigation Mesh による遮蔽領域探索	2/1	ナビゲーションメッシュ
	14	AI のデバッグ機能	1/25	AI Debug 機能
	13	Chrome Hounds AI 概要	1/18	プランニング
	12	ゴール指向 AI について	2006/ 1/11	ゴール指向型 AI
	11	Halo2 の AI について	12/21	HFSM
第2期	10	Review & Perspective 2	12/14	
	9	スクリプトによる AI 表現 ～Baldur’s Gate II の場合～	12/ 7	スクリプトによる AI 表現
	8	Quake3 の Bot Architecture	11/30	階層型 Architecture
	7B	アーケード AI	11/16	アーケード AI
	7A	Area Awareness System(Quake3)	11/ 9	AAS
第1期	6	Review & Perspective 1	11/2	
	5	錯覚（錯視）	10/26	錯視
	4	学習	10/19	学習
	3	創発	10/12	創発
	2	Blackboard Communication	10/5	Blackboard Architecture
	1	アフォーダンスとは何か？	9/28	アフォーダンス
	0	ゲームにおけるゲーム理論	2005/ 9/1	ナッシュ均衡

4.10　ゲーム開発技術の歴史についてのヒアリング

4.10.1　タスクシステムについて黒須一雄氏インタビュー

タスクシステムを中心に、黒須氏のプログラミングのバックグラウンド、80 年代から現代に至るゲームテクノロジーの推移についてお話を伺った。

黒須一雄。昭和 31 年生まれ。1979 年ナムコに入社。「ラリーX」「リブルラブル」ファミコン版「ゼビウス」をはじめ、歴史的にも名作と言われるタイトル製作と移植を行う。1985 年ナムコを退社。現在、株式会社モバイル＆ゲームスタジオ　執行役員、プログラマー。

(a)　黒須氏のプログラミングのバックグラウンド

1.プログラミングを始めたきっかけとは？

――　はじめに、タスクシステムに入る前に一般的なプログラムの話をお伺いします。黒須さんがプログラマーになられた理由などをお話し下さい。

最近、同人やプロのゲーム開発者の方にヒアリングさせていただいているのですが、全員とは言わないまでも 8～9 割は「どこでも学んでいません」とおっしゃられます。子供の頃に MSX を使っていじっていたとか、親に買って貰ったパソコンで覚えたとか、そういった独学で学んで来た方が多いです。逆にどういう世代までそうなのか、それとも最初からそうなのか、日本はそういう国なのか……。

黒須　私は昭和 31 年生まれで、大学に情報コースはありましたが、行ったところは電子工学科、電子コースなので、どちらかというとハードウェア寄りの出自になります。一応コンピューター工学、当時の名前で電子計算機工学というのがすでにありまして、はじめてプログラムを書いたのは多分大学 2 年ぐらいです。最初に書いたのはやはり BASIC からです。

――　FORTRAN ではなくて、BASIC からですか。

黒須　FORTRAN は大学 3 年の時に情報演習で取っています。その後 4 年で研究室に入って、そこではミニコンのアセンブラをずっとやっていました。

――　ミニコンですか、時代を感じますね。

黒須　ですから、今のゲーム業界にいる人たちと大きく違うのは、ゲームが好きで入ったわけではなく、プログラムが面白くて、それが活かせる場所の一つとして入っています。

――　ハードウェアではなくプログラムのほうが好きだったと？

黒須　そうです。ハードウェアが好きで大学は選んだのですが、大学に入ってからソフトウェアのほうが面白いのではないかと。一応ハード系のことも学んでいますが。その頃にナムコに入社した私たちの代までは、先輩社員からのアドバイスを受けながらハードの設計からソフトを書いて出していくところまでやっていました。

2.ナムコ入社時代の開発状況

――　黒須さんぐらいまでの世代がゲームを作るということが…

黒須　ハードウェアからはじまって、ソフトを作って出すところまでです。それで、私の同期が途中からハード専門に、私はソフト専門に移っていく時ぐらいに遠藤（雅伸氏、株式会社モバイル＆ゲームスタジオ代表取締役会長）が入社してきました。その時にはソフト部門とハード部門が完璧に分かれていました。ハード設計の人たちがいて、それに基づいてソフトを書く人たちがいるみたいな形に分かれていました。

――　黒須さんのナムコ入社は何年の頃でしょうか？

黒須　1979 年です。大学 4 年の時に就職が決まって、それから少ししたらインベーダーが流行っていた時代です。

――　入社して最初に関わられたタイトルは何でしょうか？

黒須　ナムコの場合、ゲームセンター自体がもともとの基盤ですから、入社直後はまずそこに必ず配属されていました。今はどうか知らないのですが、当時はどの部署に行く人も一度はそこに行って、どういう形でお客さんがやっていて、どういうふうにお金が入っているかというのを目で見て学ぶみたいなことを学んで、そこは最初の 1 か月ぐらいで、次に長い期間「もうひとつ違うことを経験しましょう」と、ゲームセンターに入る人も製造部の手伝いをするという形になっていて、私はその時にサービス部門、ゲーム機を修理する部門に配属されて、そこで修理の手伝いをしていました。それから半年ちょっと経ってから、やっと開発部に配属されたという感じです。

入社して約半年そういう形でやっていて、その後の半年ぐらいを使ってキャラクターエディター用の……当時はパソコンではなくてマイコンと呼ばれていた時代ですから、そんなにいいものはなくて、パレットとかも付いていなかったので、社内でパレット付きのものを作って、そのソフトを書くのを半年ぐらいやっていました。

―― パレット用のメモリーというか…

黒須　当時ですと 8 色、良くて 16 色しか出ないコンピューターしかなかったですからね。下手するとグリーンモニターしか付いていないという状態でして、実際のゲームセンターに出すマシンの場合ですと色が付いていますから、それを作る時にもともとハードウェア的にパレットを用意していたので、それに合わせた形でやっておかないと同じものが出せないという感じでした。

―― では開発用機材を作るところからスタートしていたわけですね。

黒須　そうですね。当時は作っていました。その後にラリーX に入って。

―― ラリーX では、黒須さんがメインプログラマーを担当された？

黒須　そうです。ナムコにもまだそれほど人数がいなかったので、同期入社社員のうち私がまずラリーX に入って、あと 2 人いたのですが、その 2 人もやはり似たようなことをはじめていました。その 2 人は手伝いで入っていたかもしれません。随分前のことなので、曖昧なところがありますが。

―― どのぐらいの時期まで、黒須さんが開発用の機材まで作っていたのでしょうか？

黒須　開発の機材といっても結局はキャラクターを作るほうで、プログラムに沿ったものは当時では Z80 を使った開発用の機材が存在していて、CPU エミュレーターも付いているものがありました。ただシステムがまだフロッピーだった時代で、ラリーX は全部それで作ったので、その次のボスコニアンを作っている最中にヒューレット・パッカード社の HP64000 が出てきたわけです。

―― 設計をする中である程度、専用機材を使わずに PC とか、もしくはエミュレーターで開発できるようになった時期とはございますか？

黒須　最初からエミュレーターはついていたのですが、その次の HP64000 はもっと良いエミュレーターが付いているマシンでした。

――　キャラクターなども全部エミュレーターで作れるようになったのでしょうか？

黒須　キャラクターは違うところで作ったやつをロムで乗せるので、プログラム自体はそれで作るということです。

――　それがボスコニアンの頃から……

黒須　ボスコニアンの頃から、しばらくはそれでした。そうですね。1985 年にナムコを辞めていまの会社をつくった時も同じ機材を入れたぐらいなので、しばらくは使っていました。

――　かなり色数が多かったので、メモリーをたくさん食うような、パレットを使うものは当時では、やはりかなり扱いにくかったですか？

黒須　最初から使っていたので特に扱いにくいとは感じませんでした。

3. ゲーム開発技術の変化について

――　かなり文脈は変わるのですが、そういったアセンブラとかがメインの時代から…今のアーケードはわりと Windows マシンですよね。そういったライブラリーを使い倒すような形で、プログラミングの手法自体が変わった時期はいつ頃でしょうか？

黒須　プレイステーションやセガサターンが出た時代、1993～1994 年ぐらいでしょうか。スーパーファミコン時代はまだアセンブラでしたから。ただ、その頃に、一気に全部変わったわけではなく、ゲームボーイはまだアセンブラで残っていましたし、仕事によってだいぶ違っていました。

――　アーケードも同じ時期ですか？

黒須　アーケードは、1985 年の時にひとつだけ、それをやったのが最後で…。

――　基本的にはプレイステーションやセガサターンの時に、プログラミングの仕方というか、手法みたいなものが変わったということでしょうか。

黒須　言語が変わっただけなので、手法的にはそれほど変わっていないと思います。

――　3D が入ったことで、会社として採用される人員は変わりましたか？

黒須　いえ、確かに 3D モデルができる人が今は何人かいますが、採ったのはだいぶ後です。プレイステーションやセガサターンの頃からは、ずいぶん後ですね。セガサターンでは普通に今まで 2D グラフィックをやっていた人が 3D をはじめた状態で、そんなに高い機材も使っていなかったですね。

――　それでは、プレイステーション 2 の頃になると、かなり変わってきたのでしょうか？

黒須　プレイステーション 2 の時代はあまり携わっていなかったです。会社としては何本か制作をしているのですが、多くは社外スタッフを使っていました。ただ、最近は意外と携帯電話用に 3D の人を採ったりしています。

――　最近は C++になって、オブジェクト指向になっていますが、そういうパラダイムというのは……。

黒須　それは別にないですね。もともと私自身は C、 C++嫌いなので……。

――　もっとアセンブラ寄りのほうがお好みなのでしょうか？

黒須　いえ、そんなことはなくて、あの言語はアセンブラと高級言語の間なので……。

――　Java ということでしょうか。

黒須　Java とか、もともと私は Pascal 使いなのです。Pascal、Delphi、の道を歩んできた人間なので、Delphi 自体がオブジェクト指向でしたので、 Pascal もですが、オブジェクト指向自体にはそれらが出てきた時点ですでに触れていました。業務としては、2000 年くらいに PC 向けのパズルゲームを Delphi で書いています。

(b) タスクシステムの歴史

1.タスクシステムとは何か？　－　ジョブコンとオブジェコン　－

――　まずタスクシステムの定義からお伺いしたいと思います。

黒須　今はもう亡くなってしまったナムコの先輩に深谷正一（ふかたにしょういち）さんという方がいて、新人が増えてきたときにそれぞれが勝手にやられても困るということで「こうやると簡単に書ける」というシステムを作っていただいたのがきっかけですね。それがジョブコンと呼ばれるシステムで、そこの上で動くものをオブジェコンと呼んで、それぞれが独立して動いているような、本当に完全な疑似マルチタスクなのですが……その状態ではじまりました。開発の社員を集めて深谷さんが講習をする、ということがありました。

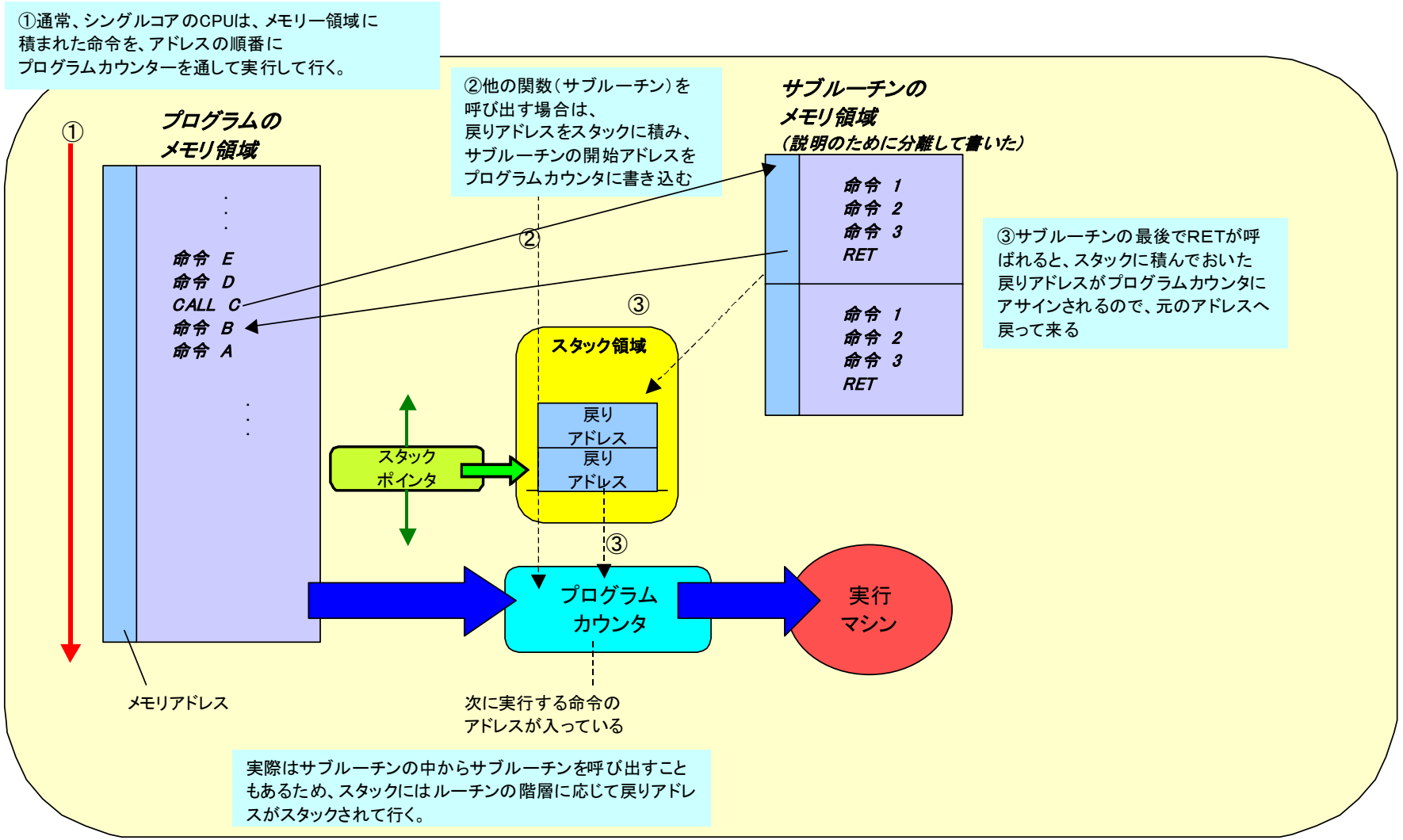


図 4.3-01　通常のプログラム

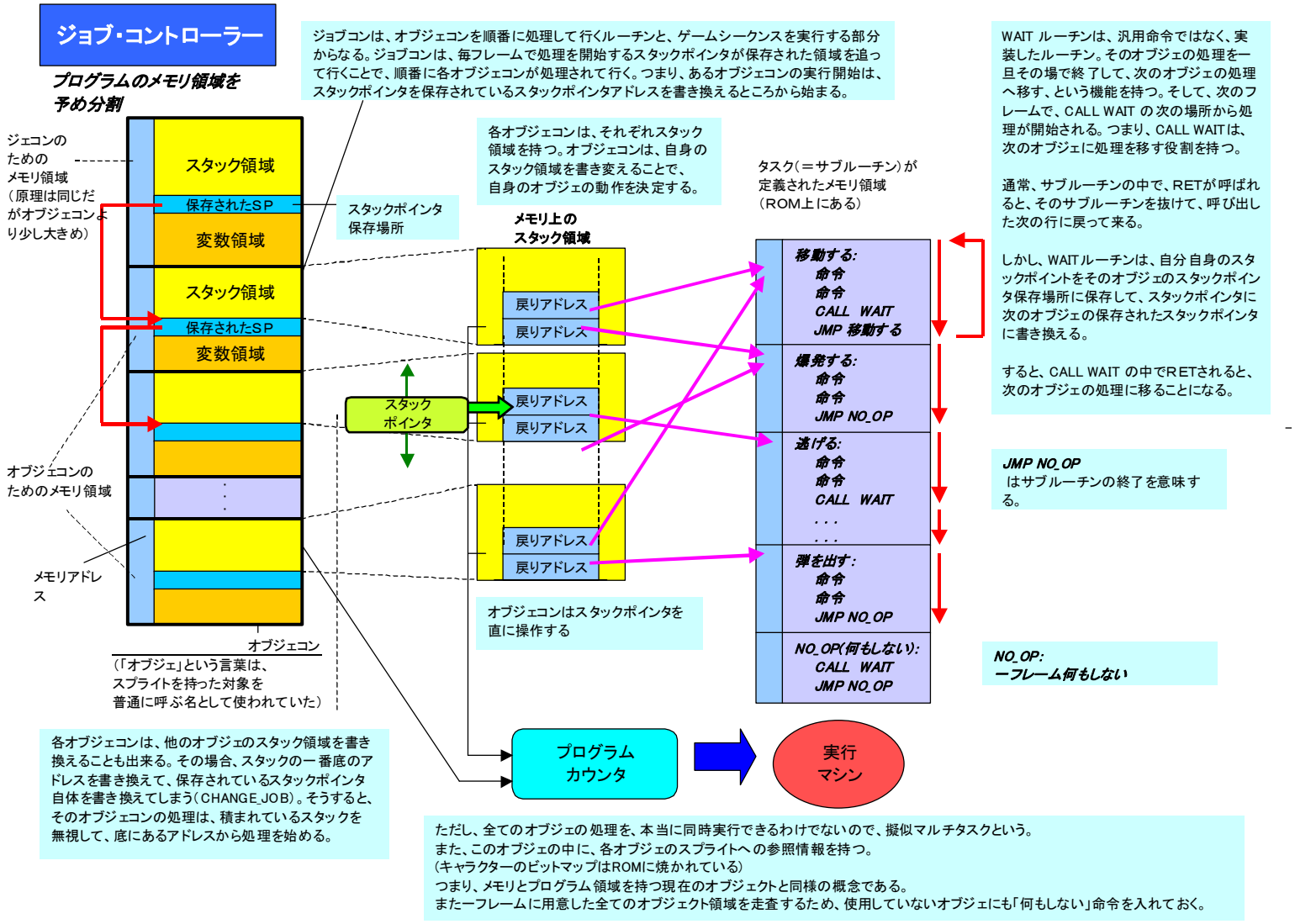


図 4.3-02　ジョブコントローラー

オブジェの処理シークエンスのイメージ

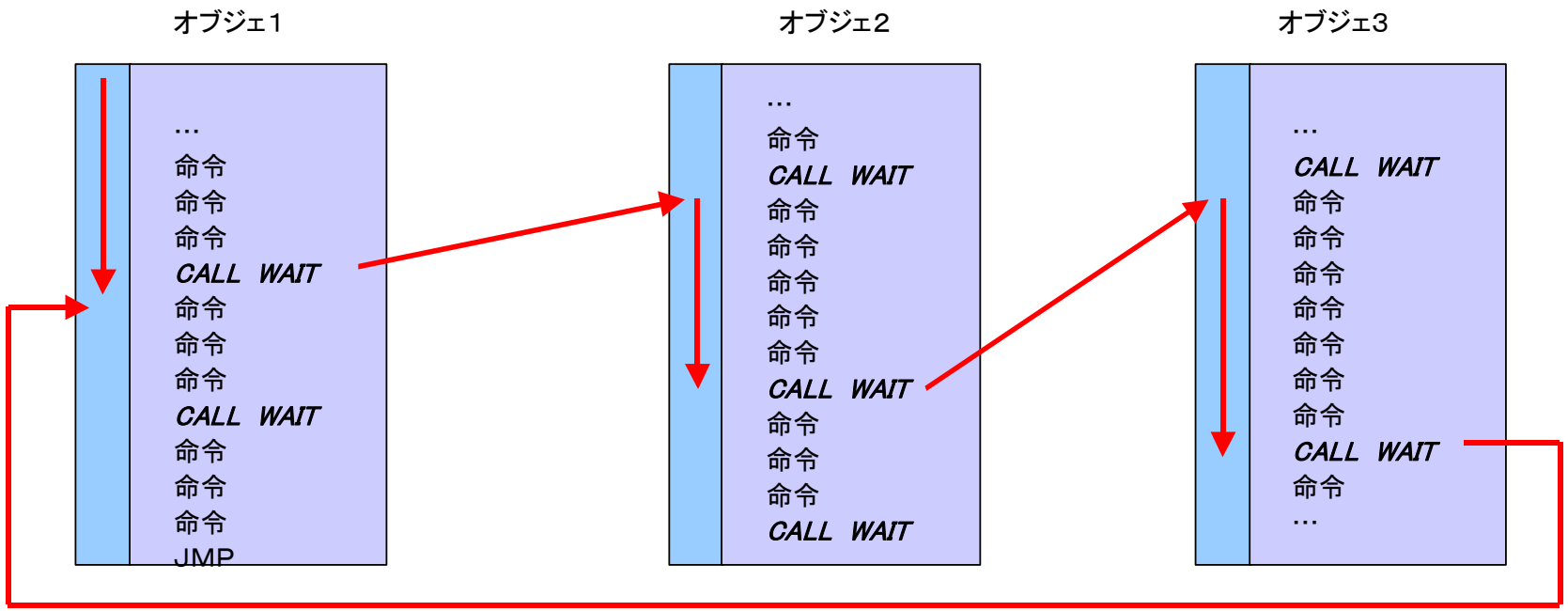
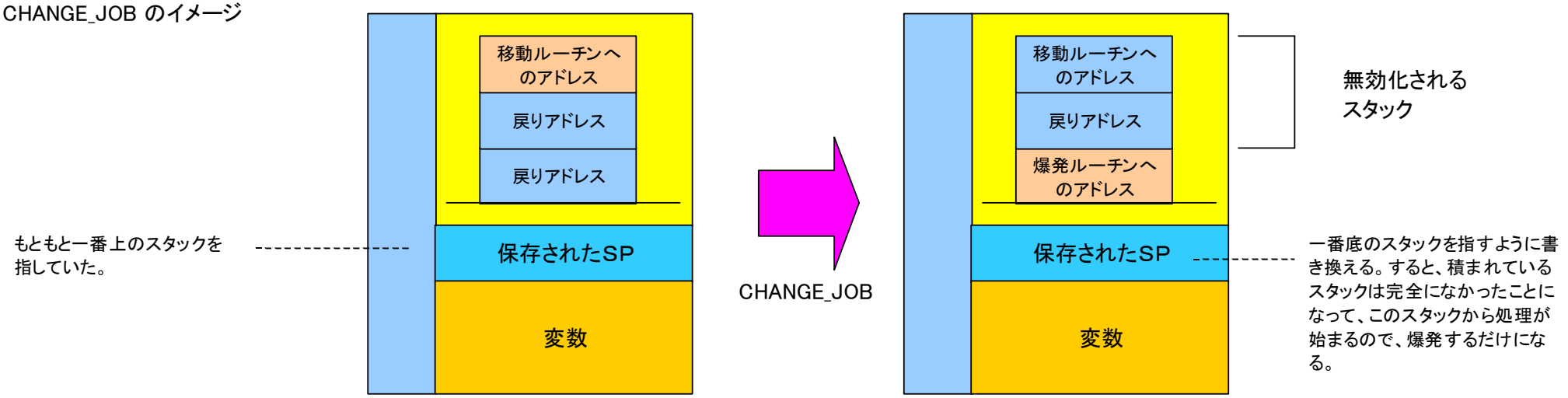


図 4.3-03　オブジェの処理シークエンスのイメージ

CHANGE_JOB のイメージ



「CHANGE_JOB」は、自分ではなく、他のオブジェクトのジョブを書き換えるための命令である。
例えば、「弾」というオブジェクト自身が、衝突判定ルーチンや、スコア書き換えルーチンを呼び出している。
弾は衝突したオブジェクトを爆発させるために、そのオブジェクトのスタックの底を爆発ルーチンを呼ぶように書き換えて、
さらに、処理を始めるスタックポイントを書き換える。すると、そのオブジェクトの処理が始まると、ただ爆発するだけになる。

図 4.3-04　CHANGE_JOB のイメージ

―― ジョブコンの上でオブジェコンが動いているということですか？

黒須　そうですね。ジョブコンはシーケンス系のコントロールなどをしています。当時は、動くもの自体をオブジェクトと、一般的にスプライトと呼ばれているもの自体をオブジェクトと呼んでいたので、「オブジェクトコントローラー＝オブジェコン」という名称で、もう完璧に CPU のアセンブラの世界で、スタックポインタ自体を変更してしまいます。

プログラムの書きかたの中で、普通は一つのスタック領域というのが存在していて、プログラムが動いています。そういう形ではなくて、それぞれのオブジェクトごとに別のスタックがあるような形にして、別々のものみたいな形で仮想的にタスクをいっぱい持てる。メモリーもそれ用に割り当てて、スタックもそれ用に割り当てて という状態です。ひとつしかない CPU を、まるで沢山同時にあるように見える形にしてやっていました。

―― プログラムもメモリーもひとつずつ…

黒須　実際には、今の CPU ほどアクセス制限とかがないので、他のオブジェクトも平気で見ることができるし、壊せる状態でした。

―― 各キャラクターに動作スタックを持たせて……

黒須　要はこれが終わったら、この WAIT のルーチンの中で次のオブジェクトのためにスタックを変更する事で次のオブジェクトの処理を行います。

2.オブジェコンとジョブコンの関係

―― オブジェクトと呼んでいるのはキャラクターだけなのですね。

黒須　オブジェコンは動くものに関してであって、シーケンス自体はジョブコンという名前でまた別にあります。ジョブコンとオブジェコンの差は、オブジェコン自体はそれほど深くまでスタック使わなくていいだろうということで、メモリーもそれほど多くなかったので、少なめのスタック領域で。
それに対してジョブコンと呼ばれる部分は、スタックも少し多めで色々なシーケンスができるわけです。

―― そのオブジェクト同士は、他のオブジェクトのスタックの中に自分の何かを書き込めるわけですか？

黒須　CHANGE JOB でオブジェクトのジョブ自体を変えることもできます。

――　当たり判定とかで衝突したら書き換えるとか……。

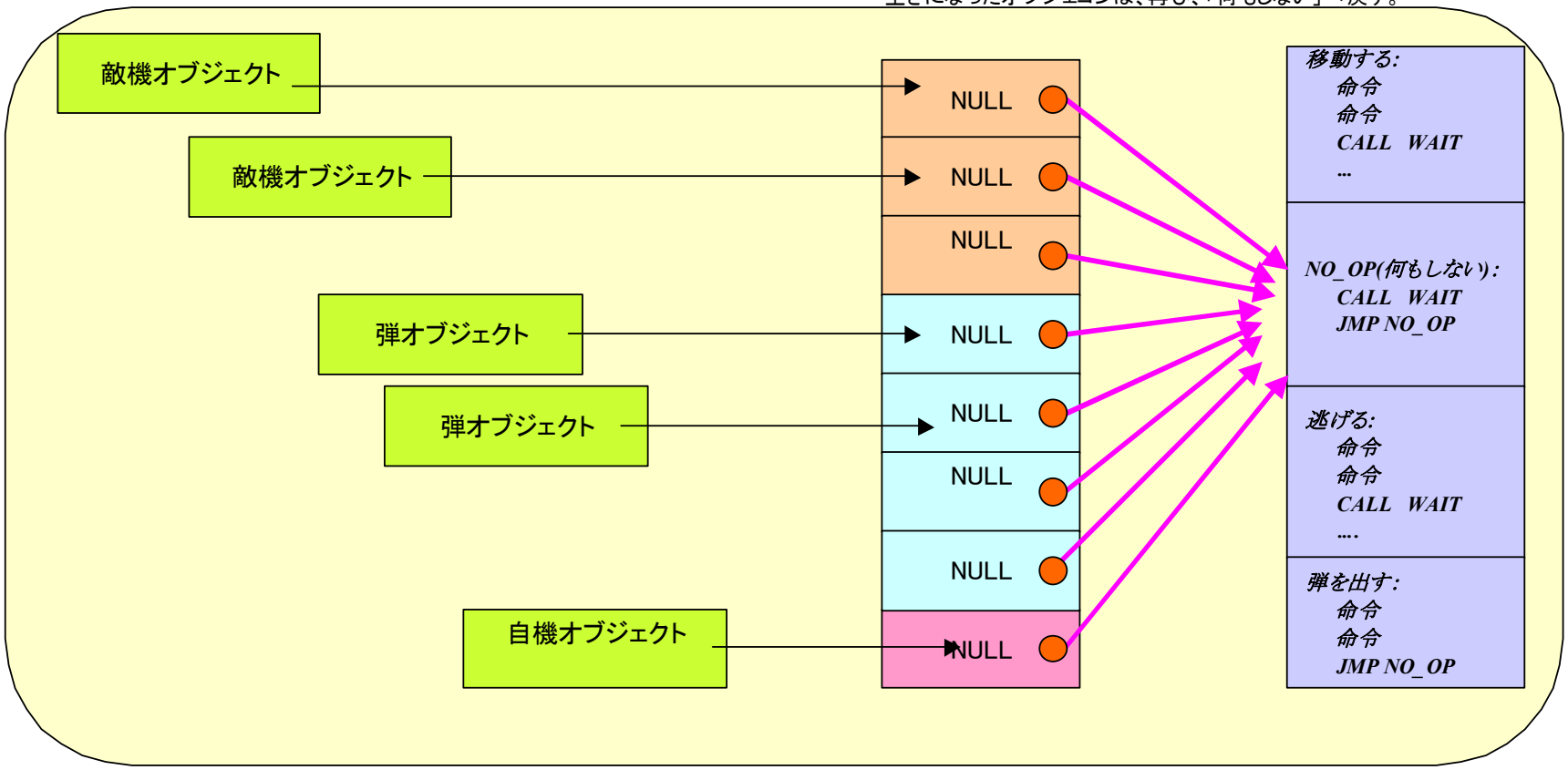
黒須　そうですね。逆に衝突して爆発ではなくて、飛ばされるとか、そういうように変えてしまえばそれは飛ばされていくという形なので、実はそれぞれの部分さえバグがなければ、バグが入りにくいです。だから、はじめて作る人にとっても、逆にこちらのほうが普通にだらだらフラグでやられるよりバグが入りにくいのと、その人が困っている時に助けやすいですね。「ここの部分でおかしくなった」と言ったらそこだけ見れば済みますから。だから「逃げるとここでバグる」といった場合に、今までのフラグ方式だとそれがフラグに合わせて処理が飛んでいるので、どこでやっているのか、長いソースの中から、ここの部分で間違っているとか、それを探す作業が必要になってきます。そういうことを考えるとこのジョブコンやオブジェコンがあると、フラグ方式では「逃げる」に行ってなかったという可能性が出るのですが、それがなくなります。

――　全体のシステムは、メインルーチンで動かすのですか？

黒須　そうです。ジョブコンというものが存在していて、ここにシーケンスがあって、例えばタイトルを出すというものだと、これ自体がオブジェクトを生成します。
実際にはオブジェコンは使われる最大数を用意しておいて、それぞれに何もしないというジョブを書いておきます。つまり呼んだらすぐに回っている状態です。何もしないというジョブが全部に書いてあると、それ自体が少しは時間を食います。

あらかじめ確保されたオブジェコン･リストの領域へ、
ジョブコンがオブジェを必要に応じて登録して行く。

空のオブジェコンのための領域には最初は
「何もしない」というルーチンが指定されている
オブジェコンが登録されると、参照ルーチンを変更することで、
動作が変化して行く。爆破などされて、
空きになったオブジェコンは、再び、「何もしない」へ戻す。



空きオブジェコンには、「何もしない命令」が登録されている。
オブジェコンが登録されると、そこに命令が上書きされる。

図 4.3-05　オブジェコンのイメージ

―― 呼び出された限り自分のルーチンの中でずっと回っている。例えば弾が飛んできた判定というのは？

黒須 それは弾自身がやります。弾自身もオブジェクトなので、弾自体がキャラクターとぶつかって、衝突判定する、それ自身がぶつかったかを判断して、ぶつかったものを爆発というふうにチェンジジョブしています。地上用の弾はまた別で、地上用のオブジェクトとして存在しています。
また、プレイヤーが弾を撃つ度に弾の新しいオブジェクトが増えます。衝突判定は弾が当たるか、端までいくと消えます。

―― 弾の数自体が増えていくだけですか？

黒須 そうですね。画面に出る全部のオブジェクトのオブジェコンが存在していて、それぞれ動作する形で、その代わりそれぞれは小さくて済むのでバグが入りにくいということです。そもそもマルチタスクがやりたくてはじめたわけではなくて、多分そういう考え方がシンプルになるので人が見やすいし、バグが入りにくいという形ではじまったというか、教えていただいたのかと思います。

3.ジョブコン、オブジェコンの起源
―― それは深谷正一さんのオリジナルなのでしょうか？

黒須 そこは本当にわからないです。深谷さん自身はかなりすごい人ですので色々なものを見ていたかもしれないし、考えたかもしれません。少なくとも、ナムコでは、深谷さんからはじまったということは間違いないです。

―― 当時、深谷さんが UNIX などのタイムシェアリングの概念をご存じだったのか…。

黒須 それは普通にコンピューターをやっている者は知っていることですから、それはわかっていました。おそらく、その応用ができないか、という側面と新人がいっぱい増えた時のことを考えた側面とのふたつではじまったのではないかと思います。

―― それまではフラグ管理という形が多かったのですか？

黒瀬 そうですね。普通にフラグでループさせて、それぞれフラグを見ながら処理するという感じです。

―― それ以降はルールみたいなもので、社内では絶対にこう作ろうといったものはありましたか？フラグに戻りたいという方もいたのでしょうか？

黒須　当時、深谷さんは神様と呼ばれていたプログラマーですから、大抵の人は深谷さんが言うと「そうだよね」と思う人が多かったですね。

―― もう大きな流れとしてそういう…

黒須　そうですね。

―― 深谷さんの名前というのは、ナムコの外でも広まっていたのですか？

黒須　源平討魔伝の最後に出てくる名前なので「亡き深谷正一さんに捧ぐ」と名前が出ていたと思うので、知っている人は皆さん知っていると思います。

4.ジョブコン、オブジェコンの実装スタイル

―― 次に、オブジェのリストは、動的というよりはマックスの箱をまず用意して…

黒須　そうです。最大使うものを用意して、それぞれ何もしないというのを入れてある状態で動いてはいます。そこを書き換えていくわけです。

―― 使い終わったオブジェクトの回収はどうしていたのでしょうか？

黒須　終わった時点で何もしないというジョブになります。新しいのをやる時にその何もないというものがあったらそれに入れます。だからどこに入っても構わないのです。ですからある意味では動的なのですが、実際には内部的には取っておかないといけないので静的になっています。

―― タスクシステムで、1990 年代によく使われていたものでいいますと、何でも一律に全部ジョブとして入れるというものがありましたが……。

黒須　ひとつは、ジョブコンという種の流れがあって、そこからオブジェコン自体を生成します。ジョブコン自体は消えないです。

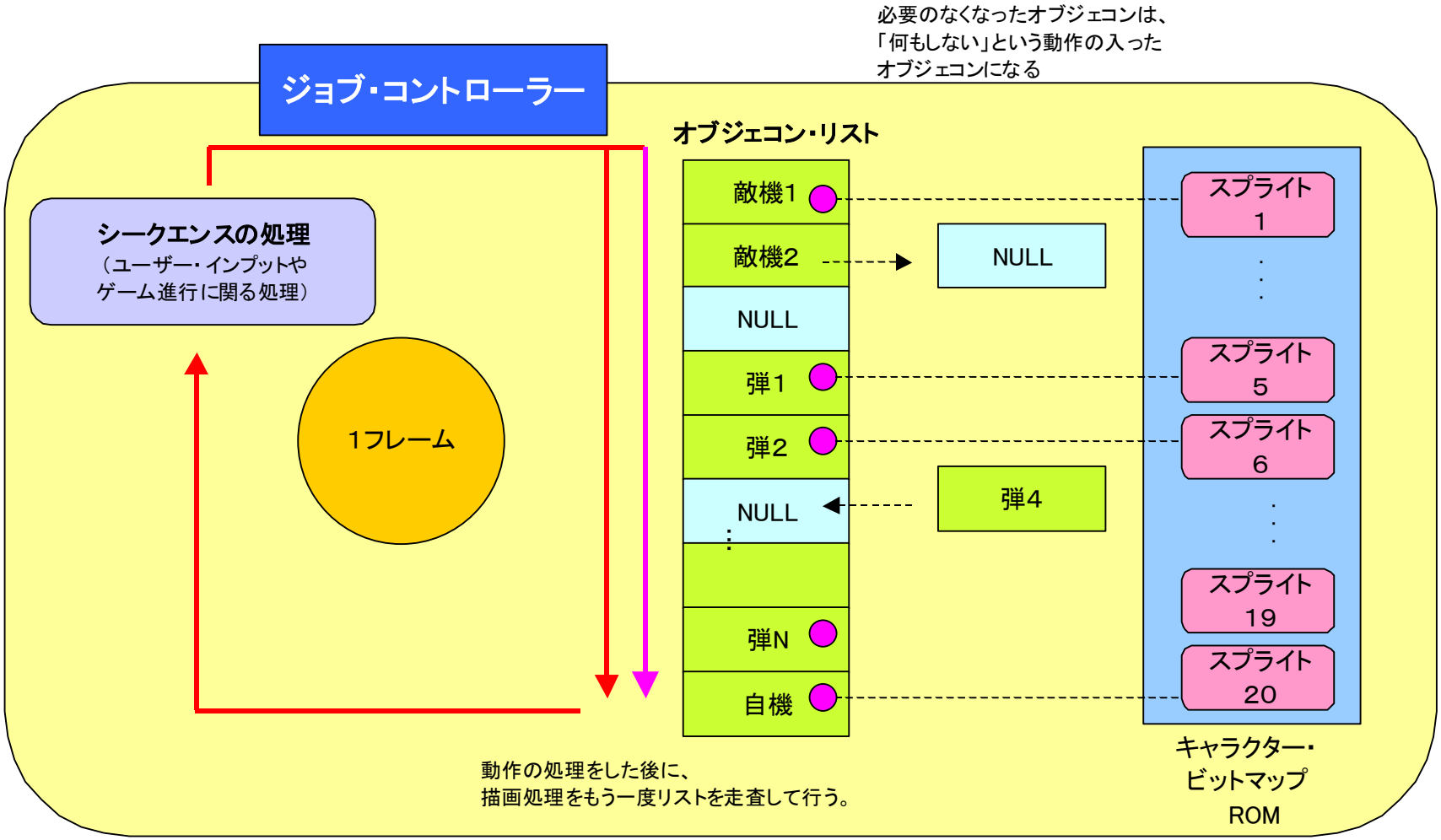


図 4.3-06　ジョブコントローラー

―― 例えば、この派生系みたいな形で、ゲームシステムが複雑になってきた時に、AIのように中長期的にある程度働くためだけのオブジェコンを作っておいて、キャラクターはキャラクターだけで作っておくといったものを作る。そこに構造的にノードをはって、ツリー構造で管理することを実装している現場もあるのですが……。

黒須　それほど、AI はなかったので……。実際にそんなことをやっていたら 60 フレームで回らないですよね。60 フレームも Z80 の時代です。とは言ってもゼビウス自体はどう分けていたか覚えていないのですが、CPU が 3 つ乗っているので、実は結構大変でした。
1 個はサウンド用に使いますが、残り 2 個でわけてやったり。

―― そこに自由度があるわけですね。今で言うマルチコア的な。それで今のオブジェコンを回すのですか？

黒須　そうですね。

―― 各 CPU にジョブコンがあるわけですか？

黒須　おそらくそれぞれあって、メイン CPU とサブ CPU があって、サブ CPU 側にメイン CPU から受けたものに対して、ジョブを発行していたのではないかなと、もう二十数年前の話なのではっきりとは覚えていないのですが……。

―― CPU が 3 つというのはアーケードのほうですか？

黒須　そうですね。ファミコン版を私が担当しました。ファミコンは CPU が 1 個しかないので、そういうわけにはいかない。ファミコンの場合だと、MOS 6502 という特殊な CPU なので、先ほどのような手法は使えなかったんです。
MOS 6502 という CPU 自体は、スタック領域が 256 バイトしか存在しない CPU なので、ここはタスクシステムとは言えないような、多分ジョブコンだけ存在している状態でオブジェクト自体は普通にフラグ管理でやらざるを得ませんでした。

―― そのオブジェコンの考え方は、アーケードで脈々とずっと繋がっていくという感じですね。

黒須　そうですね。

―― ファミコンはそれがハードウェア的にできないので…

黒須　まあ、それっぽい書き方はしますが、やはりアーケードほど自由度は高くないです。スタックが一つしかないために、似たような処理でここだけ回って戻ってくるということはやりにくいので、どうしてもそれぞれがフラグでシーケンスになってくる状態になります。

――　当時はアーケードのほうがスタック領域を使えたわけですね。

黒須　アーケードもそんなにメモリーが多かったわけではないですが、Z80 の場合は 64 キロバイトの間にメモリーさえ乗っていれば、それだけ使えました。

――　次の質問ですが、ループ型というのは、これは今言われたテーブル、マステーブルを上から……

黒須　そうですね。順次やるループ。そういう意味ではループ型になっていますね。

――　次の質問、レバー入力、これはもうジョブコンのほうですね。他のインタラクションとか…

黒須　そうです、そちらで取っておいて。描画自体は、オブジェコン自体に何を表示するというのを書く場所があるので、それを見ながら描画するみたいな…今ならそっち側で描画してもらうのですが、そこの情報を見ながら描画するだけなので、まあジョブコンで描画しているようなものです。また、描画するための情報自体はオブジェコン側に持っています。ジョブコン側は何も持っていなくて、それでオブジェクトが色々なものを持っているので、それを見ながら描画します。それぞれ「何を描画してください」みたいな情報がオブジェクト側にあるわけですね。
　1 回ジョブが回って、オブジェクトを全部動かした後に描画するみたいな感じです。それぞれを描画するわけではなくて全部終わった後にそれを描画します。

5.　「タスクシステム」と「ジョブコン - オブジェコン - システム」の違い

――　タスクシステムはナムコの中で開発原理のとして導入されて行ったのでしょうか？

黒須　私が辞めるまではそうですが、それ以降はハードウェアも変わってきているので、どんどん色々な形に変わっていったとは思います。

――　黒須さんが把握されている限りで、タスクシステムのバリエーションというのはどれぐらいあるのでしょうか？

黒須　いや、わからないです。プログラマーの数だけあるのではないでしょうか。

――　タスクシステムという用語は当時ナムコで使われていたでしょうか？

黒須　先程言った通り、ナムコで使われていた言葉はジョブコンとオブジェコンなので……ですからタスクシステムだと思って使っていない人もいた可能性があります。

――　黒須さんは、タスクシステムという言葉を初めて聞いたのはいつぐらいでしょうか？

黒須　大学時代にハードウェア系のコースを取っていますが、コンピューターもある学校ですから、ゲームの文脈の上で今言われているタスクシステムという使われかたとは少し違うところ、タイムシェアリングの概念になると思います。
　ただ、これをやっている時点で、呼び方がジョブコン、オブジェコンという名前だっただけで、それは疑似マルチタスクだといってやっていたのは確かです。

6.　ジョブコン-オブジェコン-システムの速度

――　タスクシステムは先程言われたように、最初、新人の使い方として説明されていたとのことですが、速度はどうでしょうか？

黒須　速いです。フラグですとフラグ自体をどう比較して分けるかとか、いくつかやりかたがありますが、それよりも CALL WAIT と呼んで次に戻ってくるだけなので、とても速いです。フラグを比較してフラグによって分けるという作業自体は結構時間がかかりますから。

――　これは埋め込まれたものにジャンプするだけだから、比較がない分速いという感じでしょうか。

黒須　というか、ただのリターンなので、分岐は中にあるのですが、それはそこの条件で、この 1 フレームの間にやることを決めて、行った後では確実に CALL WAIT で次のフレームまで何もしない、そこから戻ってくるだけです。それに対してフラグシステムだと、フラグを書き変えて戻って、次の時にそのフラグを見ながら何をするかという判断をしなくてはいけないわけですから。
　そのフラグに対して、大抵テーブルジャンプを使うのですが、それは当時だとテーブルジャンプの計算をして飛ばすよりもリターンのほうが早いので、フラグをしまう必要もないし、それがオブジェクトの数だけあるということを考えると十分に速い。作り方によ

っては遅くなる作り方も存在したと思うのですが、そこで使っていたジョブコンとオブジェコンが速いシステムだったということです。

―― 他のプログラマーから見たプログラムの可読性はどうでしょうか？

黒須 ひとつずつモジュールみたいなもので、きれいに分かれているので、こうラベルがあって、WAIT までを見ればそのフレームで何をしているかというのが確実に解ります。そこが最初からの目的のひとつでもあったのかもしれません。プログラマーが困っているのを助けるにもそのプログラムを見なくてはいけない、その時にそのほうが楽だというのはあります。

7. ジョブコン-オブジェコン-システムと開発体制

―― 当時は黒須さんが、常に各開発の後ろにいるという感じだったのですか？

黒須 先輩社員が後輩社員を見るの、ごく普通の世界です。

―― 当時ですと何人ぐらいが黒須さんの下にいましたか？

黒須 当時は本当に倍々みたいな感じで増えていったので、うちらの同期が何十人に対して、ひとつ下は 100 人ぐらいいて、開発に来る人の数も多かったです。

タスクシステムにしたほうが大規模開発には向いていると思います。それぞれが責任を持って作ってもらえれば大丈夫という構造にしてあればいい。ただし、タスクシステムを使いながら設計が悪いと駄目になります。

―― ではタスクシステムでやるとして、最初の初期設計というのはどなたがやるのでしょうか？

黒須 初期設計はある程度、当時のゲームセンターのものですと大体流れが決まっていました。

―― サンプルルーチンみたいなものがあったのでしょうか？

黒須 ある程度はあって、それぞれ実際にやる部分は違いますが、最初は RAM テストみたいなことをやって、ROM をテストやってみたいなのがあって、それでタイトルが出るみたいな感じです。タイトルからは、コインが入ったらスタートしてという一連の流れはほぼ決まっているので、それぞれのゲームの部分だけを考えればよいという形でした。

―― RAM テストというのはメモリーをサーチして行くのですか？

黒須　そうです。ランダムパターンみたいなもので書いて……全メモリーチェックは入れて、駄目だったらエラーを出して止めるということを当時はやっていました。

―― 企画さんとかグラフィッカーさんのとの連携ではタスクシステムの利点はありますか？

黒須　それはどうだかわからないですね。グラフィックの人も企画をやっている人も、それは知らないことで、プログラマーだけの話でした。

―― ナムコの当時の企画というのは、プログラムを理解されていたということはなかったのでしょうか？

黒須　理解しているかどうかは微妙です。別に理解する必要はあまりなくて、既に分業が確立されていたというところもあります。

―― バグが出にくいということですね。

黒須　出にくいです。

―― ハードウェアとしてハイスペックになればなるほどタスクシステムは向いているといったことはありますか？

黒須　いや、全くないです。どちらでも一緒だと思います。

―― タスクシステムは、色々な今のゲームシステムの原型になっていると思います。今の色々なゲームアーキテクチャがあると思うのですが、それを見て、これだけはオブジェクトコントロール系譜によく似ているとか、あるいは C++のクラス自体が言ってみればメモリーとプログラムが乗っているようなものですから…どう発展してきたと見られますか？
黒須　今はわざわざ呼ばなくてもいいぐらい、普通に使っていることなので、だから若い人たちはあまり意識していないと思います。

―― タスクシステムのジョブコンとオブジェコンはシステムを回してオブジェクト制御をするという発想だったと思います。今で言う「ジョブをスレッドという形で割り振って

動作させることで、ひとつの大きなシステムが動く」という当時の概念だったと思うのですが、どういう発想から生まれたものなのでしょうか。

黒須　オブジェコンだけで考えればいいシステムになっているので、こういうことをそれほど意識しているわけではなく、本当にそれだけで済むという、シンプルにしていく方向の手法として多分入ったのではないかと思います。
　フラグでやっていると、どうしても複雑になります。フラグによって、私は何というのをまずやって、そこからそれぞれがどういう処理になるのかに対して、オブジェコンであれば「最初にここで何を出しなさい」と言った時点で「私は何」というのをこの人は知っているわけで、そこを比較すらしないわけです。要は考え方が本当にシンプルになることによって、バグが減っていくだろうというのがベースだったと思います。

――　そのオブジェクト自身が何をするかということだけを判断すれば、影響はオブジェクト間の処理だけで済むということですね。状態推移をチェックするのではなくて、動くもの自身が何をするかということだけに収めようということですね。

黒須　収めることで、とてもシンプルになります。多分そこが中心だったので、別にタイムシェアリングとかそういうことは、それを見ながらこういうのは応用できるというように深谷さんが考えたと思うのですが、使っている人たちはそんなことは全然意識せずに使っていました。
それは実際に、使い勝手が良かった。ですから、オブジェコン自体は搭載できなくても、ジョブコンの部分に関しては大抵のものに入れています。

――　結局は、そのジョブコンから何をするかという先の話だから、ジョブコン自身はどこに持っていっても同じ動きをするという概念からはじまっているということですね。
黒須　そうですね。それで、できればオブジェコンも実装すればそれぞれがシンプルになるということです。どんなものもひとつずつ分けていったらシンプルになるので、そのシンプルになること自体が大事だったと思います。

――　オブジェコン自身がどの状態推移というよりは、自分自身の描画物、例えば弾だと、弾が実際どうなるというプログラムで走っている、多分そのプログラム自身がオブジェクトの弾の絵をしているというところから変わるということでしょうか。先程のお話の流れ上、オブジェコンの中で処理を簡潔にするための部分から発生した理由が一番大きいということでしょうか？

黒須　ええ。

8.デバッグ環境とデバッグ手法

―― 少し突っ込んだ話で、実際にバグがオブジェコンで出た場合というのは、どうやってバグを取るのでしょうか？

黒須 これが入った時には HP64000 という CPU 開発系の開発装置がありまして、それ自体のエミュレーターが完全にトレースもできるので、トレースの時にレジスタの値も全部見られますから、それを使ってその部分からトリガーをかけてやれば、その時の CPU の状態が全部見られるので、それを見ながら「ここでおかしくなっているのは、ここが原因だ」みたいな形でできます。

―― では、デバッカー上でかなりできるということですね。実際にそのデバッカー以外では、ソースコードを確認しつつ想定をしてデバッグをしたのでしょうか？

黒須 いえ、HP64000 が入ってからは基本的にプリントアウトするのが終わった時で、その場でフルスクリーン・エディターを持っていましたから、それを見ながら、なおかつここでできるので、現在のデバッグにわりと近い形で、エミュレーター上のみでデバッグができました。アセンブラがベースですが、ソースコードデバッグと同じような感覚で普通にトレースできますし、止めておいてそこからまた実行もできます。

―― ちなみにエミュレーターの動作速度はどうだったのでしょうか？

黒須 それは完璧な CPU エミュレーターなので CPU と同じスピードで動きます。

それを入れる時は少し高くて、これだけで 1 台 300 万ぐらいするのです。それにエミュレーターがまた数 100 万とかハードディスクが何百万とかいうやつで、それまで使っていた開発装置がひとつ 100 万ぐらいしかしなかった時代なので……。

最初に 4 台買っていただいたのですが、それにハードディスク付けてエミュレーター付けてだと、2000～3000 万になってくるので、そこでの利点がどうかというのをちゃんと出してお願いして、確実にバグも減るし開発期間も短くなるという話をして、当初 3～4 台入れたのですが、良かったのでどんどん増えました。

―― ICE（In-circuit emulator）と考えていいのでしょうか。

黒須 そうです。

―― CPU にそのまま挿して PC 上でデジタル情報が…

黒須　その前のやつにもあったのですが、やはりパワーやデバッグ機能が弱いとか色々ありました。

――　そういった環境というのは、ハードウェアとソウトウェアの職種を完全にわかつという流れになったのでしょうか。

黒須　確かにプログラムとハードウェアと完璧に分かれたのは、これを入れた時ぐらいからですね。

――　ハードウェアの人が、ゲームのデバッグに付き合わなくて済むようになったのでしょうか？

黒須　いえ、でも好きな人は多いですから（笑)。やらなくてもいいのにデバッグをしているというのはよくある話で「それ本当にデバッグなのか？」とか「なぜハイスコア出している？」という（笑)。他人のやつもよく遊んでいました。

――　ジョブコンとオブジェコンは基本的にシーケンシャルで動いていたと思うので、設計上でまず何をやるというのが決まっていたので、そこで動的で何か処理の順番を変えるという概念は基本的にはないということですか？

黒須　CHANGE JOB で勝手に変えることは可能です。

――　状態をわざわざ管理する必要はなくて、スタック処理単位がオブジェクトに対して影響していければ大丈夫という…

黒須　そうです。

――　あと、オブジェクトを並べる順番というのは影響しますね。CHANGE JOB の後が全部、ドミノ倒しに変わるのですよね？

黒須　いいえ、オブジェクトの並べる順番が表示に影響はありますが、CHANGE JOB によって、それ以降がドミノ倒しに変化するわけではありません。あくまでも、そのオブジェクトだけが影響を受けます。場合によっては順番を決めて、ここからしか使わないとか、特殊な書きかたをする場合もあります。そのへんは初期設計として 1 回詰めて。
たとえば弾系は、ここから後ろで空いているところを使うみたいな、弾同士は多少上下があってもいいだろうとかはあります。そうじゃないと、見えなくなる可能性があって「何

に当たったの？」と言われてしまうので、シューティングの場合は最後に弾を描画しないといけないですね。

―― ジョブコン上でどのオブジェクトから回すか、みたいな定義だけはきっちりして、オブジェコンではその自分より後に対しては影響力を持てる……。

黒須 いえ、別に手前のものでも大丈夫です。次のフレームが、大抵、敵とか自分とかが前の方にいたとすると、後ろの弾が前のやつに当たるので、後のほうにいて、前のやつで弾が当たっていたりしたら、ジョブを変えておくので、次のフレームから爆発シーンに変わっていきます。

―― もう現場では使われていると思うのですが、黒須さんとしては今タスクシステムが浸透していることに関して何か思うところやご意見はありますか？

黒須 普通にあるものなので、それは別に否定もしないですね。

―― 逆に良かったというのは、正しく設計ができる手法のひとつとして存在していることは良いことではないかと思うのですが。

黒須 実際にタスクを分けることで分業とかもしやすくなっているはずなので、そういう意味では実際に必要で、この時になかったとしても後に出てきただろうと想像できます。

―― それはプログラマー同士の分業ですか？

黒須 そうですね。プログラマー1 人で片付けられないものになってきた場合にできるはずなので。ジョブコンやオブジェコンがなくても、いつかは出てきたのではないかと思います。ただ、これが出てきたことによって、開発システムもそうですが、分業もプログラムの効率もデバッグも、連鎖的に上手く物事が回るシステムになりました。

(c) 1980 年から現代までゲーム開発技術の推移

1. 80 年代の開発環境の形成のされ方

―― ひとつのゲームに対するプログラマーの人数というのは増えているのでしょうか？

黒須 規模によります。1 人でやるものから十数人でやるものまで、規模によって全然違います。

―― 1980 年代のナムコでも十数人でというのはありましたか？

黒須　ないですね。その頃は大抵 1 人でやっていました。ゼビウスなどは深谷さんがまずある程度アップして、それぞれの部分を遠藤が作り上げていくという形だったので、ただ同時にやっていたわけではなく、交代したみたいな感じです。

―― そこで例えば、シニアプログラマーとその下のジュニアプログラマーの分業みたいな形になっていると思うのですが、それと同じような形でゼビウス以外のタイトルでもそういう、入ってきた人に続きをやってもらうみたいなことはありましたか？

黒須　ないです。当時はなかったです。

―― では、どうやって育てたのですか？

黒須　いや、ちゃんと育てていない可能性があるので、ひどいとは思いますが…職人に近い状態で見て盗めという感じだったかもしれません。
　システムは HP64000 が入ったことによって、それ自体が今で言う LAN みたいなものでつながっていたので、ハードディスクは誰でも見られるので、皆ログインして始めて作ってログアウトしていく状態でした。

―― そのスペックは当時の一般の人が持っているパソコンと比べ物にならないですよね？

黒須　そうですね。スピードはそれほど速いわけではなくて、全体につながっていることが大きな違いです。

―― ナムコの開発体制というのに、アメリカとかの情報というのは入っていたのですか？

黒須　もちろんアタリジャパンはナムコが持っていましたから、アタリとの関連性はかなり強いです。ただ、ソフトに関してはそれほどでもなくて、ハードウェアに対しての影響を結構強く受けていました。

―― では、アタリが開発システムで HP64000 を使っていたのでしょうか？

黒須　いや、使っていないと思います。アタリは当時 VAX などを使っていたのではないでしょうか。

当時の感じだと DEC（VAX）を使っているところもありましたし、ナムコ時代もテスト的に入れたのですが、HP64000 のほうが使いやすかったので、そっちを増やしました。

2.　80 年代の技術の情報環境と開発環境の関係について

――　そういった、海外のコンピューター動向とかを専任的に収集される方とかはいらっしゃいましたか？皆さんが各々勝手に集めていた感じでしょうか？

黒須　入ったばかりの頃にはそれほど本がなかったですが、「インターフェース」とかは読んでいました。それ以外ですとネットがないので図書館とかで調べるしかありませんでした。大学の図書館とかに行けば、色々な資料がありますが、でもそれを探すための取っ掛かりがなかなかありませんでした。

――　ナムコがどういう形で、当時、技術情報を入手して開発体制を整えていったのでしょうか？　そうした部門は深谷さんが見ていらしたのですか？　そういう管理システム全体をどうマネージメントするとか、設計するというのはどこの部署だったのですか？

黒須　それは、色々なところが営業に来たのを皆で見に行って、良いものがあったら入れてみるといった形でやっていました。

――　例えばシニアプログラマーと実際の製品開発のために働くプログラマーというのが、きれいに分業されている状況だったらまた違ったと思うのですが、実際に自分でソースを読んで、最初はタスクの部分を書いているだけでも、実際には全体のソースをちゃんと読んで理解するみたいな形でプログラマーは育っていったと思います。

それで、ナムコにずっとおられる方もいますが、辞めて出て行く人もおられます。その中で他社に行く方も多いと思うのですが、行った先で恐らく自分用に考えていたタスクシステムを再実装するといったことが、かなり行われていたと思うのですが、そういうので当時のナムコから例えば人が出て行くとか他社に転職するというのはわりとありましたか？

黒須　あまりいなかったのですが、私がなぜ辞めたのかも今は記憶にないぐらいに最初に辞めたほうなので……。

――　例えば黒須さんがお辞めになった後の話だと、恐らく技術として外に出たとしても、それが本格的になるのは 1980 年代後半以降でしょうか？

黒須　多分そうですね。ただ、3D を始めたのが 1980 年代ですから。ナムコは色々とやっていましたからね。仮想 3D からはじまって…

―― 産業として見た時に、アメリカから最初に渡ってくる形でゲーム産業が発展して……ナムコはアタリの影響が大きいですよね。

黒須　かなり大きいです。

――　そういった中で、どのように開発がスタートしたのか関心があるのですが、入社した時点では、すでにもうナムコはゲームに参入しています。ギャラクシアンから……ジービーが出た頃ですね。黒須さんが入社された頃には……

黒須　パックマンを作っていました。もう開発組織としては大体整っていた時期ですね。入った時点では。中村製作所からナムコになって 2 年目ぐらいだと思います。何年か前のレポート用紙に中村製作所と書いてあるのがまだ残っている時代でした。
当時は新しい CPU が出る度に持ってくるので、向こうですとインテルとモトローラーが作っているのですが、こちらだと、それのセカンドソースの NEC や日立がありましたから、「新しいのが出たので是非」と、エンジニアサンプルとか書いてある CPU とかを持って来られて「面白いね」と遊びながら……

――　CPU とかが変わると、どんどん開発環境自体も進化したのですか？

黒須　それが良かった点が、先言った HP64000 自体が色々な CPU に対応できるものだったので、MOS 6502 は、もうあったのですが、ファミコン時代のエミュレーターはだいぶ後にならないと出ず、HP64000 でファミコンを作っていました。アセンブラ自体は存在していたので。それ自体は新しい CPU が出るとアセンブラもすぐ出るシステムでした。

――　それはどこが提供していたのですか？

黒須　ヒューレット・パッカードです。あと、そこ自体はアセンブラを作るためのものも存在していたので、今で言うコンパイラ-コンパイルみたいな形で、特殊な 4 ビット CPU とかは当然ないので、それ用のアセンブラを作るためのものがあるので、それを使って 4 ビット CPU の開発も行ったりしていました。まあ、売れてくると後から出てきますが、富士通が作るような 4 ビットとかだとなかなか……。
当時は横川・ヒューレット・パッカードでしたが、やはりヒューレット・パッカード

はアメリカの会社なので、向こうの作った CPU はわりと早く来るのですが、日本で作られた 4 ビットとかは知らんぷりされているみたいな感じでした。

3. コピー問題と技術背景

―― 当時だと、デッドコピーとかの問題があったと思うのですが、それの対策でどんどん何か…

黒須 最初は結局 Z80 とかだけでやっていたのが、そこで 4 ビットのやつを使ったりしていました。

―― それはコピー対策で？

黒須 そうですね。でも、それも結局コピーされるので、そうするとまた別の機能を付けたりして、最初は「CPU のプログラムは読めない」と言っていたのですが、結局読まれるようになってしまいました。

―― 当時ナムコは CPU や IC の設計みたいなことをやっている部署があったのですか？

黒須 CPU は最終的には作ろうとしていましたが、確か作ってはいないです。実現していないかもしれません。要は既存の CPU よりも速いやつを作りたいということがあって、それで色々やっていた時もありました。

―― 時代背景が全然わからず短絡的な質問で申し訳ないのですが、やはり当時で一番 CPU を酷使するようなアプリケーションを動かしていたのは、グラフィックも込みだとゲームだったのでしょうか？

黒須 どうですかね…

―― 今は完全にゲームがグラフィック・ボードを引っ張るという感じがします、当時のハードウェアも、アーケードはどんどん性能が高いものを入れて速かったという印象はあるのですが……

黒須 まあ、すぐに持って来てくれたのと、何だかんだと他よりも特殊なものを大量に使うので、要は半導体メーカー自体が一番売りやすかったのかもしれません。

―― 上得意ですよね。

黒須 結局、他だと医療用は別として、家庭用のコンピューターを作っているところとかだったら、なかなか買わないじゃないですか、結局 OS ありきなので、OS ベースの CPU しか乗っかってこないみたいな……それに対してゲームメーカーなら OS はなくてもいい状態でやっていましたから、それが値段と機能が良ければ採用するというのがあるので。RISC CPU ができて、アセンブラだとやっていけなくなって……。

―― 転換期ですね。

黒須 そうですね。そういう意味で言うと。RISC CPU の場合には最高のスピードを出すためには命令の順番を変えないといけないじゃないですか、それは人の手では無理なので、それでコンパイラが必要になってきたのかなという気もします。

―― アセンブラで書けないわけではない？

黒須 書けますけど、書いたときに、この命令の後はこの命令しかできないという決まりがあって、それをやらないために NOP（no operation ）を入れると、せっかくの CPU を遅くしてしまうだけなので、そのへんはコンパイラがスケジューリングしてくれたほうが速いです。それで、これは無理という話になってくるので……。

ARM（Acorn RISC Machine）とかは安い値段で出てきて、何で 3DO があんなに高かったのだろうかっていう（笑）。コストを考えたら「この値段で売れるのに」と思っていました。多分 3DO だと中の部品代を合わせても 1 万円しないです。CPU が何百円で入ってくるので。ARM は安かったのです。

インタビュアー　三宅　陽一郎（株式会社フロム・ソフトウェア）
インタビュー　2009 年 3 月
インタビュー中の図は、黒須氏の解説から三宅が作成し、黒須氏にチェックして頂いた。

第 5 章　GDC における海外のゲーム関連技術についての調査

三宅　陽一郎
株式会社フロム・ソフトウェア　技術部
（GDC の発表資料は次のＵＲＬからダウンロードできます。）
http://mygdc.gdconf.com/vault/1337

5.1　はじめに

本章は、2009 年 3 月 23 日から 27 日まで、米のサンフランシスコで開催された、ゲーム開発者のためのカンファレンス、GDC2009 (Game Developers Conference 2009)の技術レポートである。2009 年という年は、ゲーム産業にとって節目となる年である。次世代機（Xbox360、PlayStation 3, Wii）の開発期を 3 つの時期に分けるとすれば、2009 年は第一期と第二期の幕間に当たる。

（Ⅰ） 2004 年～2008 年（次世代適応期）
次世代機の開発の基礎を築く。次世代機のファーストタイトルをリリース。

（Ⅱ） 2009 年～2011 年（発展期）
基礎をブラッシュアップ、クオリティーアップしながら、
2～3 個目のタイトルをリリースしながら量産体制へ移行する。

（Ⅲ） 2012 年～2014 年（完成期）
技術は完成し、その次の世代の開発が入って来る。
次世代機の最終的な完成タイトルがリリースされる。

GDC2009 が行われる 2009 年 3 月という時期は、次世代機向け開発の最初の時期が終わり、第一期からステップアップを見越して次の開発へ向けて準備、進行している時期である。まず、GDC の概要をここで説明する。GDC は開発者から見た場合 3 つの機能を持っている。

① 情報を収集する場所。
② 開発の潮流を構成し確認する場所。
③ 開発者同士が交流する場所。

① GDC（Game Developer's Conference）は、技術分野のみならず、ゲームデザイン、CG，サウンド、ビジネス、ミドルウェアなど、全分野の世界中の開発情報が集まる場所である。5 日間に渡り、講演、パネルディスカッション、ラウンドテーブル、キャリア・パビリオン、エキシビションから構成される。

図 5.1-01　GDC の講演風景

（最も大きな講演の場合（ピーター・モリニューの講演）。欧米の方の講演はいつ聞いても自信満々である）

② GDC は、ゲーム開発の情報を各企業、各開発者が持ち寄ることでお互いに確認し、カンファレンスを通じて自分たちでゲーム開発の潮流を作って行こうという意識が高い。そういった意味で学会などの意識に近く、自分たちでゲーム開発の歴史を作って行こういう意識を共有している。そういった意識が、他のゲーム・カンファレンスと GDC の質を分かつ大きな要因となっている。また、これは、そもそも GDC が問題意識の高い開発者同士がお互いを刺激して開発の気運を盛り上げて行くことを目的として始まったイベントであることが影響しているのかもしれない。

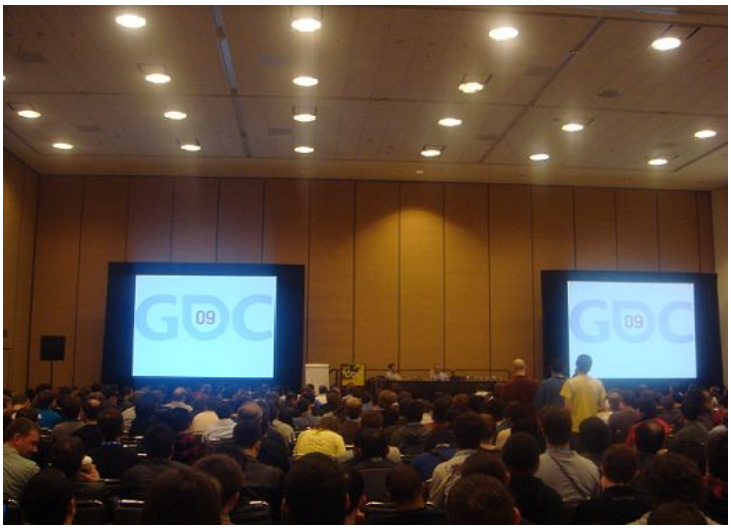


図 5.1-02　GDC の講演後の質問風景

（講演後には決まって活発な議論が行われることが多い）

③ 18 時でカンファレンスが終わった後は、職種（AI など）や、地位（エグゼクティブなど）や、団体（IGDA など）ごとに、パーティが開催されている。そこで、ビジネス・コネクションのみならず、各技術分野のコミュニティの形成や、技術のディスカッションが行われる。これは、GDC のもう一つの重要な機能、コミュニティ形成の役割を担っている。後述するが、ゲーム AI コミュニティや、AI Programmers Guild も、そういったコミュニティ形成の中から出て来た成果の一つである。

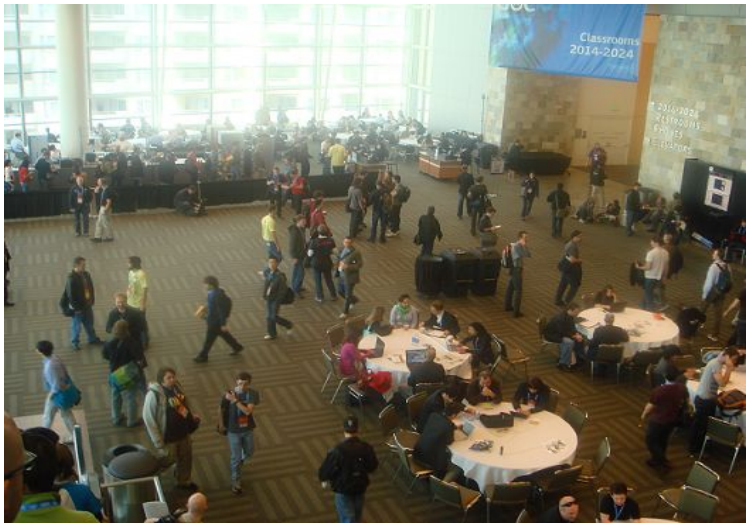


図 5.1-03　会場の風景

（至るところでいろいろな出会いがあり開発者同志の交流がある）

昨年、2008 年の GDC は、第一期の次世代機の最初のタイトルの開発をしながら培った基礎技術やノウハウの集大成のようなカンファレンスであった。2009 年の GDC は、遅れて来た大作、「KILLZONE 2」(Guerilla)を別にすれば、これからの第二期へ向けて、第一期で形成した土壌を、どうブラッシュアップするか、レベルアップするか、という課題が主なテーマとなっていたのである。

具体的な技術的な特徴としては、以下の 5 つのポイントである。

(1) PlayStation3 における CELL プロセッサの使用法（プログラミング）
(2) レンダリングの新しいトレンド（deferred Lighting と Light Pre-pass Renderer）とシェーディングの新しいトレンド（Global Illumination）
(3) キャラクターアニメーションの精緻化とメタ AI
(4) プロシージャル手法（地形生成、セミ・プロシージャル手法)
(5) インディーズにおける技術とゲームデザインの融合

本報告書では、以上のような今年の GDC2009 に見られる技術方向の姿を浮き彫りにしながら、これまでの技術の潮流を分析し、これからの開発の方向を予測する。

5.2 PlayStation 3 における SPU の使用法

PlayStation 3 に搭載された SPU（SPE 内）は、並列に動作させることが出来る。グリッド構想[1,2]など、本来は、ネットワーク上にある CELL プロセッサを使うことも構想の内にあるが、現在のところ、各コンソール単体に搭載された SPU を如何に使いこなすか、が課題となっており、各企業が研究して来た。日本の CEDEC でも、こういった SPU の使用法についての講演が行なわれて来た。『メタルギア・ソリッド 4』（株式会社コナミデジタルエンタテインメント）のグラフィックにおける SPU の使用法は現時点の最高峰の一つと言われている[3,4]。また、2007 年に CESA が開催した「第 2 回ゲーム産業開発者のためのスキルアップ講座」[5]における、実際の CELL プロセッサの開発者による講演も行われている[6]。基本事項は SCEI や IBM のサイトで文書で公開されている[7]。さらに、「PlayStation Edge」の公開の効果も大きい[8]。ここでは、GDC2009 で特に SPU に特化した講演について幾つかピックアップして解説を行う。

5.2.1 Insomniac の PlayStation3 Programming 講座

GDC は、前 2 日間がチュートリアルとして、各技術に関するまとめセッションが行われるが、このチュートリアルにおいて、開発会社 Insomniac Games [9]による、SPU の使用法について一日セッション「Insomniac Games's Secrets of Console and Playstation 3 Programming」が行われた[10]。Insomniac は、R&D の公開に力を入れており、その資料は今回の講演資料を含めて Insomniac の R&D のサイトから全てダウンロードできる[11]。また、Nocturnal と呼ばれる技術 Wiki も公開されている[12]。

講演は、

(1) SPU gameplay （Joe Valenzuela）
(2) Insomniac Physics (Eric Christensen)
(3) Pre-lighting in Resistance 2 (Mark Lee)

からなり、ゲームエンジン、物理、レンダリングに関して詳細な解説が為された。

(1) SPU gameplay

「SPU gameplay」では、SPU のためのコーディング技術についての基本事項が解説された。SPU のコーディングの特徴として「マルチプロセッサ」「NUMA」（Non-Uniform Memory Access)、コードの試行錯誤、コーディングデザインの知識不足と難しさを挙げた。さらに、本講演では、魚の群れと巨大魚を SPU と PPU を使い分けながら制御する例を挙げて、実際のコーディングに踏み入りながら説明した[13]。解説のためにゲームフレームは単純化され、「如何にキャラクターの状態をアップデートするか」をテーマとした。アップデートとはこの場合、シェーダーや物理シミュレーションを意味する。PPU 側のメインメモリと SPU は DMA を通してデータを転送し、SPU は小さなメモリ上で SIMD 演算によってローカルストレージに置いたデータによって高速な計算を行うため、SPU と PPU の間でデータ構造とコード構造の柔軟なコンパクト化が PlayStation3 のプログラミングには必要とされる。

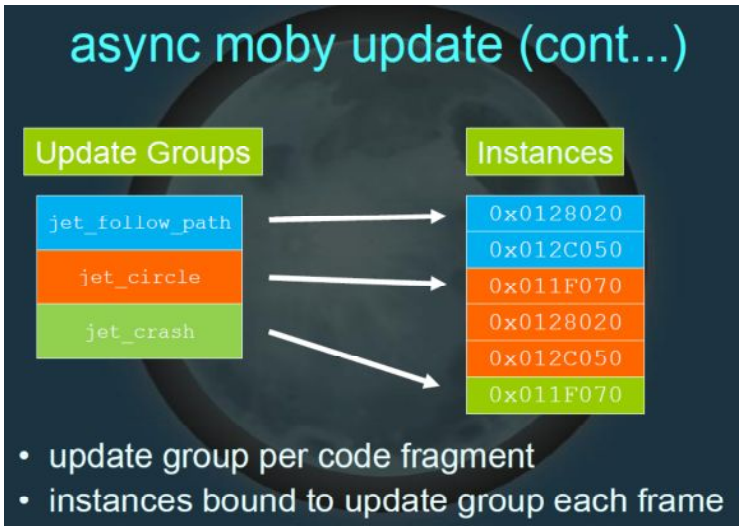


図 5.2-01　SPU のためのアップデートの原理

（グループに対して、アップデートに属する関数が対象とする変数をインスタンス化して SPU で処理を行う[13]）

例えばたくさんのグッピーの状態を一フレーム内で更新する場合、PPU であればオブジェクト単位でアップデートクラスによってアップデートを行う手法が考えられるが、SPU で行う場合は、必要なデータと処理関数部分(code fragment)を切り出して更新する必要がある。つまり、アップデートの処理を行う関数群に対して、それが対象とする変数群を集めてインスタンス化して SPU で処理を行う。

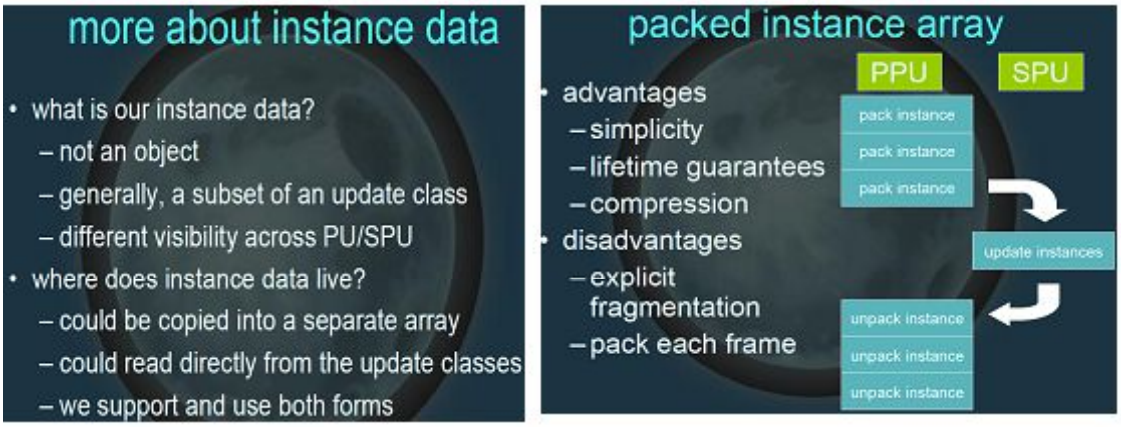


図 5.2-02　シェーダーにおけるインスタンス・パック化と SPU の処理[13]

毎フレームパック化さえたインスタンスが SPU で処理され、再び PPU 側に戻される。

また、コーディングレベルにおいて最初からデータ構造体を分けておくという Tips も紹介された。

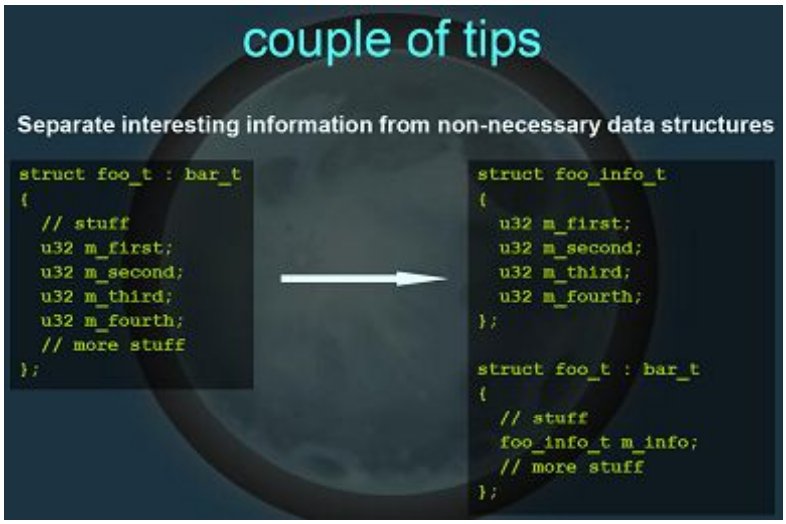


図 5.2-03　メモリ構造体の工夫[13]

（通常（PPU で使う）のデータ構造から SPU 用にデータを小さくアレンジしておく）

図 5.2-04　魚の群れと鯨のシミュレーションの例

（グッピーは完全に SPU 制御を行い、鯨の方は非同期処理を SPU で行う[13]）

魚の群れと鯨のシミュレーションの設計においても、多数のグッピーは小さなデータで完全に SPU で処理するが、鯨は非同期的な処理を SPU に振り分けて処理させている。このように、データ構造、プログラムにおける PPU と SPU の使い分けのシステム、そして SPU を使って一フレームに限らず非同期的に数フレーム内に、処理を最大化するプログラムの設計が解説された。

(2) Insomniac Physics

次に、「Insomniac Physics」では、物理シミュレーションについて、

①『Resistance: Fall of Man』 の PC から PS3 への移植
②『Ratchet & Clank Future』
③『Resistance 2』
④ 現在

へ至る SPU を用いた物理エンジンの変遷について解説された[14]。衝突判定と物理シミュレーションについて

① SPU と PPU が並列で処理。
② SPU で完全に処理。物理インターセクション・シェーダー、物理ソルバー・シェーダーの実装など、シェーダーと物理の共通処理を利用した計算を SPU 処理。
④ フレーム内で Immediate(即時処理、IK やラグドール)と deferred (遅延処理)を明確に分離。この分離は一フレーム内の処理を高速化する。
⑤ 処理オブジェクトのリスト化、キャッシュも SPU で処理。物理コリジョン・シェーダーの実装。

のような遷移を辿って来た。全体としては、本来の物理処理をシェーダーとして SPU に実装し溶け込ませて行くことで、とうとう殆どの処理を SPU で実現することに成功した。

Insomniac はこの他にも SPU に関るテクニックを自社のサイトの R&D で公開しており、多くを学ぶことが出来る。また毎週、社内で技術発表会を行なっていると記載されている。

3 番目の「Pre-lighting in Resistance 2」は、SPU というより、主旨はレンダリングの手法の解説であるので、後述のレンダリングの項で解説を行う。

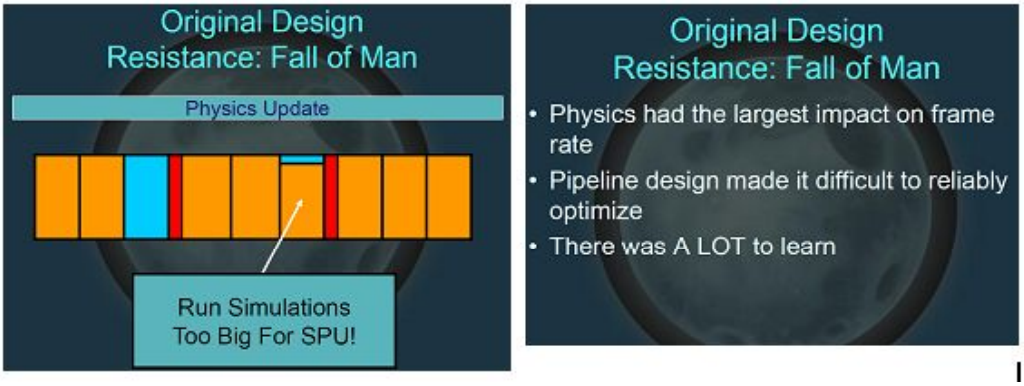


図 5.2-05 SPU と PPU が並列で処理[14]

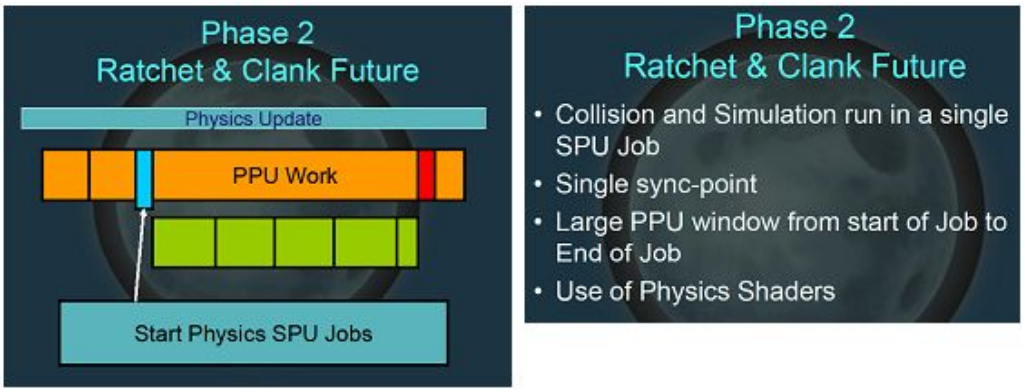


図 5.2-06　SPU で完全に処理[14]

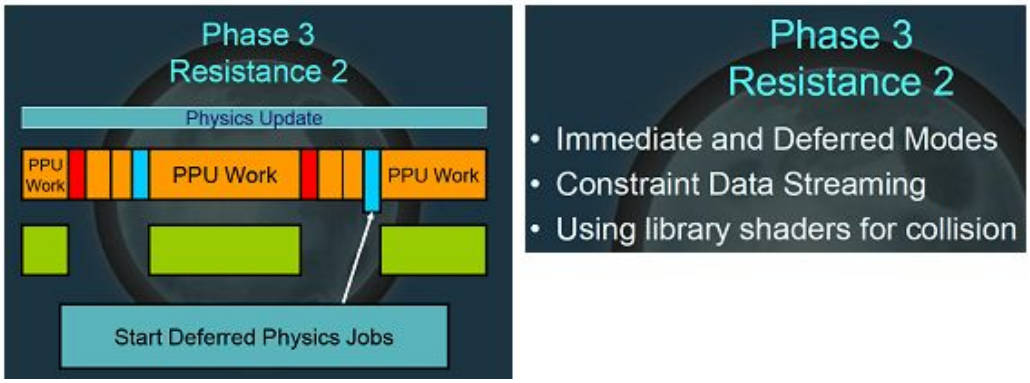


図 5.2-07　Immediate(即時処理、IK やラグドール)と、deferred (遅延処理)を分離処理[14]

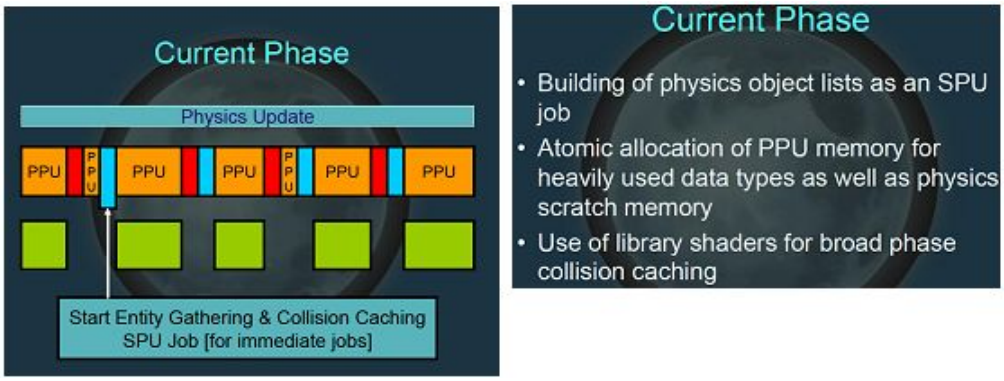


図 5.2-08　処理オブジェクトのリスト化、キャッシュも SPU で処理[14]

5.2.2　GOD OF WAR における SPU の利用

「Practical SPU Programming in God of War III」と題して、Jim Tilander, Vassily Filippov(Sony Santa Monica)　氏による講演が行われた[15]。彼らの方針は「SPU は

co-processor（補助プロセッサ）でない」これは当たり前のことであるが、SPU と PPU をなるべく平等に補完的に使おうという方向であった。

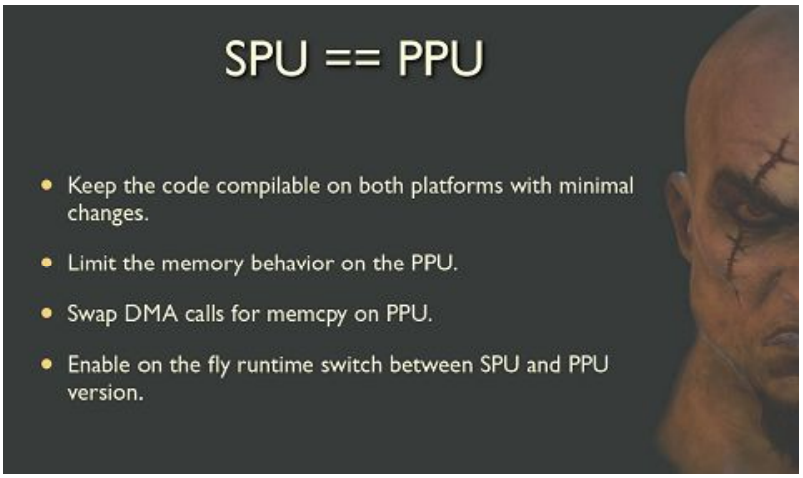


図 5.2-09　SPU と PPU のコードの互換性を維持する[16]

つまり、基本的に PPU で処理し切れない処理は SPU 側に任せることで、一フレーム内の処理を分散させようとする方針である。そのための PPU のコードは、少しの変更で SPU のコードとして使用できるようにされている。また、ラインタイムにおいて同じ処理を SPU と PPU へ動的に切り替えられるようになっている。さらに、PPU，SPU，RSX（Nvidia）3 者の協調が強調されていた点であった。また、プロファイラによって、どういった処理が並列化されているかを常にモニター出来るようにデバッグに適した環境を用意している。

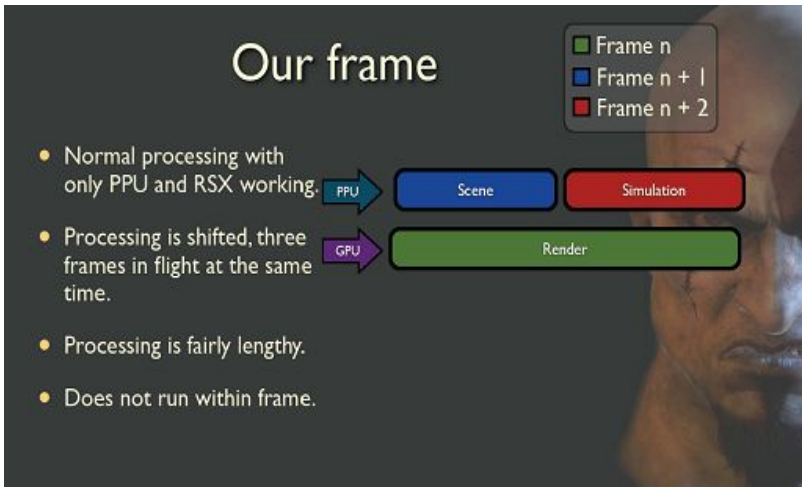


図 5.2-10　SPU なしの処理。一フレームで処理し切れない[16]。次の図へ。

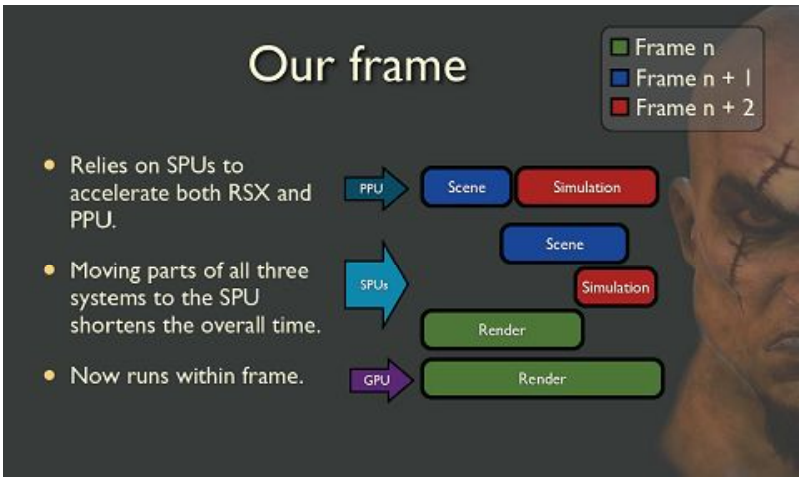


図 5.2-11　SPU を PPU に相補的に使用することで、複数のバッファを使って処理をし切れるようになる[16]

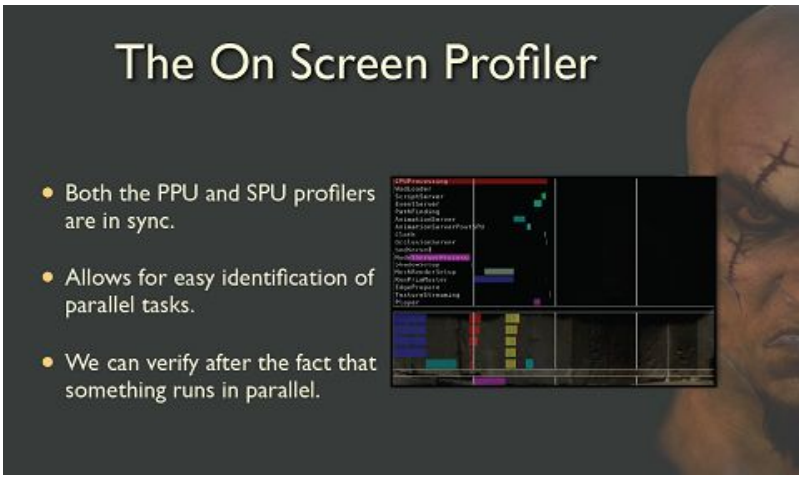


図 5.2-12　PPU と SPU のプロファイラ[16]

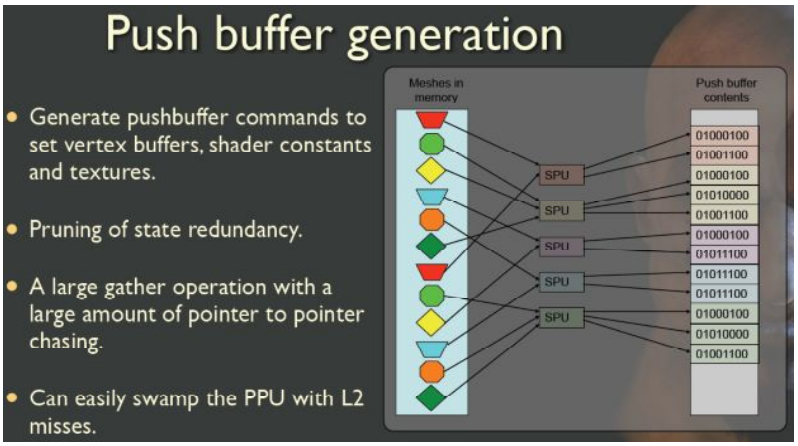


図 5.2-13　Push buffer generation[16]

Push buffer generation とは、計算に扱う頂点バッファ、シェーダーの定数、テクスチャの実データをポインタを解消してフェッチすること

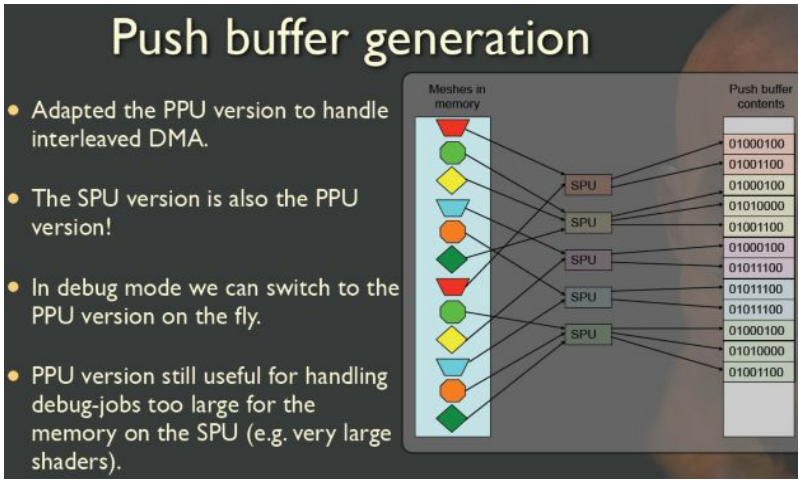


図 5.2-14　SPU と PPU のコードは同じものになる[16]

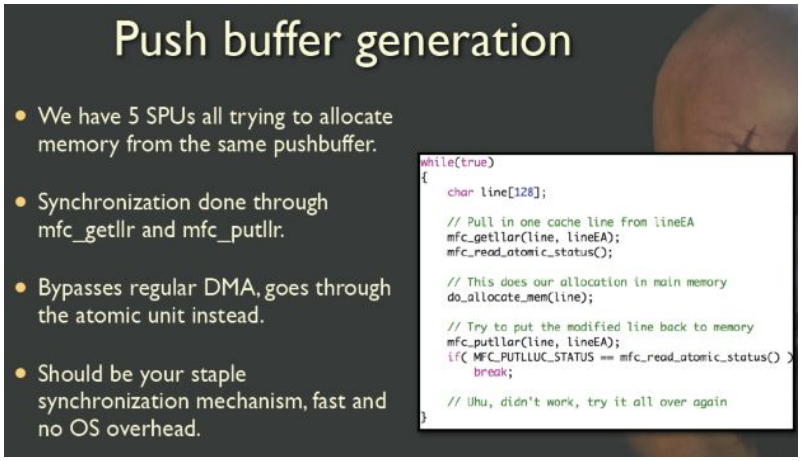


図 5.2-15　Push buffer generation によって SPU をメモリ・アロケートする[16]

またシーン描画については「Push buffer generation」というテクニックが解説されていた。これは、処理のはじめにポインタを解消して実データをバッファリングする手法であり、ポインタに依存しない処理に還元することで、SPU と PPU のコードをより近いものにする技術である。実際、バッファリングした後は、SPU と PPU は処理的に近いコードになる。

5.2.3　KILLZONE 2 における SPU の使用法

『KILLZONE 2』(Guerilla Games)は、新規タイトルにも関らず PlayStaton2 でミリオンセラーを果たした z『KILLZONE』の続編である。Guerilla Games は現在は PlayStation 3 に特化した開発を行っており、SPU の使用方法の研究においては、非常に深く多岐に渡る使用法を実践している。パーフレームの処理から、より汎用的な処理まで、AI からグラフフィック、パーティクル・シミュレーション、データ解凍、レンダリング、衝突判定など、非常に多岐に渡って SPU を使用している。また、強力なプロファイラに

よっては並列処理の負荷をモニターできる仕組みになっている。

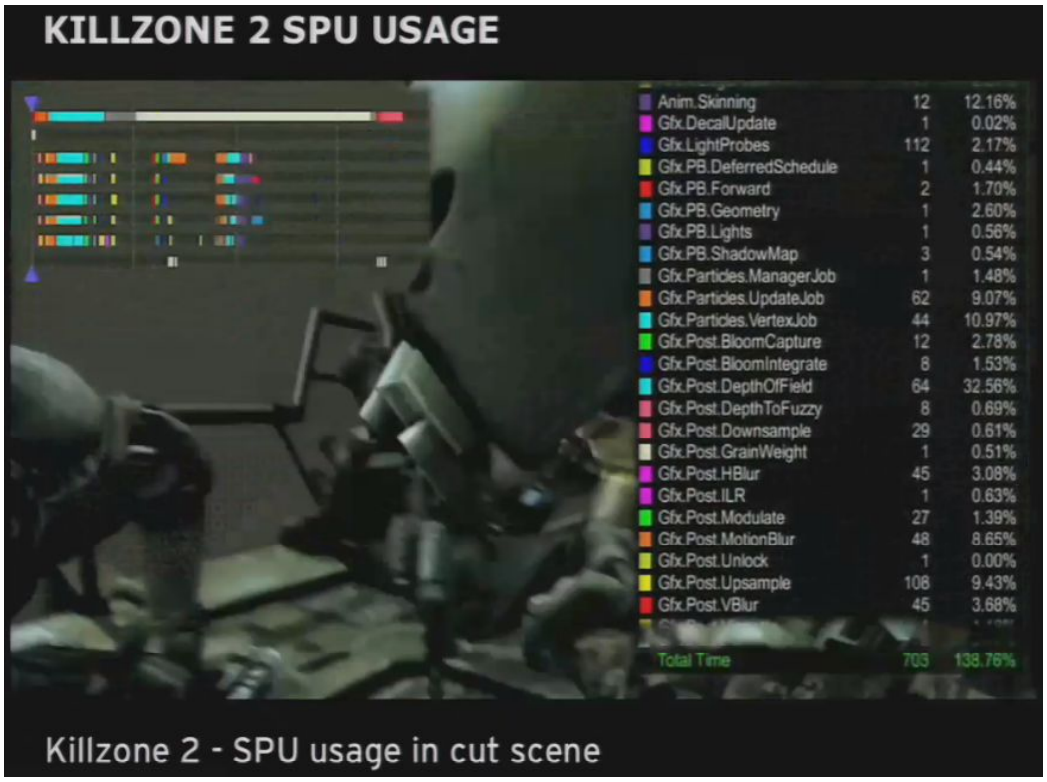


図 5.2-16　「KILLZONE2」における SPU プロファイラ[17]

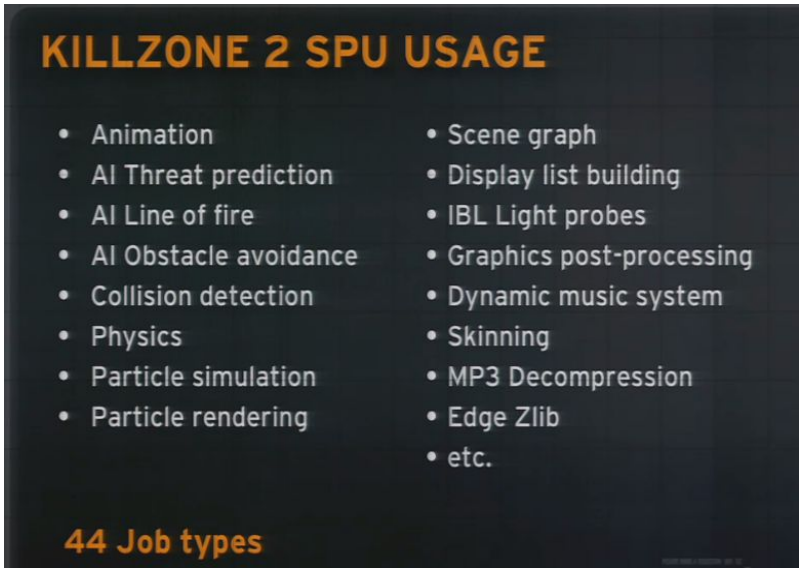


図 5.2-17　「KILLZONE 2」における 44 の SPU 使用リスト[17]

① パーティクル・シミュレーション

最初は PPU で処理していたが、最終的に SPU で殆どの処理を行う。
速度的には 20 倍の速さになった。

② イメージ・ポストプロセッシング

「モーションブラー」「深度マップ」「ブルーム」は SPU で処理を行う。
SPU5 つを働かせることで高速化・高品質化を実現した。

③ アニメーション

エッジアニメーションが非常に高速。SPU の複数並列化によって 12%
に計算時間を高速化。PPU に比べても 5 倍の速度を実現した。

④ AI

ウェイポイントベースの AI で、小さな反復計算が多く、SPU に適している。
特に、射線計算を SPU で行い、プレイヤーの位置を予測して行動する[18]。

⑤ 総合スケジューリング

SPU はジョブベースの構成で複数のジョブマネージャーを統一的な
ジョブマネージャーが総括する。

「KILLZONE 2」は印象として、真正面から SPU を汎用プロセッサとして用いているように見える。ファーストパーティとしての役割でもあり、SPU の正統な使用法を、PlayStation 3 のタイトルを開発する他の全ての企業にアピールしているように思える。実際、各トピックは PPU で処理した場合と、SPU で処理した場合の比較が為されており、SPU を使えばフレーム内の処理や、フレームを超えた処理を軽く行えること一つ一つケース実証している。

『KILLZONE2』は AI にも非常に力を入れているタイトルである。『KILLZONE』『KILLZONE 2』の AI の特徴は、位置を基本にした思考に特化した AI という点である。

『KILLZONE 2』の AI は、プレイヤーを攻撃する最適な位置を取り、プレイヤーが見えなくなるとプレイヤーの出現位置を予測して移動する。以下に、原理を説明して行く。

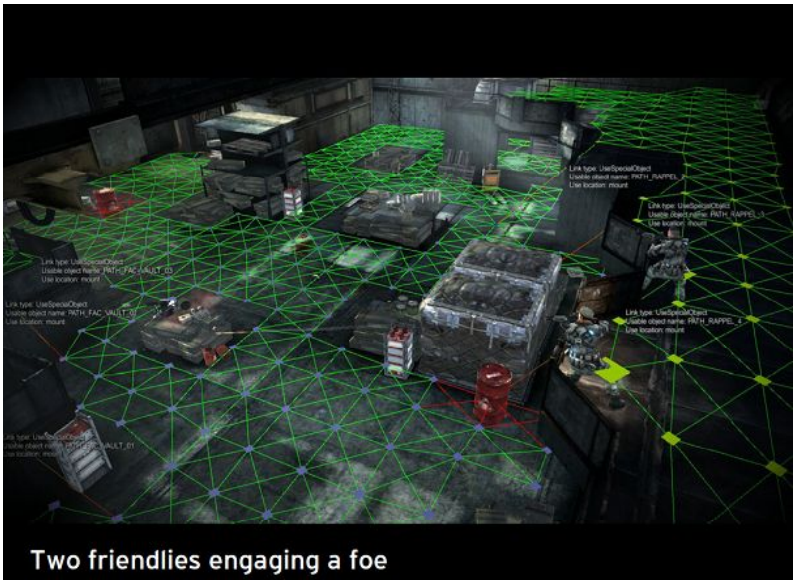


図 5.2-18　『KILLZONE 2』におけるウェイポイントの様子[17,18]

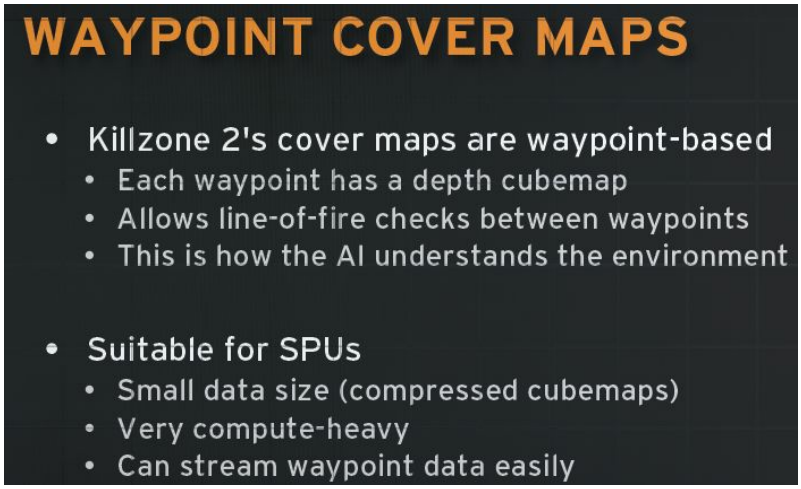


図 5.2-19　KILLZONE 2 のウェイポイント基本スペック[17]

『KILLZONE 2』のパスデータは、ウェイポイントデータである。つまりポイントであり、それが相互にリンクされたデータからなる。基本的に、四角形をたすきにかけたような構造となっている。通常、パス検索と射線判定、視線判定は全く別の機能であるが、KILLZONE シリーズでは、この機能もウェイポイントデータと統合されている。これを見るために、前作の『KILLZONE』のシステムを振り返ってみる。

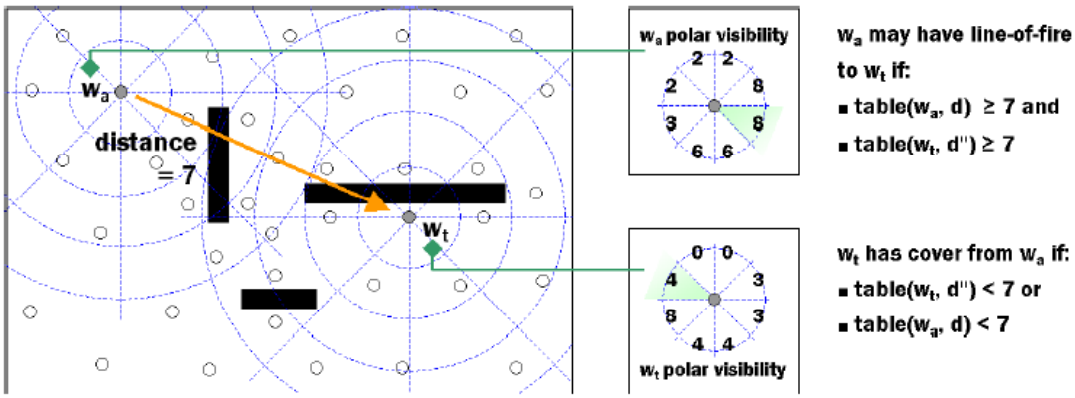


図 5.2-20　『Killzone』 の LOS 事前計算の結果をウエイポイントに埋め込んでおく方法[9]

8 方向に対して LOS の最大値を事前に計算しておく。その情報を用いて、簡単な計算で、「射線計算」「隠れる判定」を行う。8 方向だからアバウトというわけではない。キャラクターは常に移動しているので、逆にこれぐらいの粒度で十分であるという考えである。精度を上げれば（例えば 64 方向）では、それだけ正確になるが、精度に依存した AI は、微細な誤差や敵の急激な運動に弱くなることもある。なので、少々敵が動こうが、誤差があろうが、AI として健全に動くシステムを作る、これを「ロバストな AI」という

『KILLZONE』ではまず、各ウェイポイントから、8 方向に対してどれだけ視線が通るか、という長さを事前にテーブル化して記憶しておく。そして、ゲーム中である点 A からある点 B まで視線が通るかどうかを、A が持つ（だいたい）B 方向への視線距離が実際のユークリッド距離 L より長いか、それと、B から見た（だいたい）A 方向の視線距離 L より長いかを判定する。この二つの条件が満たされれば、A から B までは見えている、と判定するのである。つまり、簡単なテーブル参照で、A から B までの視線チェックが出来てしまうわけである。

『KILLZONE2』では、この 2 次元 8 方向から、3 次元立方体 6 面の各面に対して 12x12（或いは 16x16）のテクセルが張られている視線テーブルへと劇的な変化を遂げている。

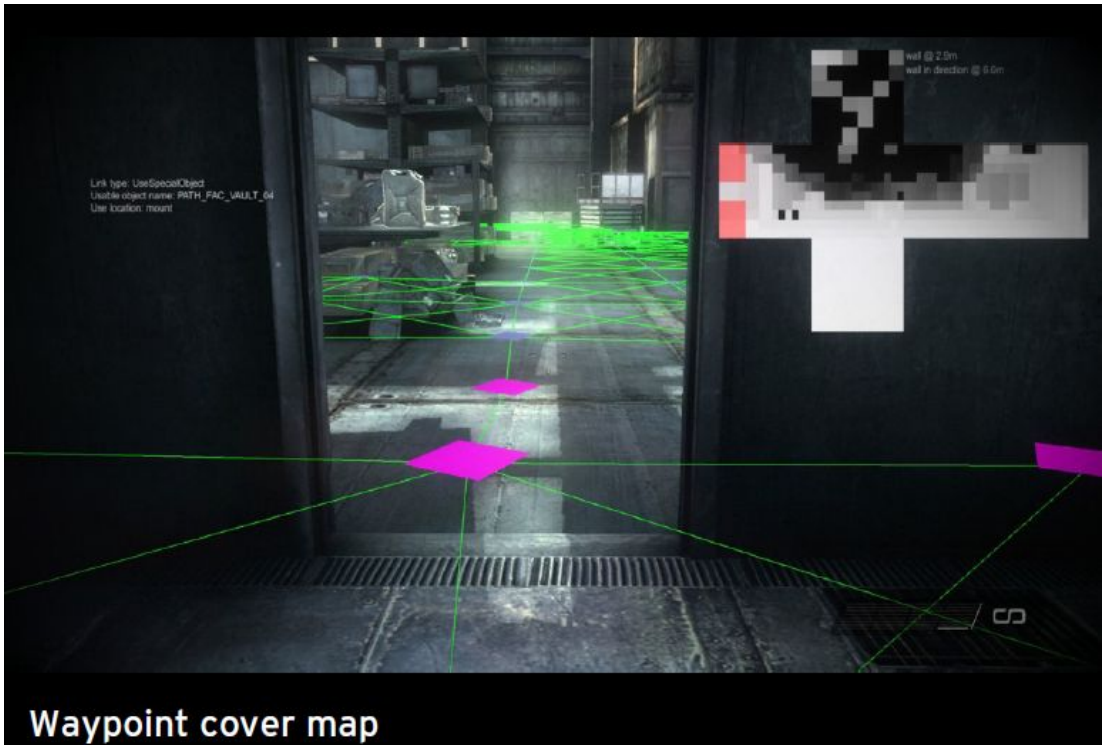


図 5.2-21　『KILLZONE 2』におけるウェイポイントとデプス・キューブマップ[17,18]

（各ウエイポイントで事前計算されているデプス・キューブマップを展開して表示したもの（図右上））

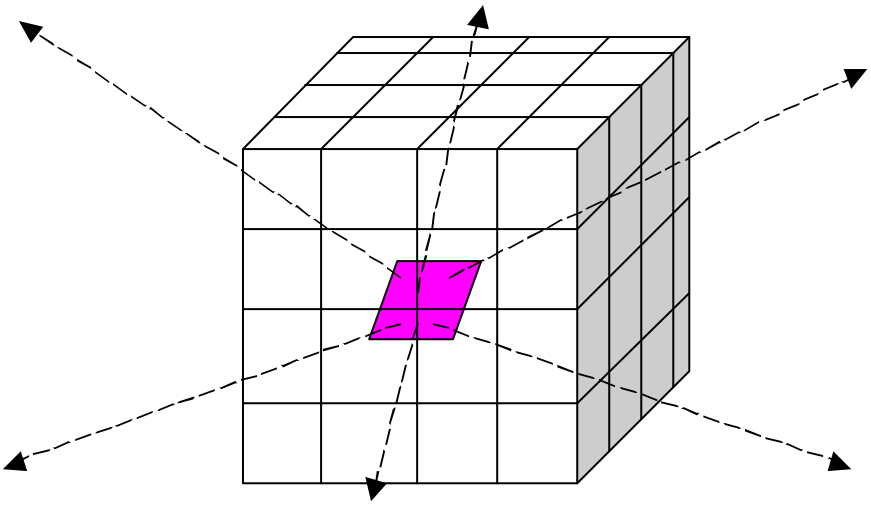


図 5.2-22　『KILLZONE 2』における深度キューブマップのイメージ[17]

（中心のウェイポイントから 6 方向に（ここでは解説のために）4x4 テクセル方向の深度データを事前に計算してテーブルとして各点に持たせておく。これを見やすいように展開したものが、一つ前の図の右上にある展開図）

この深度キューブを利用して高速で視線判定と射線判定をウェイポイント間で行うのが、『KILLZONE 2』のシステムである。さらに、他の AI の思考もウェイポイントと高速な射線判定の上に構築されている。例えば、敵の移動予測である。それは、次のステップで行なわれる。

① 一度、プレイヤーを見たら、その時刻と位置を記憶する。

② プレイヤーを見失ったら、一番最近見た位置を中心に、目撃時間からの経過時間と共に、プレイヤーがいるであろう予測領域を広げて行く。

③ 移動予測領域に含まれるウェイポイントのうち、AI たちが目撃してプレイヤーがいないことを確認すると、その点を予測領域ウェイポイントリストから削除して行く。

④ もし、ある領域を特定できたら、そこへ AI が赴くか、近くに向けて威嚇射撃を行う。逆に、予測領域ウェイポイントリストが長くなり過ぎた時点で、予測を止める。

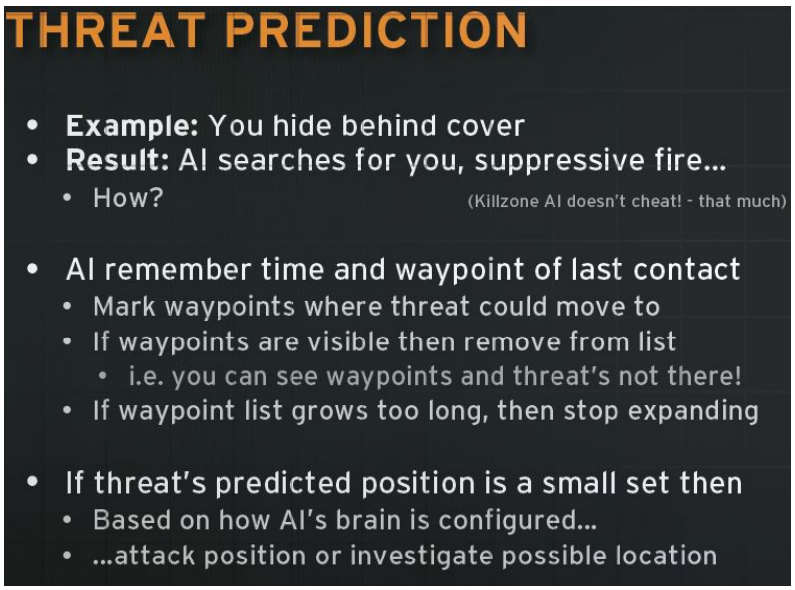


図 5.2-23　脅威予測のコンセプト[17]

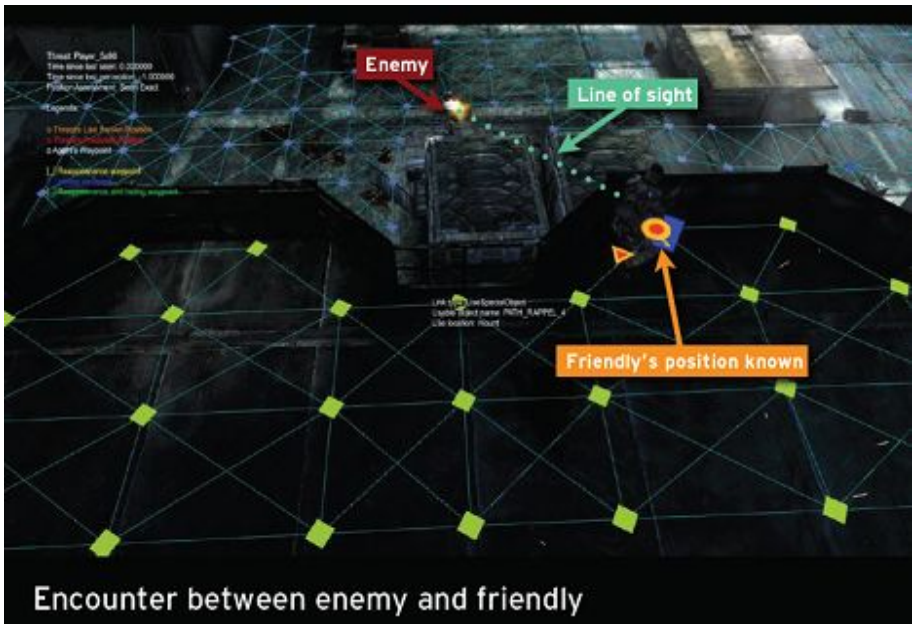


図 5.2-24　敵（赤）とプレイヤー（オレンジ）が階を挟んで対峙する[17,18]

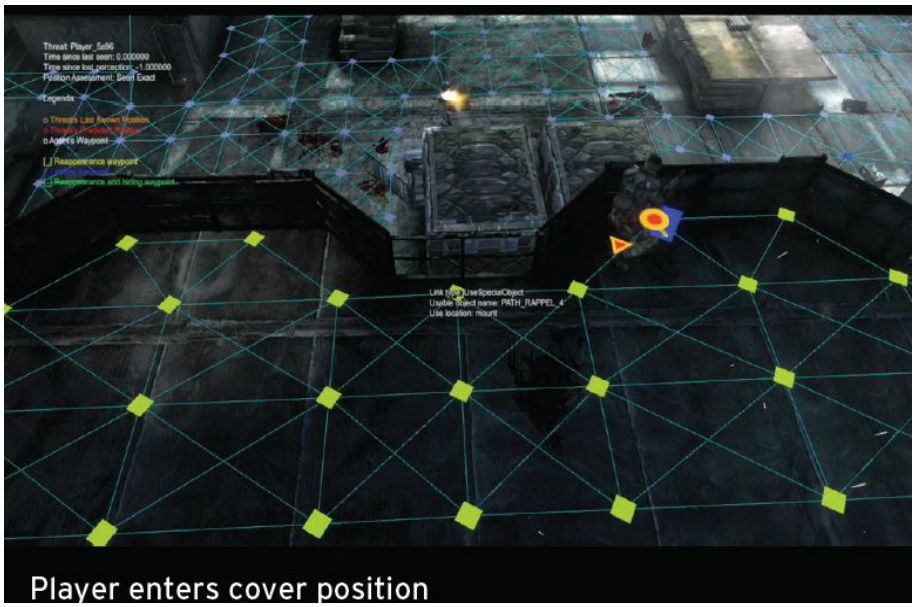


図 5.2-25　プレイヤーは屈むことで AI の視線から逃れる[17,18]

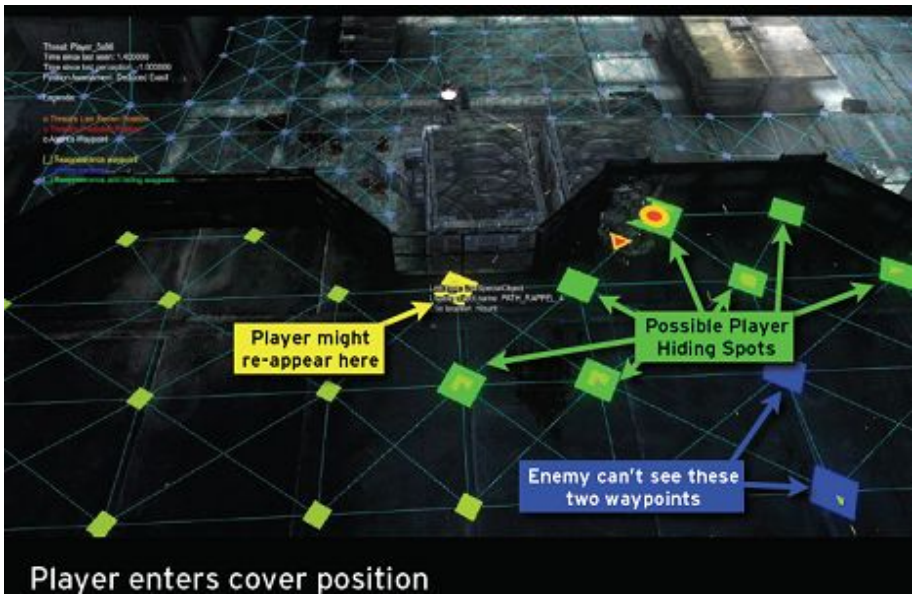


図 5.2-26　AI による推測[17,18]

プレイヤーを見失った AI は、最後にプレイヤーを見た場所と時刻を覚えていて、その位置を中心として経過時点に半径が比例する球領域のウェイポイントをプレイヤーの出現しそうな位置として候補に上げリスト化する。AI はそのリストの点においてプレイヤーがいないことを確認したらリストから削除して行き、プレイヤーのいる領域を探って行く

また、『KILLZONE 2』では、AI 同士でお互いの射線に入らないように工夫されている。これもウェイポイントを用いる。AI が向いている方向の射線領域をカプセル領域で表し、そのカプセル領域と交差するウェイポイントをマークし（おそらくポイントのコストを挙げる）、他の AI がそのウェイポイントをなるべく通らないようにするのである。

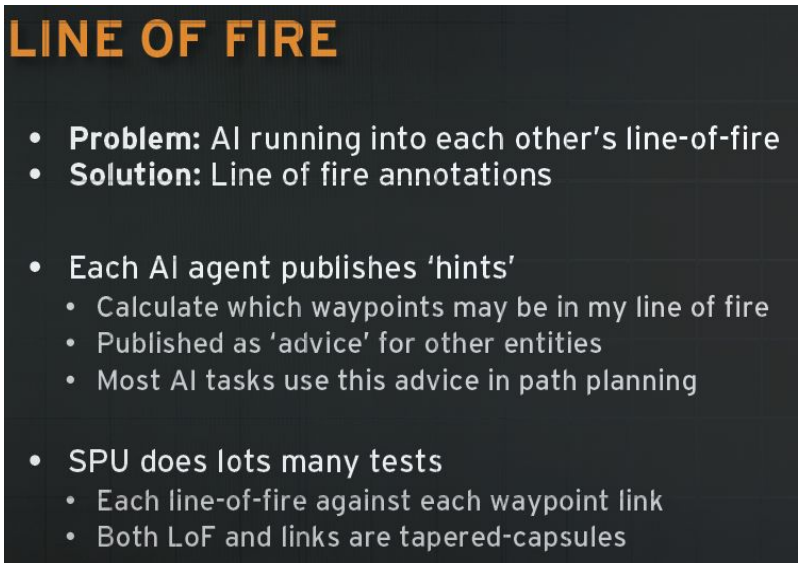


図 5.2-27　AI はお互いに味方の射線に入らないような動きをする[17]

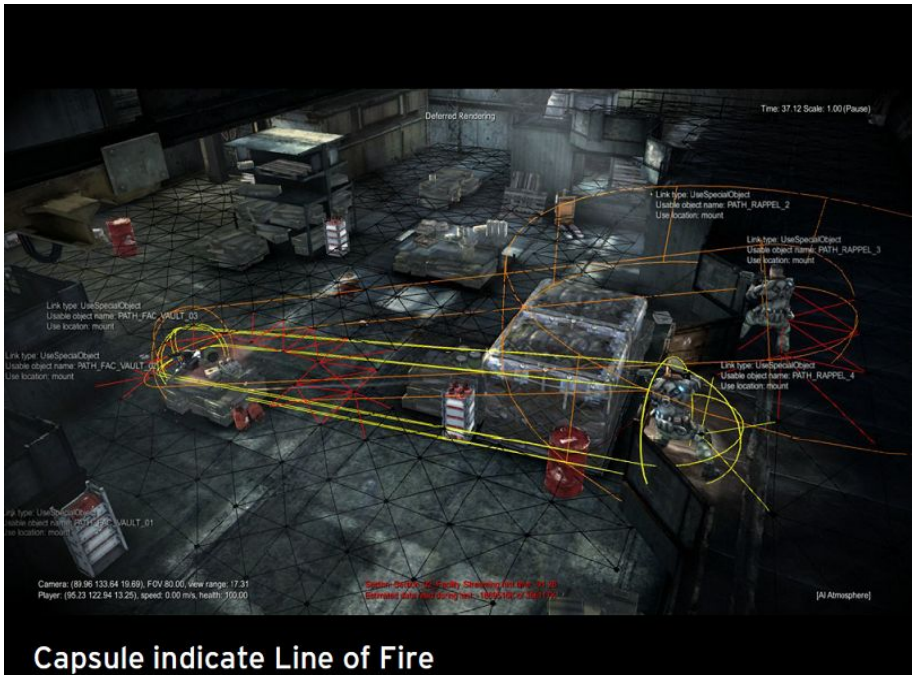


図 5.2-28　AI の射線領域のカプセルによる表現 [17]

図 5.2-29　進入禁止ウェイポイント[17]

AI の射線カプセル化領域と交差するウェイポイントには、他の AI が立ち入らないようにする。赤く印されているウェイポイントとエッジが進入禁止領域

5.3　レンダリングの新しいトレンドとシェーディングの新しいトレンド

5.3.1　Deferred Renderer　と Light Pre-pass Renderer

G-Buffer(Geographical Buffer) を使ったディファード・レンダリング(deferred rendering)とは、1988 年に、Michael Deering 氏によって、SIGGRAPH 1988 で発表された方法であり、最近、ゲームにおけるレンダリングで注目され応用されている方法である[19,20,21]。

まず Geometry に関する情報、深度, 法線, スペキュラ指数, albedo, baked lighting, gloss などを、MRT（マルチレンダリングターゲット）からなる G-Buffer に書き込んでおく。そして、この G-Buffer の情報から、後でライティングの計算を行う。つまり、ライティングはポストプロセス的に行うことになる。こういった手法は、『CryEngine 3.0』『Uncharted』『KILLZONE 2』[22]で用いられている。

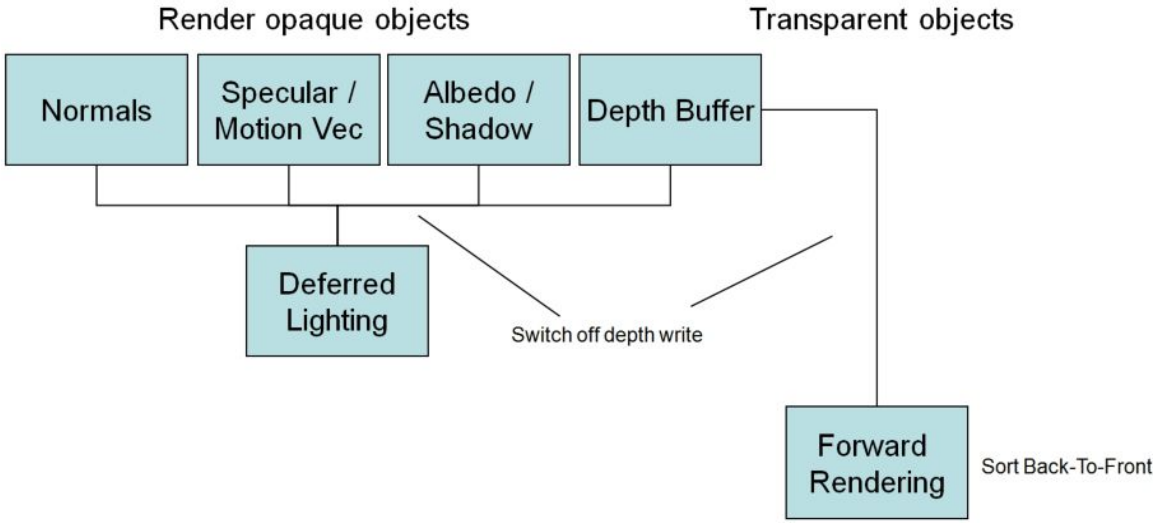


図 5.3-01　Deferred Lighting [19]

一方で、Light Pre-pass Renderer [23] [19]とは、深度と法線情報だけを 1 つか 2 つのレンダリング・ターゲットに保存し、次に全光源のライティング情報をライトバッファに保存し、これからライティングを行う手法である。これは、ライティングと他のレンダリングの属性を分離することで行うことが出来る（これには若干の議論があり、この方法については後から様々な手法が提案されている)。

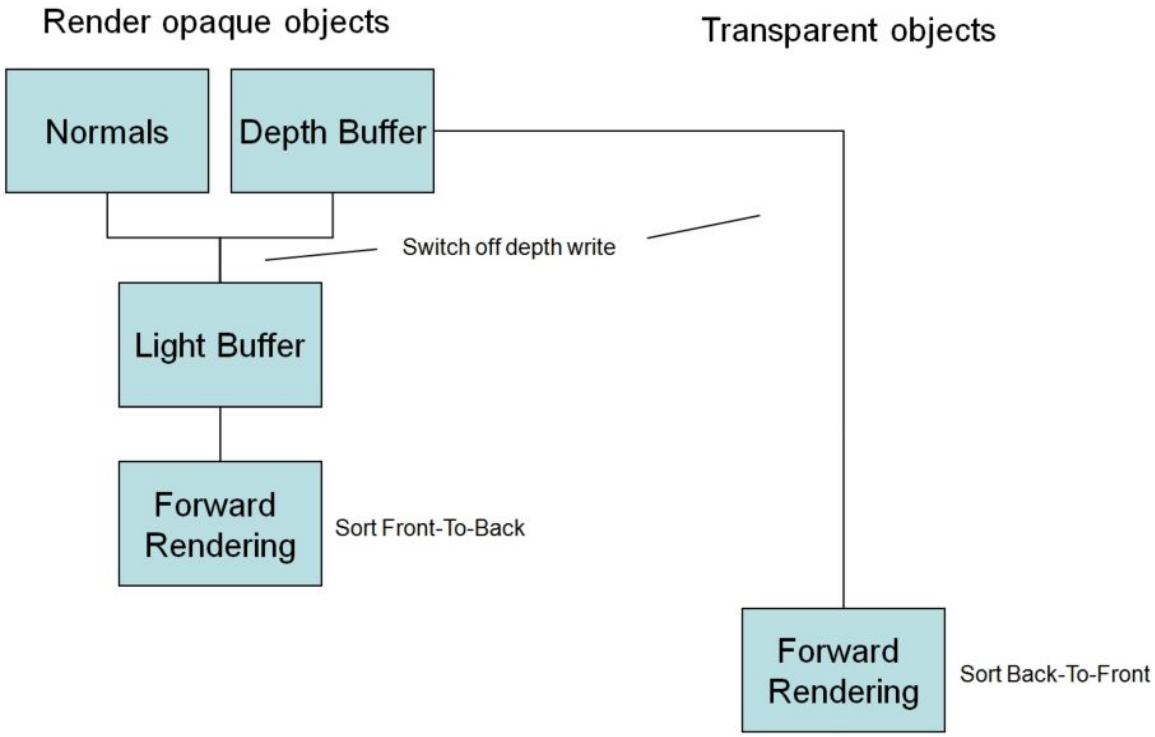


図 5.3-02　Light Pre-pass Renderer [19]

5.3.2　Resistance2　における Pre-rendering と deferred-rendering

『Resistance 2』で特徴的なのは以下の点である。

① 最初のジオメトリーパスでは、法線とスペキュラ指数のみをキャッシュする。
② screen space pre-lighting　を行う。
③ メインのジオメトリーパスをキュッシュする。
④ マテリアル特性などのレンダリングはセカンド・ジオメトリーパスで行う。

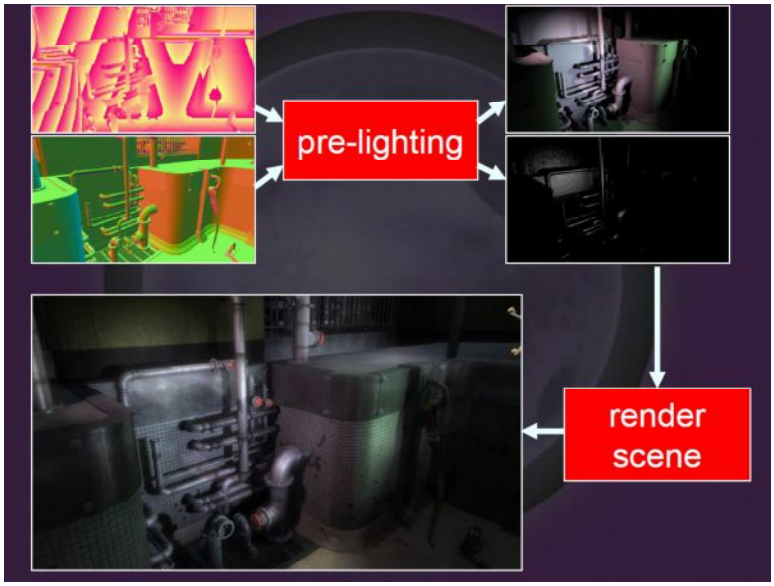


図 5.3-03　『Resistance 2』における Pre-lighting と deferred rendering[24]

全体のプロセスとしては、

(Step1)　深度と法線マップ

深度パスをレンダリングするときに同時に法線を書き込む。
法線情報は、プライマリー・レンダー・バッファーに格納する。
スペキュラ指数は、ノーマル・バッファのαチャンネルに保存する。

(Step2)　深度の解決

MSAA を非 MSAA にコンバート
非 MSAA のライティングとシャドウバッファーによる
ステンシリングによる最適化

(Step3)　サンシャドウの集積

全ての静的なオブジェクトのサンシャドウをあらかじめ計算しておく。

(Step4)　動的なライトの効果を加算する

ステンシルボリュームに基づいて各動的光源のライティングを加算する

(Step5)　レンダリング・シーン

サンシャドウはレンダリング後に処理される。
ライトレベルを最終的なライティングから定義し、
このレベルがある閾値を超えるとそのジオメトリに
シャドウがつけられる。

5.4　グローバル・イリュミネーション

グローバル・イリュミネーション（Global Illumination、大域照明、以下 GI）とは、照り返しを始め、あるオブジェクトのレンダリングに際して、他のオブジェクトからの影響を考慮してレンダリングする手法一般を指す。ゲームにおける GI は、拡散反射光の効果を取り入れた手法を指すことが多い。2008 年の CEDEC、この GDC でも発表があった、『ソニック ワールドアドベンチャー』[25]『メタルギア・ソリッド 4』[26]を始め、『KILLZONE 2』『CRY ENGINE3』『UNREAL ENGINE 3』『Miirror’s Edge』[21,27]など、次世代ゲームの主要なレンダリング手法の一つである。ただ、まともに計算すると膨大な計算を必要とするため、それぞれ GI の効果を事前計算を行ったり、リアルタイムで計算する場合は、GI の効果を再現する集約データを用意するなどの手法が取られている。

「KILLZONE　2」では特に、球面調和関数のモードを持った光源を幾つかマップ上に配置し、GI を擬似シミュレーションしている[17]。

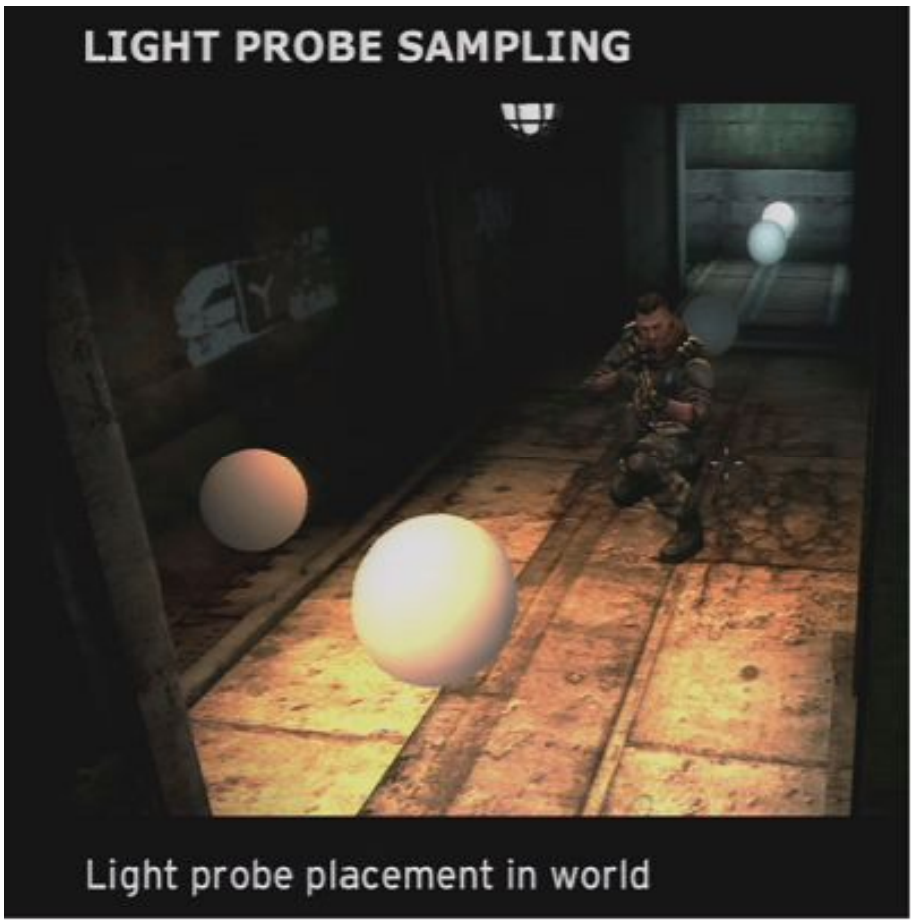


図 5.4-01　「KILLZONE 2」における Light Probe によるグローバル・イリュミネーション [17]

5.5　キャラクターアニメーションの精緻化とメタ AI

この数年に渡る米のゲーム AI の発展は飛躍的なものであり、その成果は昨年までに、『Halo3』など、大型タイトルを始め大きな成果として結実した。同時に、そこには、IGDA SIG-AI (AI 専門部会)を通じた 12 年に渡るゲーム AI 開発者コミュニティの発展があり、GDC 以外にも様々なカンファレンス、また、AI Game Programming Wisdom [28]を始めとする、優れたテキストを作成することで、知識・ノウハウを築いて来た。今年は、その中心メンバーが 「**AI Game Programmers Guild**」 を結成し、100 名近い著名なゲーム AI プログラマを中心に、より強固で綿密な技術ネットワークを形成している（ちなみに筆者で 101 番目のメンバー)。AI Game Programmers Guild は、1～2 日のチュートリアル・デイに、ゲーム AI に特化した講演、パネルディスカッションからなる「AI Summit」を開催した。…と言うと、このギルドがあたかもゲーム AI 技術全般を支配しているように思えるが、実際はそうでない。それは最大の中心の一つであるが、そこに属さずに、ゲーム AI 技術を研鑽し発表している企業は多いし、また日本やアジアのゲーム AI との結び付きもまだ弱い。ただ、このギルドは台風の目のように、これからのゲーム AI の最大の運動の中心であることは間違いない。

5.5.1　キャラクターアニメーションの精緻化

パス検索技術はこの数年でもはやデファクト・スタンダードになり、GDC2009 では、パスの軌道をキャラクターの運動特性に合わせて滑らかにしたり、キャラクターのゲームステージとの物理的相互作用を自然なアニメーションで繋いだり、そういった、キャラクターアニメーションをより精緻でリアルなものにしようという講演が多く見られた。
また、この分野は、Autodesk、Havok、Unity、Natural Motion といったミドルウェア・メーカーが揃って力を入れている分野でもある。3～4 年前までは、AI のミドルウェアは小さなミドルウェア会社が多かったが、大手のツールメーカーに吸収されるなどして、逆に大手のミドルウェア・メーカーが力を入れて存在感を大きくしつつある。GDC2009 のキャラクターAI については、本報告書のゲーム開発技術ロードマップの「プログラミング AI」の「2009 年のキャラクターAI」の項で解説を行っているので、そちらを参照して頂きたい。

5.5.2　メタ AI

メタ AI とはゲーム全体の状況をインタラクティブにコントロールする AI であり、ゲームそのものに知能が与えられたものである。従来であれば、ゲームのコントロールというものは、それこそ、ゲームプログラミングそのものであったわけだが、そういった事前に仕込んでおく静的なゲーム・ロジックではなく、むしろ、プレイヤーのゲームプレイン

グを常にモニターしながらゲームを動的に変化させて行くのがメタ AI の役割である。メタ AI に関しては、本報告書のゲーム開発技術ロードマップの「プログラミング AI」の「メタ AI」の項で解説を行っているので、そちらを参照して頂きたい。

5.6 プロシージャル技術

プロシージャルとは「手続き的」という意味であるが、プロシージャル技術は、「手続き的にコンテンツを自動生成する技術」、簡単に「自動生成技術」と呼ばれる。学術的には「autonomic generation」と呼ばれる場合もあるが、ゲーム業界では、自動生成という意味で、プロシージャルという言葉が用いられる。プロシージャル技術は、大規模開発におけるコンテンツの自動生成から、小規模開発におけるコンテンツ生成まで、また、ゲーム全体から、部分的な適用など、幅広い応用が目指されている。

以下、今年の GDC の発表内容に沿って解説する。

5.6.1 HALO WARS における地形自動生成

講演「HALO WARS: The Terrain of Next-Gen」において、「Age of Empire」シリーズで世界的なヒットを飛ばした Ensemble Studio が、スタジオの閉鎖の最後に製作した作品が「HALO WARS」である。「Age of Empire」では、地形を自動生成し、その地形をさらに自動解析することで、AI やオブジェクトを自動配置していた[29]。また「Age of Empire 3」では、段差のある 3D の地形を自動生成していた[30]。「Age of Empire」シリーズはは対戦モードを持っているため、毎回のゲームごとに地形が変化することが、プレイヤーたちに毎回戦術を変更させ考えさせる役割を持っていた。

「HALO WARS」（Xbox360）では、より広大なマップに地形自動生成のテクニックを発展させて使用しているが、これは開発段階で用いている[31]。方針としては、

① ハイトマップではなくベクターフィールドによって地形を生成する。
② Level of Detail は、テッセレーションにより対応。
③ 生成する地形は、ゲーム・シミュレーションとしては細かすぎるので、CPU のために、SimRep という低分解能データを、GPU のために VisRep という高分解能のデータを用意する。
④ ゲーム境界の向こうは節約のため折り返した図形を使う。

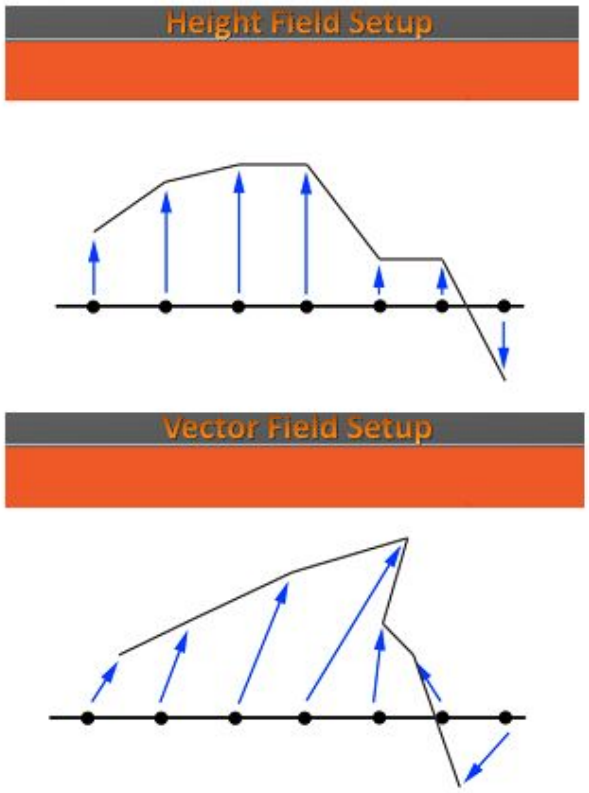


図 5.6-01　ハイトマップからベクターフィールドへ[31]

（『Halo Wars』の地形の各ポイントにはベクターが関連付けられていて、そのベクターの変位を使って地形の自動生成を行う）

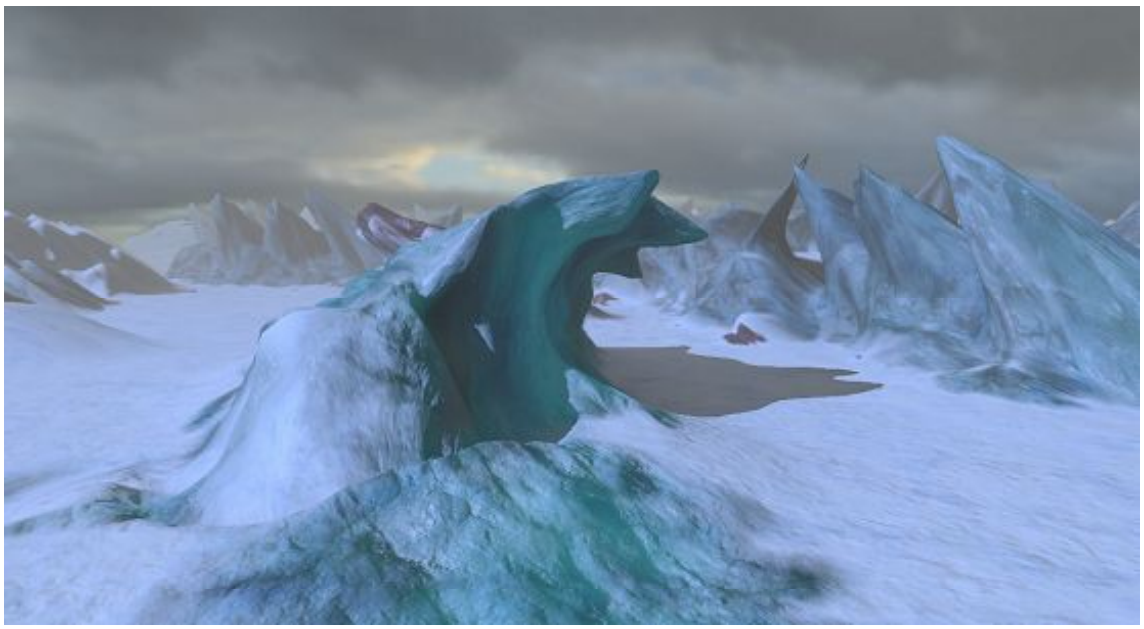


図 5.6-02　『Halo Wars』において自動生成された地形[31]

5.6.2　LOVE におけるオール・プロシージャルな世界

講演「Making LOVE in Your Bedroom」おいて、インディーズ・ゲーム（小規模独立開発）としてよく知られた、 Eskil Steenberg 氏が一人で作っている MMO「LOVE」の開発が紹介された[32] [33]。「LOVE」の特徴は、プロシージャルな生成を使ってコンテンツを自動生成することで小人数で大規模なコンテンツを作り出してしまうところにある。以下、その特徴的な点を挙げる。

①　コンセプト・アート　コストのために削ってしまう。自動生成では要らない。

②モデリング　　自作モデリングツール「Loq Airou」[34]は、sub-division の機能を持ち、ローポリゴンモデルをハイポリゴンモデルに自動変換してくれる。また、幾つかの簡単なモデルを組み合わせて、新しいモデルを作れようになっている。

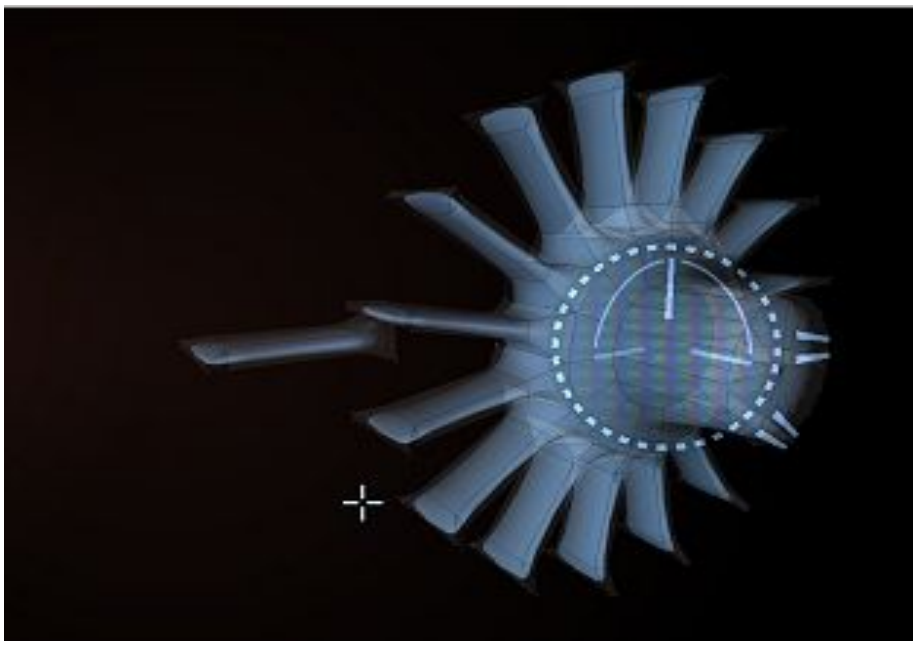


図 5.6-03　「Loq Airou」[34]

③テクスチャ切り出し　ローポリゴンモデルに関して 100％自動的な UV マッピングを行う機能を作成。

④テクスチャ　マテリアルベースで自動生成しようと画策中。

⑤アニメーション　自動的なものはまだない。「Spore」にようにしたい。

⑥レベル生成　大局はマニュアルで、詳細は自動生成がよい（これが「LOVE」のことなのかは不明）。

⑦パイプライン　verse [35]という自作パイプライン環境を構築。さまざまなツール環境で作成したモデルがリアルタイムでゲーム環境で再現される。また、Co On [36] というシーンエディターがある。

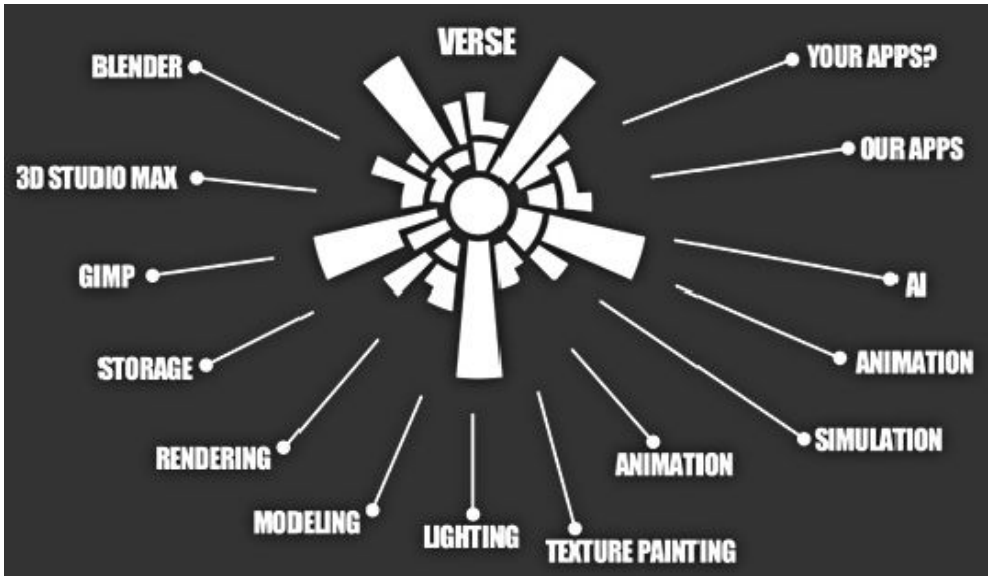


図 5.6-04　verse [35]

⑧ゲームエンジン　自動レベル生成。4 つのセルから、一つを隆起させ、その隆起した一部をオブジェクトで置き換えることでレベル生成を行う。

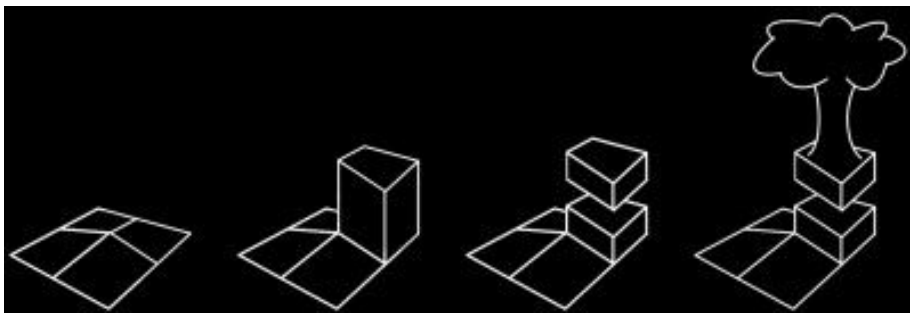


図 5.6-05　地形自動生成の原理[33]

講演内容からは、ゲーム内のランタイムにおける生成以上に、コンテンツ製作ツールの環境整備や手製の生成の円滑化に力を入れていることがわかる。

5.6.3　セミ・プロシージャルというアプローチ

通常、自動生成と言えば、アルゴリズムで完全に生成するか、或いは、ある程度の事前データから出発して生成するか、何れかであるが、講演「Dynamic Walking with Semi-Procedural Animation」では、アニメーターの作った 2 つ程度のアニメーションを精緻に解析し特徴を抜き出し、そこから新しく必要なアニメーションを生み出す「Semi-Procedural」と名付けられた方法を展開している[37] [38]。この研究を行ったのは、北欧の大学院修士の Rune Skovbo Johansen 氏[39]である。Unity [40] との共同研究であるこの成果は、Unity の 3D ゲームエンジンに組み込まれる予定である。

『Spore』におけるプロシージャル・アニメーションの手法は、「アニメーターが用意したヒントデータを利用してアニメーションを生成する」[41] [42]方法であったが、この手法は、

データ　⇒　データ解析　⇒　解析したデータに基づくデータ生成

という過程を取るという意味で新しい手法である。

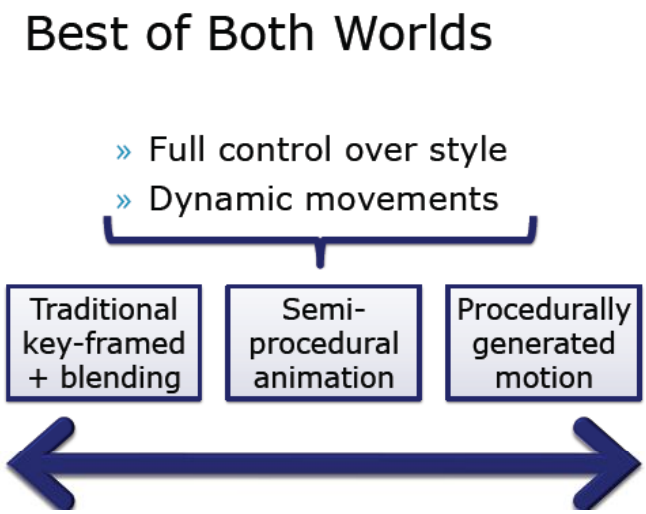


図 5.6-06　セミ・プロシージャルのコンセプト[37] [38]

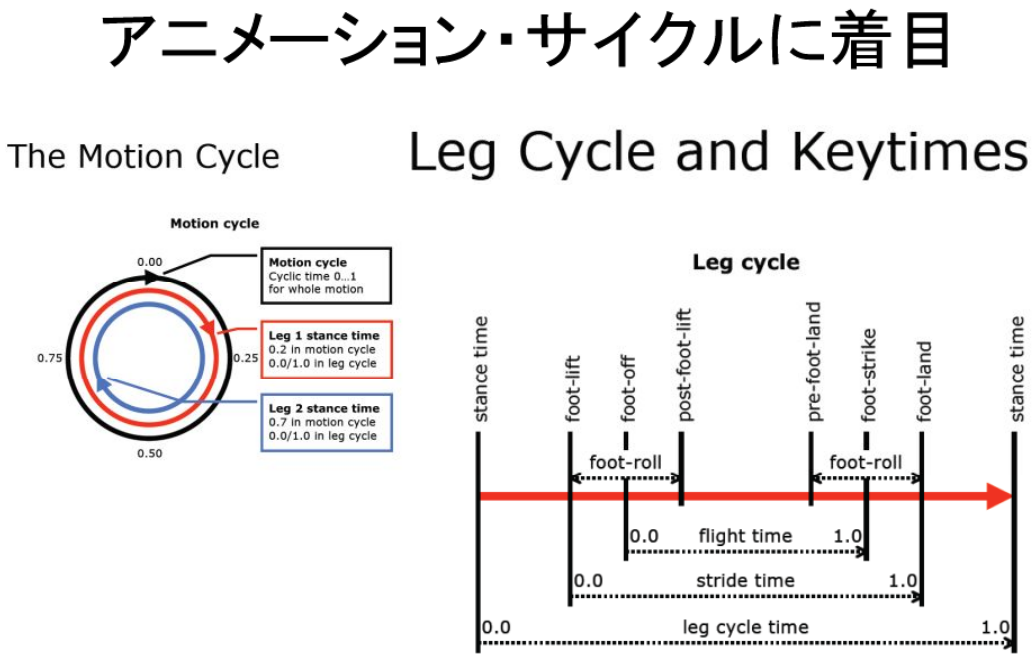
この手法はまず、アニメーターにキャラクターアニメーションを最低 2 つを用意させる。このアニメーションからデータ解析によって、

① 歩行サイクル
② 歩幅
③ 足先の軌道
④ 運動軸

など下半身の運動に必要なパラメーターを抜き出し、このパラメーターを使って、もとのアニメーションを状況に応じて変更し適用する。

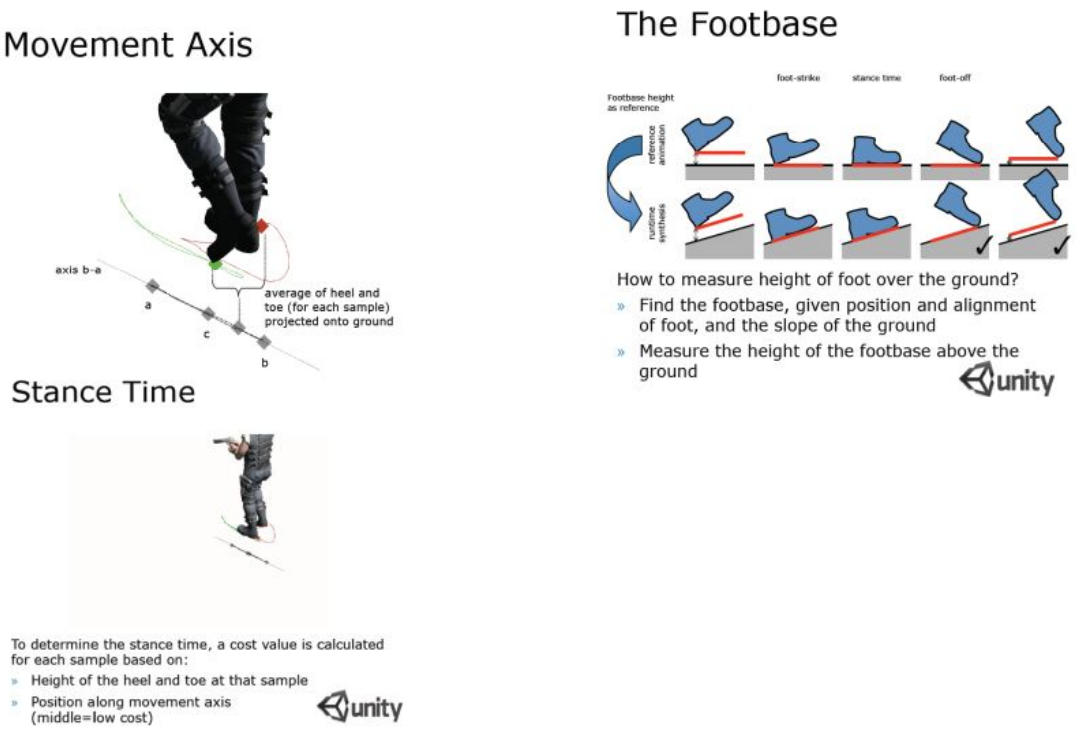
（Ⅰ） 地形に応じて足の置き方を変更しながら歩く。
（Ⅱ） 横へのステップの動きを生成する。
（Ⅲ） 段差に応じて足の運動を変更して歩く。

などが、自動的に可能となる。



http://cmpmedia.vo.llnwd.net/o1/vault/gdc09/slides/Rune_Skovbo_Johansen.pdf

図 5.6-07　歩行のサイクルを解析から求める[37] [38]



http://cmpmedia.vo.llnwd.net/o1/vault/gdc09/slides/Rune_Skovbo_Johansen.pdf

図 5.6-08　足の置き方を計算から求める[37] [38]

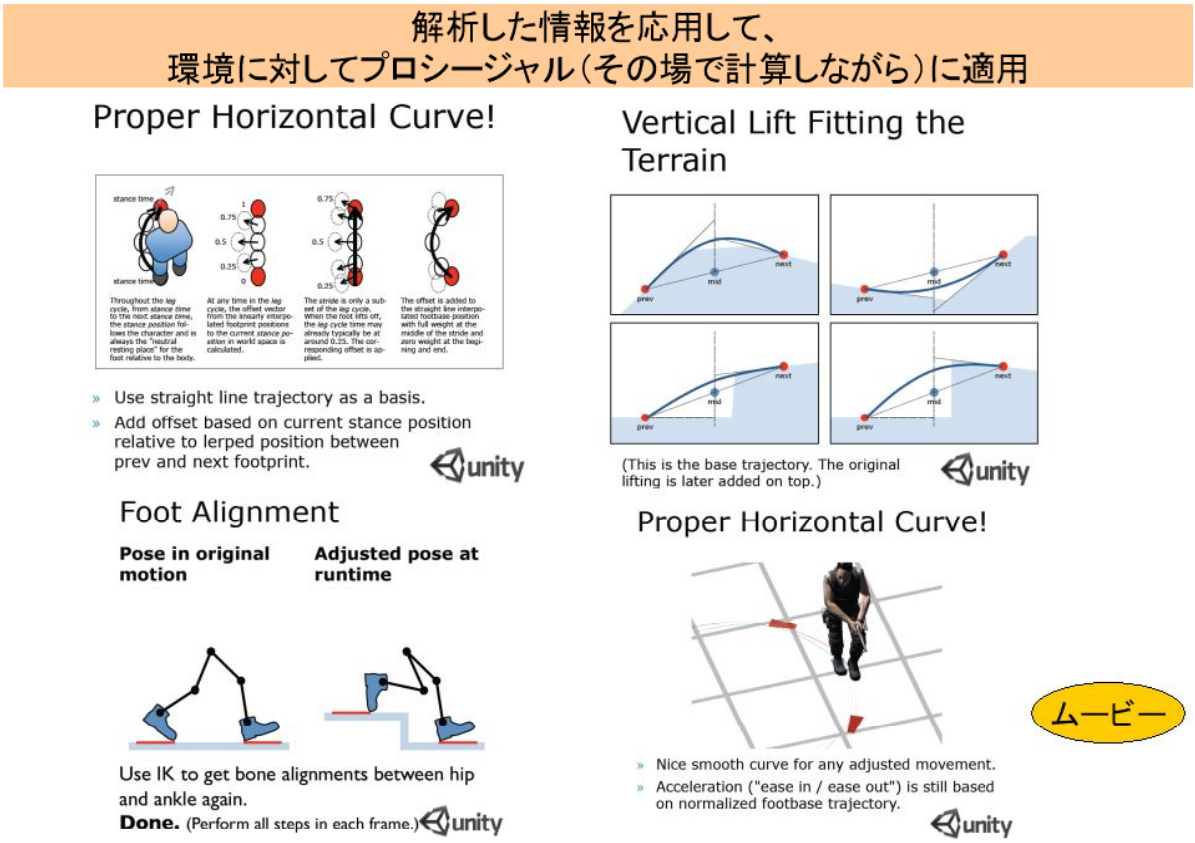


図 5.6-09　地形に応じた動作を生成する[37] [38]

5.7　インディーズ・ゲームにおける技術とゲームデザインの融合

インディーズ・ゲームとは、独立系（パブリッシングを持たない数人の）開発会社や、学生作品などの作品を指す。インディーズ・ゲームは、自ずから小規模なゲームになる傾向があるが、そういった小規模なゲームは、近年の大型ゲームへの反発や、携帯ゲーム機、WiiWare や、XboxLive のダウンロード販売、「STEAM [43]を通してのダウンロード販売」、iPhone など携帯電話機のゲームの普及を通じてビジネスモデルへ組み込まれ、大型ゲームが失速する中でも、この数年、ますます隆盛を見ている分野である。また、米の不況により大規模な開発会社による数千人規模のリストラによって、スピンオフしたベテラン開発者が独立系開発スタジオを立ち上げる流れが加速している。こういった様々な要因が重なって、ますますブームが加速していると言ってよい。逆に、小さなパイにタイトルが集中し過ぎたために、iPhone のようにビジネスモデルが危ぶまれる分野もある。

GDC では、Independent Games Festival [44]が開催され、独立系や学生のゲームなどに対して、さまざまな賞を用意しており、ここで表彰されることでビジネスチャンスを得

るタイトルが多い（『Darwinia』（Introverison）[45]など）。また、一般講演においても、独立系を対象とした講演や、Independent Games Summit [46]のように、丸一日独立系に限った講演とパネルの集合も開催されている。

こういったインディーズ・ゲームは、技術的に見ても、様々なチャレンジが出来る分野である。特に、大きな計算リソースを必要とする技術ではなく、「小さなアルゴリズムを工夫してゲームの面白さに直接結び付ける」という分野の技術である。上記で説明したプロシージャル技術を応用した「LOVE」や、2D で物理シミュレーション技術を応用した「WORLD OF GOO」[47]（2008 年 IGF デザイン・イノベーション大賞[48]）など、「ゲームデザインに応用技術が直接結び付いている」ことが小規模なゲームの大きな特徴である。これは、ゲーム開発を期間内に収束させるために、基礎エンジンを固めた上でゲームデザインを考えて行く大規模ゲームでは難しいことである。また小規模なゲーム開発は、開発サイクルが早いために、そういったアイデアを次々に試すことが出来るという利点を持つ。

こういった流れから、ゲーム産業は次のことに気が付きつつある。

① 大規模開発だからと言って何もかもが出来るわけではない。小規模開発でしか出来ないゲームデザインもある。
② 大規模開発は開発を収束させるため、ゲームエンジンを固定し、開発パイプラインを固定するなど制約が多い中で開発しているおり、柔軟性を失いつつある。
③ ユーザーにとって大規模であるか小規模であるかは関係なく面白さを見出せればそれでよい。そして、大規模なゲームにも小規模なゲームにもどちらにも需要がある。
④ 小規模なゲームでも、プロシージャルのように最先端の技術を、大規模なゲームに適用するのとは違った方法で適用することが出来る。
⑤ 小規模なゲームは、もはやレトロゲームの模倣ではなく、むしろイノベーションの場となりつつある。

デジタルゲームの発展というとき、ついゲーム開発者はこれまで大規模なゲームに目を奪われ過ぎて来た。しかし、小規模なゲームはデジタルゲームの発展が進む、もう一つの進化の方向かもしれない。GDC2009 はゲームの進化における、その明確な分水嶺を明確に示していた。

図 5.7-01　『WORLD OF GOO』の画面

（うまくグーをつなぎながらステージを渡っていく）

図 5.7-02　GDC2009 における Expo における IGF のコーナー

（熱心に受賞作の解説をするインディーズゲームの開発者たち）

5.8 References

[1] "久夛良木健氏が語る、PlayStation 3とCellの正体　（GameWatch）",
http://pc.watch.impress.co.jp/docs/2003/0901/kaigai015.htm
[2] "「PS3の次のステップは、Cellコンピューティング（PCWatch）",
http://pc.watch.impress.co.jp/docs/2005/0624/kaigai194.htm
[3] "「METAL GEAR SOLID 4」関連セッションレポート その2 次世代機という"理想"と、PS3という"現実"の狭間でもがくエンジニア達(GameWatch)",
http://game.watch.impress.co.jp/docs/20080912/cedec_mgs.htm
[4] "西川善司の3Dゲームファンのための「METAL GEAR SOLID 4」グラフィックス講座 職人芸的最適化術によって生まれたPS3最高峰グラフィックスの秘密に迫（前篇）(GameWatch)", http://game.watch.impress.co.jp/docs/20081203/3dmg4.htm
[5] "第2回ゲーム産業開発者のためのスキルアップ講座（CESA）",
http://www.cesa.or.jp/news/2007/20070202Skill.html　(2007)
[6] （株）ソニー・コンピュータエンタテイメント　井上敬介：　"Cell Broadband Engineの性能を自然に引き出すための考え方と実行環境",
http://research.cesa.or.jp/pdf/shiryo6-2-3.pdf
[7] "CELL BROADBAND ENGINE", http://cell.scei.co.jp/
[8] "「PLAYSTATION Edge」講座
ソニー謹製プレイステーション 3専用フレームワークが登場!(GameWatch)",
http://game.watch.impress.co.jp/docs/20070316/pe.htm
[9] "Insomniac Games", http://www.insomniacgames.com/
[10] "Insomniac Games's Secrets of Console and Playstation 3 Programming",
https://www.cmpevents.com/GD09/a.asp?option=C&V=11&SessID=8543
[11] "Insomniac Games R &D", http://www.insomniacgames.com/tech/techpage.php
[12] Insominiac Games：　"Nocturnal Initiative",
http://nocturnal.insomniacgames.com/index.php/Main_Page
[13] Insominiac Games：　"GDC09 - Gameplay Systems on SPUs",
http://www.insomniacgames.com/tech/articles/0409/GDC_2009_spugp2.php
[14] Insominiac Games：　"GDC09 - Insomniac Physics",
http://www.insomniacgames.com/tech/articles/0409/Physics_GDC_2009.php
[15] Jim Tilander (Game Programmer, Sony, Santa Monica), Vassily Fillipov (Lead Game Programmer, Sony, Santa Monica)：　"Practical SPU Usage in GOD OF WAR 3", https://upload.cmpevents.com/GD09/a.asp?option=C&V=11&SessID=8766 (2009)
[16] Jim Tilander, Vassily Filippov (Sony Santa Monica)：　"Practical SPU

Programming
in God of War II",
http://www.tilander.org/aurora/comp/gdc2009_Tilander_Filippov_SPU.pdf　(2009)
[17] Michiel van der Leeuw：　"The PlayStationR3's SPUs in the Real World",
http://mygdc.gdconf.com/vault/play/963　(2009)
[18] 西川善司：　"西川善司の3Dゲームファンのための「KILLZONE 2」グラフィックス講座(後編)　～オランダ精鋭部隊がCELLプロセッサで実装した「脅威推測型AI」の秘密（GameWatch)",
http://game.watch.impress.co.jp/docs/series/3dcg/20090424_153727.html
[19] Wolfgang Engel：　"Renderer Design", http://www.wolfgang-engel.info/RendererDesign.zip
[20] "Deferred Rendering in Killzone 2 -Guerrilla Games", http://www.guerrilla-ames.com/publications/dr_kz2_rsx_dev07.pdf
[21] 西川善司：　"GDC2009　3Dグラフィックス最新事情", http://www.z-z-z.jp/zenji/IGDA_GDC2009.zip
[22] 西川善司：　"西川善司の3Dゲームファンのための「KILLZONE 2」グラフィックス講座(前編) (GameWatch)",
http://game.watch.impress.co.jp/docs/series/3dcg/20090417_125909.html
[23] Wolfgang Engel：　"Light Pre-Pass Renderer",
http://diaryofagraphicsprogrammer.blogspot.com/2008/03/light-pre-pass-renderer.html
[24] Insominiac Games：　"Resistance 2 Prelighting",
http://www.insomniacgames.com/tech/articles/0409/GDC09_Lee_Prelighting.php
[25] "セガ、新作「ソニック　ワールドアドベンチャー」におけるライティング技法を公開　次世代水準"の映像美を実現するグローバルイルミネーション（GameWatch）",
http://game.watch.impress.co.jp/docs/20080912/sonic.htm
[26] 西川善司：　"西川善司の3Dゲームファンのための「METAL GEAR SOLID 4」グラフィックス講座 職人芸的最適化術によって生まれたPS3最高峰グラフィックスの秘密に迫る(前編) (GameWatch)",
http://game.watch.impress.co.jp/docs/20081203/3dmg4.htm
[27] "BEAST", http://www.illuminatelabs.com/products/beast
[28] Steve Rabin：　"AI Game Programming Wisdom", Charles River Media
[29] Dave C. Pottinger：　"Terrain Analysis in Realtime Strategy Games",
http://www.directxnedir.com/pottinger.doc　(2000)
[30] "3DゲームファンのためのAGE OF EMPIRESエンジン講座(後編)",
http://game.watch.impress.co.jp/docs/20050313/aoe3.htm　(2005)
[31] Colt McAnlis (Microsoft Ensemble Studios)：　"Halo Wars: The Terrain of Next

Gen", http://www.bonfire-studios.com/files/GDC_09_HW_Terrain_final.zip
[32] "Interview: Eskil Steenberg On Why LOVE Is All You Need (Gamasutra)", http://www.gamasutra.com/php-bin/news_index.php?story=22960 (2009)
[33] Eskil Steenberg： "Making Love (GDC2009 Lecture)", http://news.quelsolaar.com/#post41
[34] Eskil Steenberg： "Loq Ariou", http://www.quelsolaar.com/loq_airou/index.html
[35] Eskil Steenberg： "verse", http://www.quelsolaar.com/verse/index.html
[36] Eskil Steenberg： "co on", http://www.quelsolaar.com/co_on/index.html
[37] Rune Skovbo Johansen： "Locomotion System", http://runevision.com/multimedia/unity/locomotion/
[38] "Make characters dynamically walk and run on any uneven terrain (Unity)", http://unity3d.com/support/resources/example-projects/locomotion-ik
[39] Rune Skovbo Johansen： "runevision.com", http://runevision.com/
[40] "Unity3D", http://unity3d.com/
[41] Chris Hecker： "Real-time Motion Retargeting to Highly Varied User-Created Morphologies", http://chrishecker.com/Real-time_Motion_Retargeting_to_Highly_Varied_User-Created_Morphologies
[42] 倉地紀子： "Sporeのモーションリターゲット・パイプライン", CGWORLD 2009年1月号 , 125
[43] "STEAM (Valve)", http://store.steampowered.com/?l=japanese
[44] "Independent Games Festival", http://www.igf.com/
[45] Introversion： "Darwinia", http://www.introversion.co.uk/darwinia/
[46] "The Independent Games Summit", http://www.indiegamesummit.com/
[47] "WORLD OF GOO", http://www.worldofgoo.com/
[48] "2008 Independent Games Festival Winners", http://www.igf.com/2008finalistswinners.html

第 6 章　海外におけるゲーム関連技術教育のマニュアル

6.1 「IGDA カリキュラムフレームワーク 2008」の翻訳意図について

「IGDA カリキュラムフレームワーク 2008 年版」が、この報告書に収録される意図としては、一つには、日本語訳も行われた「2003 年版」から、5 年が経過する間に、欧米圏のゲーム開発の方法論は、かなり成熟しつつあるということがある。

このフレームワークで示されている内容は、広範囲にわたっており、ドキュメントの中でも示されているように、すべての大学のシステムで、すべての内容を含めることができるカリキュラムを編成することは物理的に難しいと考えることができる。

それでも、この「カリキュラムフレームワーク」が重要であるのは、それがゲーム教育の方向性を指し示すと同時に、教育を通じてゲームが意味している全体像を示している意味合いがあるからである。

「2003 年版」がリリースされたあと、2005 年にこのカリキュラムフレームワークの内容を反映した「Introduction to Game Development」（Charles River Media）がリリースされた。これはゲームの教科書となることを意図されて編纂された書籍で、27 人からなる執筆者が、カリキュラムフレームワークの各トピックに対応する形で、新規に執筆を起こしたものから、他で執筆されたものを再収録したものまで、コンテンツは様々であるが、全体としては 980 ページにも及ぶ大著となっている。

各トピックスに沿って、教科書がまとめられているため、包括的かつ体系的な内容になっているところにも特徴がある。また、どのトピックスを選択して特定のゲームコースをデザインすればよいかといった提案まで行われている実践的な内容にもなっている。

これは、当然、将来のゲーム開発の技術の方向性を決定づけていくものであると述べることができる。「カリキュラムフレームワーク 2008 年版」の登場は、新しい教科書の改訂を促すものと考えることができ、それは技術の潜在的な成長性をくみ取るできるものと考えることができる。

日本では、こうした「カリキュラムフレームワーク」の策定作業と、それに続く「ゲームの教科書」の策定作業が試みとしても、本格的には実施された例はない。そのため、今回の翻訳は、日本のゲーム開発者教育と、また、技術的な将来性を検討するための俯瞰図を獲得するために必須となるドキュメントとすることを意図して、実施した。

また、IGDA の教育 SIG は、2009 年 1 月 30 日から 2 月 1 日にかけて、「Global Game Jam」という 48 時間耐久で、全世界で同時にゲームを開発するイベントを開催して、1600 人以上の参加者を獲得し、大きな成功を収めた。ゲームデザイン訓練を運動として実践する活動を行うことで、教育の意味を拡張する試みも行っている。

IGDA カリキュラムフレームワークの日本語訳は、参考資料として、本報告書最終ページに記載している。

第 7 章　日本のゲーム関連技術教育についての課題と提案

7.1　ゲーム関連技術教育の現状と課題

7.1.1　ゲーム関連技術の編纂と蓄積

2008・2009 年という、いまや国策として日本のコンテンツを世界へ届けることで、より多くの人々に質の高い娯楽と感動を与えようとする時代に、コンテンツを製作する技術に着目することは、日本のゲームコンテンツを今後、より高い品質で効率よく生産する基盤を構築すると共に、コンテンツに関わる開発者の専門性の高いキャリアパスを構築し、コンテンツ産業が良い人材を獲得し、世界に誇る仕事を築いて行くために必要なことである。

ところが、30 年を超える歴史を持つ日本のゲーム産業における輝かしいゲームコンテンツの歴史に対し、ゲーム関連技術に関しては、その歴史が編纂・集積されることが少なかった。これは、ゲーム産業全体で共通に意識されていることでもあり、日本のゲーム産業を切り開いた偉大な先達が定年退職などによって引退を迎え始めるようとしている 2009 年という時期に至っては、失われて行くゲーム関連技術史に対し、その重要さを痛感すると共に、益々、危機感を強くするものである。現在の開発技術、ノウハウの共有化については、（社）コンピュータエンターテインメント協会（Computer Entertainment Suppliers Association、CESA）、日本デジタルゲーム学会（Digital Game Research Association JAPAN、DiGRA JAPAN）、ゲーム開発者協会日本（International Game Developers Association JAPAN、IGDA JAPAN）、有限責任中間法人ブロードバンド推進協議会（Broadband Association、BBA）、（財）デジタルコンテンツ協会（Digital Content Association of Japan、DCAJ）などにおける技術講演、技術セミナーによって展開されてはいるものの、失われた歴史全体を再構築することは容易なことではない。しかし、この困難な仕事はゲーム産業の今後の発展に重要な役割を担っており、本委員会委員の賛同を得て、第一のステップとしてゲーム開発技術の歴史を俯瞰する全体像を描くことを一年を通じての目標と定めた。

7.1.2　ゲーム関連技術の歴史的俯瞰と国内外の動向

まず、技術各分野において第一人者と目される開発者に、その分野の歴史と展望をまとめて頂いた（第 3 章「ゲーム開発技術ロードマップ」）。それぞれの解説には、未来の発展の方向を含めた技術ロードマップも用意されており、一見してその分野の歴史が俯瞰出来るようになっている。そして本文では、その解説が詳細に為されている。本解説は技

術者のみならず、ゲーム産業とゲーム開発を導いて行かれる立場にある方にも、ゲーム産業の政策に積極的に関わって行く方にも、今後の技術動向を見定める有用な情報を提供するものである。

また、欧米における開発技術について、IGDA Ed-SIG（Education-Special Interest Group, International Game Developers Association、教育専門部会）がゲーム開発技術教育に必要な事項をまとめた文書「IGDA Curriculum Framework」（2008 年度版）の翻訳を、第 6 章とその付録に掲載した。この文書は非常に精緻な技術項目リストであり、実際に複数の大学で採用されているものである。

さらに、世界全体の技術動向に関しては、毎年開催される世界最大のゲーム開発者のカンファレンス、GDC（Game Developers Conference、毎年 3 月にサンフランシスコで開催）における技術発表について、その全体の動向を第 5 章にまとめた。GDC の技術情報は、最新のゲーム開発技術情報であり、「世界の技術動向は毎年 GDC を観測していればわかる」と言われるほど、日本を含め世界中からレベルの高いゲーム開発技術情報が集約されている。また、本カンファレンスは技術発表の場であると同時に、ゲーム開発技術の潮流を作っていく場として機能していることにも注目したい。講演による情報共有のみならず、ディスカッションや交流を通して、これからの技術の方向を開発者全体で形作って行く役割を果たしているのである。

一方で、国内における技術潮流について考えるとき、日本は多数の優秀な技術開発者を擁しているにも関わらず、未だそれぞれの持つベクトルを集約しゲーム産業全体を活性化させる大きな潮流を形成するまでには至っておらず、これから、これまでの意識の高い人々による努力の上に、より高く広い技術交流の場が構築されて行くことで、世界に存在感を持つ技術潮流を形成するために進んで行かなければならない。

7.1.3　ゲーム関連技術教育の現状

日本のゲーム産業では教育に高い意識を持ち、充実した社内教育を整備している企業も少なくない。本報告書の第 4 章では、そのような企業に協力をお願いし、社内教育に関するインタビューを実施した。何れの企業も非常に先進的な取り組みを行なっており、ゲーム産業における社内教育がさらに充実して行くために必要な高い見識が含まれている。

また、日本のゲーム開発技術の潮流をこれから形成して行くためには、これまで、日本のゲーム開発技術はどのような流れを辿って来たのかを知ることが重要である。そこでゲーム業界において 30 年近いキャリアを持つベテラン開発者に開発技術の歴史についてヒアリングを行なった。そのインタビューの内容は非常に驚くべきものであり、1980 年代初頭、すでに現在のゲームフレームワークを先取りする技術フレームが確立されていたことがわかった。そして、その系統は今もなお「タスクシステム」として発展を続けているものであり、今回の技術調査の大きな成果の一つとして、日本のゲーム開発技術史に一

本の系統を見出したといえる。

今回の調査を通じて、これまで手がつけられて来ることの殆どなかったゲーム開発技術編纂という困難な仕事においては、その全体像を一旦、把握することさえ大きな仕事であることを痛感した。そのため、最初のステップとしても、本書は大部なものになった。しかし、とりあえず全体像を漏らすことなく大きくつかもうとする最初のステップは完遂したと考える。

7.1.4　ゲーム関連技術教育の課題

以上の調査報告をふまえて、ゲーム関連技術教育の課題を以下のように要約する。

(1)　ゲーム関連技術の対象の柔軟な確定

ゲーム関連技術は、なによりも IT に他ならない。したがって、ゲームの開発に関する技術の広がりと深みに対して、柔軟に考える必要がある。つまり、10 年前にはゲームに関連があるとは思われなかった技術が、現在あるいは未来には、ゲーム開発の中核的な技術になっているということが十分にありうる。IT としてのゲーム関連技術の進歩とともに、その広がりと深みにも配慮してゲーム関連技術の対象の変化にも留意しなければならない。

(2)　ゲーム関連技術の応用・汎用性への配慮

ゲーム関連技術は、たんにゲーム開発にのみ使用されるものではなく、アニメや映画など他のコンテンツタイプやコンテンツジャンルの制作にも共通するものである。また、コンテンツ制作だけでなく、広く創造的活動にも活用されるべきものである。まず、ゲーム関連技術の社会的有用性、汎用性について認識されなければならない。

(3)　ゲーム関連技術の歴史の記録と共有化

文化系であれ理科系であれ、あらゆる学術分野の根底には歴史的経緯の編纂と蓄積、さらにそれらに対する歴史的考察が存在する。現在ある技術の由来・来歴を知ることが、それを未来に展開していく場合に重要である。ゲーム関連技術においても、本調査でおこなった整理を基礎に、ゲーム関連技術の歴史をより詳細に記録する作業を着実に進めていかなければならない。

(4) 教育機関におけるゲーム関連技術教育の明示化

ゲーム関連技術教育は、わが国の場合、専門学校・ゲームスクールや大学・大学院で行なわれてきた。また、ゲーム関連の企業内教育の役割も重要である。それぞれの教育機関や教育制度の目的や特質に応じたカリキュラム編成、指導者構成を実現させなければならない。一般に、専門学校やゲームスクールにおいてはゲームの名を冠した学科・コースが設置されているのに対して、大学・大学院においてはそのような例はまだ少ない。ゲームの名を冠することなく、大学や大学院でゲーム関連技術教育が行われていることを、その内容とともに明示化することも重要である。また、大学・大学院においては、ゲーム関連技術教育と同時に、もうひとつの役割であるゲーム関連技術研究の推進も期待されるとともに、教育と研究の連携・統一が図られるべきである。

(5) ゲーム関連技術教育における産学連携の推進

ゲーム関連技術教育においては、産業界と教育機関の連携が決定的に重要である。とくに、カリキュラムと指導方法の開発、指導者の確保は、産学連携に期待される大きな役割である。米国などでは、これらの点における産学連携が進んでいるが、わが国の場合はまだ少ない。連携の課題を明確にして、ゲーム関連技術教育の分野でも産学の連携を推進することが求められる。

(6) 技術セミナー、カンファレンス、研修開催への支援の強化

教育機関におけるゲーム関連技術教育よりも実践的な教育の実現の機会として、企業内教育が期待される。企業内教育は人材マネジメントの一環として行なわれるのが通例である。わが国のゲーム産業においては企業内教育の方法と内容が外部に対して公開されることは少ないが、企業の枠を超えて同様の成果を期待できる場として、すでに各種の技術セミナーやカンファレンスが充実しつつあり期待されている。今後は、これらの活動に従事されている機関や個人への理解を進めるとともに、活動の継続と発展のために支援を強化していくことが重要である。

如上の課題意識の下に、具体的なゲーム関連技術教育の進め方について、次節でいくつかの提案を行なうこととする。

ところで、本報告書は、今後、日本のゲーム開発技術を詳細に紐解いて行くための大きな基礎となり、俯瞰地図を提供するものと自負している。また、ゲーム産業に関わる方々、ゲーム産業を推進する立場にいる方々は、本報告書を活用し、これから日本が世界

に存在感を持つゲーム開発技術の潮流を形成して行くための基本情報として活用して頂くことを希望する。

7.2　ゲーム関連技術教育についての提案

近年の劇的な技術革新、国際競争の激化、多様化する顧客ニーズなどの経営環境の変化を背景として、ゲーム産業においては、とりわけゲーム関連技術教育の重要性が認識されつつある。そこで、本節では、これまでの調査から明らかになった知見と前節で述べた現状と課題に基づいて、ゲーム関連技術教育の前進のために以下のように提案する。

7.2.1　新卒採用者に対するゲーム関連技術教育の提案

新卒採用者には 1 ヶ月から半年程度の期間、入社後 2 年～5 年の先輩社員（教育担当者）のもとで、ゲーム開発に関する専門的な教育を実施している。例えば、プログラマーに対しては C++、グラフィッカーに対しては、Maya や 3D STUDIO MAX、Photoshop などの画像処理ソフトの使用法、企画職に対しては企画の立案法などの基礎を習熟させるための教育が施されている。しかしながら、これらは単なるツール教育ではなく、ゲームデザイン教育という観点が重要であることはいうまでもない。これらの基礎教育を経た後、プロジェクトでの実践を通じて、ゲーム開発者のスキルアップが行われている。学校教育と新卒採用者対象教育のシームレスかつ段階的なゲーム関連技術教育を提案したい。

7.2.2　中途採用者に対するゲーム関連技術教育の提案

中途採用者には即戦力としての役割を期待しており、実践をベースとした属人的な学習となっていることから、リカレント教育の充実が今後の課題として提示されている。とりわけマネージャーやリーダーの育成という観点から、プロジェクト・マネジメント教育が緊急の課題となっている。なぜならば、優秀なマネージャーやリーダーの育成が組織力を強化させ、ひいては技術開発力の強化にもつながるものと認識されているからである。しかしながら、ゲーム開発者のキャリア志向として、マネジメント（管理職）よりもエキスパート（専門職）を目指す者が多い傾向にある。したがって、マネジメントとエキスパートのキャリアラダーの構築と、ゲーム開発者の自律的なキャリア選択が課題として提示されている。属人的学習の利点も活用し、より専門的・実践的な中途採用者対象のゲーム関連技術教育を提案したい。

7.2.3　経験からの学習の促進の提案

ゲーム開発会社では、社内勉強会が定期的に開催されており、ゲーム開発者間の協働を促す仕組みづくりがなされている。しかしながら、このような社内勉強会は業務補完的な位置づけであり、経験からの学習が最も有用であるという指摘もある。これは、プロジェクトにおける省察的実践の重要性を標榜するものでもある。つまり、単に経験の幅を広げるだけではなく、いかに経験から省察的に学習するかということが重要なのである。このような背景から、階層別研修の導入など、定期的な省察の機会が整備されつつある。企業内教育・研修においては、経験に基づく学習と合理的に体制化・制度化された教育・研修の融合の実現を提案したい。

7.2.4　語学教育と Off-JT の提案

近年、ゲーム開発に関する海外文献の出版が著しく拡大しており、日本のゲーム開発技術の低下が懸念されている。そのような中、開発者の語学力（とりわけ英語）の向上は、ゲーム開発技術の獲得という観点から、企業内、ひいては業界全体においてその重要性が高まりつつある。企業内における対策としては、TOEIC による英語能力の把握や、資格取得奨励金などの外発的動機づけ、英語学習に対する費用の一部負担などの施策が行われており、人的資本への投資が活性化しつつある。

また、国内であれば CEDEC（CESA Developers Conference）、海外であれば GDC（Game Developers Conference）、SIGGRAPH（Special Interest Group on Computer GRAPHics）などへの日本人ゲーム開発者の参加が、飛躍的に拡大してきている。これらの Off-JT で得られた知識や技術は、イントラネットを活用したリファレンスガイドの作成・提供を通じて、社内のゲーム開発者のスキルアップに活用されている。語学教育の重視と Off-JT の機会の増大を提案したい。

7.2.5　カリキュラムフレームワークの邦訳とその活用の提案

IGDA Education SIG（(国際ゲーム開発者協会教育専門部会）は、2003 年 2 月 25 日に、『IGDA Curriculum Framework』（Version 2.3beta）を策定し、2005 年 6 月 30 日にこのカリキュラムフレームワークの内容を反映した『Introduction to Game Development』Charles River Media を刊行した。これはゲーム開発教育の教科書となることを意図されて編纂された書籍で、27 人からなる執筆者が、カリキュラムフレームワークの各トピックに対応する形で、新規に執筆を起こしたものから、他で執筆されたものを再収録したものまで、内容は多岐にわたり、全体で 980 ページにも及ぶ大著となっている。

その後、2008 年 2 月に『IGDA Curriculum Framework』（Version 3.2beta）が策定

され、本調査報告書において邦訳された。このカリキュラムフレームワークは、新しい教科書の改訂を促すものであり、それは技術の潜在的な成長性を汲み取ることができる内容となっている。具体的には、「ゲームとは何か」というゲームカリキュラムの中核的概念をはじめとして、「批判的ゲーム研究」、「ゲームと社会」、「ゲームデザイン」、「ゲーム・プログラミング」、「ビジュアル・デザイン」、「ゲーム・プロダクション」、「ゲームのビジネス」という 9 つのコアトピックスから構成され、代表的な学位取得プログラムと大卒者の将来像が描かれている。

『IGDA Curriculum Framework』は、教育研究者のみならず、企業内教育においても有益なカリキュラムの構造が提示されている。しかしながら、これは、北米におけるゲーム開発手法に沿って開発されたカリキュラムワークフレームであり、日本におけるゲーム開発手法とは異なる考え方とを比較し、有益な情報は積極的に吸収して、日本型のカリキュラムフレームワークに改訂するとともに、実際のゲーム関連技術教育において活用されることを提案したい。

7.2.6　ゲーム関連開発技術教育とキャリアに関する提案

本調査では、統計的調査やインタビュー調査により、ゲーム関連技術教育の現状と課題を明らかにしてきた。しかしながら、それらの教育とゲーム開発者のキャリアの問題は、表裏一体のものとして捉えられなければならない。とりわけゲーム開発者の平均勤続年数や平均年齢は、他の一般産業と比べると短期・若年型であり、ゲーム開発者として、充実した職業生活を送るためには、教育のみならず、キャリアディベロップメントという観点をも考慮されなければならない。このようなゲーム開発者の働き方を、バウンダリーレス・キャリア（境界のないキャリア）としての新しい働き方として捉えるべきか、それとも人材の未定着を人材マネジメントあるいはキャリアディベロップメントにおける問題として捉えるべきか。ゲーム関連開発技術教育とあわせて、キャリアの問題についても十分に検討を深めていくことを提案したい。

如上の提案は、ゲーム関連技術教育の考え方に関する提案とゲーム関連技術教育の位置付けや枠組みに関する提案が中心となっている。いうまでもなく、ゲーム関連技術教育は、これらとともに、その内容が肝心である。ゲーム技術教育の内容については、前節において現状と課題という形で整理しているので、本節における諸提案と合わせて、ご理解いただきたいと考える。最後に、心ならずも大部になってしまった本調査報告書が、わが国におけるゲーム関連技術教育の発展のために積極的に活用されることを再度期待したい。

参考資料　IGDA カリキュラムフレームワーク日本語訳

IGDA カリキュラムフレームワーク
ゲームとゲーム開発の研究

Version 3.2 beta,

2008 年 2 月リリース

（日本語訳版 2009 年 3 月リリース）

はじめに

IGDA 教育専門部会（**The IGDA Education Special Interest Group, EdSIG**）はこの新しいカリキュラム・フレームのバージョンを作成するために、協力して活動して参りました。たくさんの声を聴き、アイデアを取り入れ、地域毎の作業と共に活動することは決して簡単な努力ではありません。このバージョンのフレームワークは、たくさんのワークショップ、カンファレンスにおけるパネル、そしてディスカッションの成果です。カリキュラム・コミッティーは e-mail を通じて頻繁に連絡を取りつつ一年を通じて何度も修正を重ねて来ました。取り残したことや拾えきれなかった声などが疑うべくもなくあることは承知していますが、ドキュメントを世界中の同輩に公開し、産業界からもレビューを頂くことで、ご指摘を集めて参りました。

IGDA 教育専門部会（EdSIG）は、一つのカリキュラムがすべての学校に適用できるということも、一つの学校の専門学部においてさえ、一つのカリキュラムを適用するべきであるとは考えておりません。どういうことかと言いますと、カリキュラムフレームワークとは、一つのモジュール的なアプローチであって、単一の詳細のカリキュラムではありません。このフレームワークは、一連の教育機関の教材とカリキュラム用の科目に利用できる形で、ゲームを作り学ぶために必要とされる知識領野と実践的なスキルを提供するものなのです。

このフレームワークは、実用的なドキュメントとして、個別のコースの作成から、包括的なプログラムの開発を支援するように、教育者と生徒に役立つようにデザインされています。また、ゲーム専攻のない教育機関においても、学生たちが個人的な学習の道筋を得るためにも役に立ちます。我々はこのフレームワークが、チームワーク、ライティング、プレゼンテーション、そして専門領域を超えた経験に関連した基本的なアイデアを提供するものとなることを望みます。学生はその教育における探求を通じて、これらのソフトスキルに触れるべきであると考えます。こういった技術の習得は、大学院において特に欠けているものであり特別な注意が必要です。

カリキュラム全体の前進に貢献してくれた Tracy Fullerton、ナレッジ・ベースの構築フェーズを率いてくれた Magy Seif-El Nasr、完璧なまでにドキュメントの修正をしてくれた Yusuf Pisan、Wiki を構築し活動をコーディネートするという膨大な作業をしてくれた Darius Kazemi and Darren Torpey に特別に感謝を贈りたいと思います。また、アドバイザリーボードの、Rob Catto, Doug Church, Robin Hunicke, Katherine Isbister, Katie Salen, Warren Spector,Eric Zimmerman にも大きな感謝を贈りたいと思います。彼らの協力なしには、このような大規模で広範囲なプロジェクトを始めることは出来ませんでした。そして、IGDA の疲れを知らぬエグゼクティブ・ディレクター、Jason Della

Rocca にも大きな感謝を送りたいと思います。彼はオープンで透明性のある仕事環境を整備してくれたのみならず、絶えることのない励ましやアドバイス、サポートを与えてくれました。

このフレームワークをガイドとして使用してください、EdSIG listserv ではご質問に答えたり、アドバイスを提供しています。EdSIG wiki は、授業計画のデータベースになっています。こういった授業計画を活用し、またご自身が開発したコースをコミュニティで共有するために投稿してください。これらのドキュメントは、クリエティブコモンズのもとに自由に用いることが出来ます。

Susan Gold
Chairperson, IGDA Education SIG

□訳注：カリキュラムフレームワークの日本語訳は、新清士（IGDA日本）、三宅陽一郎（フロム・ソフトウェア）、尾形美幸（CG-ARTS協会）が担当しています。1～3を新が、3～8を三宅が、APPENDIX A を尾形が行いました。

また、日本語訳も、クリエティブコモンズライセンスに準拠します。

目次

連絡先情報:

International Game Developers Association
19 Mantua Road
Mt. Royal, New Jersey 08061
USA
www.igda.org
igda.org／education

Phone： 856 423 2990
Fax： 856 423 3420
Email： contact@igda.org

国際ゲーム開発者協会日本（IGDA日本）
千葉県市川市新田1-25-16
www.igda.jp

電話番号: 080-3313-4423
Fax：047-322-4776
Email: info@igda.jp

1. イントロダクション

40年前には珍しい存在であったデジタルゲームは、今では、エンターテインメントの最もポピュラーな形式の一つとなり、グローバルな文化として広く普及した要素になっています。デジタルゲームが偏在化と成長したことによって、我々はゲームが単なる商業的な製品として理解するだけではなく、様々な視点から評価する必要性に迫られています。ゲームは美学的な対象であり、コンテクストを学べ、技術的な構造物であり、文化的な現象であるなど、他にも様々な側面を持っています。

ゲームの潜在的な可能性にたどり着くように、ゲームとゲームの研究のために、産業とアカデミックは、ゲームが持つ完全な潜在性にたどり着くために、ゲームを駆動しているアイデア、ゲームによって提供される経験、そして、この若いメディアの社会的と文化的な重要性についてアイデアと経験の密接な関係の深い理解をすることを育んでいかなければなりません。この種の進歩は、アカデミックと産業が共同作業をしたときのみもたらされるのです。

協力はすでに始まっています。開発者は、リスクが増加し開発コストが飛躍的に跳ね上がったことでせき立てられるように、コンセプト的にも技術的にもインスピレーションを求めて研究者と頻繁に会うようになっています。同じように、アカデミックも、ゲームの文化的な重要性と技術的な挑戦を認識しはじめています。アカデミックは開発者との対話を通して調査研究を豊かにしつつあります。複数の大学でゲームの科学的で学術的な研究のためのプログラムの開発をはじめる際、多くの機関で産業の声をカリキュラム作る上ために参考にするようにもなってきています。

国際ゲーム開発者協会（IGDA）のミッションは、**メンバーが同朋につながっていくことでゲーム開発者のキャリアを進歩させ生活を充実させる、プロフェッショナルな開発を促進し、開発者コミュニティに影響を及ぼす問題について主張する**というものです。

IGDAの専門部会（SIG）の一つである、IGDA教育委員会（IGDA Education Committee）のゴールは、**開発者と教育者の間のインタラクションを促すことを助けること、教育プログラムの開発を円滑にすること、そして、ゲームの進化に貢献すること**です。産業とアカデミックのインタラクションには多くのメリットがあります。新しい技術を研究室から製品へと移していく際に円滑になること。産業での経験を教室に持ち込む教育がより豊かになること。ゲームの創造者間でのより批評的なアプローチを生じさせること。同時代のメディア文化の理解をより高めること。アカデミックとゲーム開発者の間の深い交流を促進させることといったことです。

1.1　このレポートの目的

我々が発表するこのドキュメント「カリキュラムフレームワーク」は、ゲームに関連する教育プログラムのためのコンセプト的なガイドです。

科学的なゲーム調査やゲーム研究の分野は若いにもかかわらず、ゲームに関連する教育機関の数と多様性はすでに広大です。単一のカリキュラムをそれらすべてに使うことは出来ません。それゆえ、このドキュメントはモジュール的なカリキュラムフレームワークとして、単一の詳細なカリキュラムでなく発表します。我々はゲームを開発したり研究したりするために求められる知識の領域と実践的なスキルについて述べています。そして、世界中の教育機関に提供するリソースとカリキュラムとして、利用可能なフォーマットにしています。

我々は特定のコースや、妥当な履修時間、特定の学位プログラムの必要性を提案してはいません。また、このフレームワークは開発者が重要とすべき知識領域について伝える試みでもありません。代わりに、このフレームワークはコアトピックの組み合わせを提案します。これはゲームに関連するカリキュラムを構築する上で関係する一般的な領域のリストです。我々は、あなたがコアトピックスから、あなたの必要性に応じて、適合すると思えるものを含めたりはずしたりする形で、組み合わせて対応させることを意図しています。必要性の構成要素のメニューというより、このドキュメントは、あなたのプログラムを成長させ、焦点を絞るための可能性をリストにしているのです。

このような複雑な分野では、どこにも「特効薬」となるアプローチはありません。
我々の望みは、個々の教員や管理者、学生が特有な教育の必要性や教育機関のコンテクストでこのフレームワークを適切な面を適応してほしいということです。

我々はこのレポートの中で、多様なゲームプログラムの性質を説明し、広くて多様な顧客に報いるためにこのレポートを作成しました。
我々は次のような方の助けになると考えています。

- 教育機関でゲームに関連する教育プログラムを開発しようとしている教育者や管理者
- 仕事を探している新しい卒業生に期待できることを、知ろうとしている、もしくは影響を増すことを望んでいるゲーム会社
- ゲーム分野で自身に合う興味や目標、スキルを決めようとしている学生
- 急速に変化する分野の中で教育をどのように続ければよいかを考えているプロフェッショナル

- 公的な教育組織、政府当局、認定組織、などゲーム分野の深い理解を求めている誰でも

  そして、
- 今、利用可能なゲームに関係する教育プログラムを広い範囲で意味を見いだそうとしている誰でも

1.2　このレポートの範囲

ゲームに関連する教育プログラムは多様なものが存在します。ゲームデザインやゲーム・プログラミングというように、わかりやすい名前があるプログラムがある一方で、広いプログラムの中で特殊化されたものとして組み込まれているものもあります。過去10年以上にわたり、ゲームに関連する教育プログラムの数もタイプも劇的に増加しています。それらを目録化して分類化することは、我々の目標や能力を越えていることです。カリキュラムフレームワークの構成要素を通じて　我々はゲーム教育の現在の外観を述べようと試みています。セクション3のコアトピックのリストは、ゲーム教育に関係する主要な領域のすべてを捉えようとしています。一方で、すべてのコアトピックをカバーしているどんな教育プログラムも、教育機関もあるとは思っていません。コアトピックはゲーム教育の違った領域について考えるためのフレームワークを明快に提供するものです。

教育機関としての認定の意味と価値は、違った文脈と違った国では、相当異なっています。現在では、アメリカ合衆国では、ゲームの学部の学生のための認定制度はありません。いくつかの州では、コミュニティカレッジや民間の訓練機関では承認と認可を行っているにもかかわらずです。イギリスでは、Skillset（http://www.skillset.org）が、オーディオ・ビジュアル産業への連絡窓口として業界組織として責任を負っており、現在までに4つのゲームコースの認定を行っています。

ゲームの研究は高度に学際的な分野であるため、どのようなタイプの認可制度が最も適切でメリットがあるのかが明快ではありません。コンピュータ・サイエンス学部によりデザインされ提供されているゲームの学位は、フィルムスクールによってデザインされ提供されているゲームの学位と相当違っているようにみえますが、どちらのプログラムの卒業生もゲーム産業にとって価値のある貢献者になっています。我々は、認定制度を作るには早すぎるかどうか、もしくは、ゲームそれ自身は独特な学問分野ではないのかどうかという議論は脇に置いておいて、分野の外観を述べようと試みています。違った国や組織がゲームプログラムの認定をはじめようとしたとき、我々はリファレンスのためにこのドキュメントにそれに見合う形で追記するつもりです。

1.3 バックグラウンドと歴史

IGDAのゲーム教育についての専門部会（SIG）は、ゲーム産業とアカデミック間の前例のない協力的な努力によって、2000年に組織化されました。そのときには、わずかな研究に値する文化的経済的な力を通じて、ゲームを洗練されたメディア表現として見ていた先駆的な教育者と、ゲームに魅了されていた増加しつつある学生のみでした。同じように、価値のある研究プログラムを勢いよく進めたり、共通言語を作り出したり、ゲームを議論するために共有する知識の土台を構築するといった、アカデミズムと関係性を構築することに価値を見いだしていたのは、手で数えられるだけのゲーム開発者だけでした。

2000年、教育委員会がゲーム産業とアカデミズムのコラボレーションとコミュニケーションを推し進めるために創設されました。IGDAが掲げる目標により推進されることで、委員会は様々な分野のゲーム開発者とアカデミズムの間に橋を架けることをはじめました。

2003年、教育委員会は「IGDA Framework： The Study of Games and Game Development version 2.3 beta1」という名前でカリキュラムフレームワークの最初のバージョンをまとめ上げました。このドキュメントは最終的な成果物することを意図しているものではまったくありませんでした。進行させている実践を取り上げ、そのときの状況のスナップショットとして見せるためのものでした。

2003年以来、よりたくさんの大学や民間の機関がゲームコースの提供をはじめました。コンピュータゲーム産業は成長を続けており、すべての兆候が、今後も成長が続くであろうことを示しています。そして、ゲーム産業とアカデミック間の強い関係性を持つことは、より切実になることでもあります。

この「カリキュラムフレームワーク2008」は、「カリキュラムフレームワーク2003」を土台にしています。このドキュメントをまとめ上げる上で、我々は二つのフェーズの活動を行いました。フェーズ1では、我々はIGDA wiki (http://igda.org/wiki/index.php/Category:Courses)でカリキュラムコースのアウトラインを集めました。そして、教育者がリファレンスとして使うためにナレッジ・ベースの開発をしました。我々はコースを分類して、容易にブラウジングできるようにナレッジ・ベースを構成しました。ナレッジ・ベースは成長を続けており、すでに情報源としての価値を証明しています（注2）

フェーズ2では、我々はナレッジ・ベースから得られた新しい洞察やゲーム開発者からの協力的なフィードバックによって「カリキュラムフレームワーク2003」の改訂を行いました。IGDAのウェブサイトでは、ドラフトのドキュメントの作成をする過程で、より広いコミュニティからの加えられたコメントを見ることができます。それらは、メーリン

グリストからの投稿やカンファレンスでのワークショップやパネルを通じて得られたものです。

（注1）「IGDAカリキュラムフレームワーク2003」に与えられたバージョンナンバーは2.3 betaでした。最終的な成果物ではありませんでした。我々は、このドキュメントは進化を続けていき、常に「Beta」のままであり続けることを期待しています。バージョン1よりも、バージョン2.3がなぜ残っているのかは謎のままです。一つの仮説は、ドキュメントはリリースされた年を反映しているのではないか、バージョン2.003として表記されていて、そしてそれが短く2.3とされてリリースされたというものです。現在のドキュメントは、Game Developers Conference (GDC)の発表された年を反映するとすれば、バージョン2.008 betaと表記されます。

（注2）IGDA Curriculum Knowledge Baseは、
http://igda.org/wiki/index.php/Curriculum_Knowledge_Base にあります。このドキュメントは、コミュニティの努力の結果であり、更新作業が続いています。

1.4 指針とする原則

カリキュラムフレームワークを開発する指針としての原則は、次のようになります。

1. アイデンティティを共有する：ゲームに関係する教育プログラムの数やそれらの全体的なインパクトの劇的な成長によりゲーム教育の共有されたアイデンティティをはっきりと述べておく必要性が生まれています。ゲームは社会に対して十分なインパクトを持っています。そのインパクトはエンターテインメントに限られることなく、ゲームはしばしば、教育、訓練、リクルート、広告、シミュレーション、意志決定など、たくさんの様々な分野で使われています。

ゲームは今日の文化の一部を代表する重要なものです。ゲームの重要性が社会にあるということは、教育者には我々が何をしているのかを社会への理解を促す責任があるということでもあります。

2. 研究分野：「カリキュラムフレームワーク」は、特別なプログラムを決定づけるものではありません。しかし、ゲーム教育を作り上げるための研究分野をアウトライン化したものです。我々は現在のゲーム産業に貢献するためなのはもちろん、次世代のゲームを制作する上で、貢献する求められる知識を作りだし、定義し、調べていくことを行っています。我々のゴールは専門分野の壁を越え、広い観客に到達できるような有用性の高い図像を描くことです。

3. コラボレーションと言葉の共有：ゲーム産業とアカデミックのプログラムの進化は続いているなかで、「カリキュラムフレームワーク」は産業とアカデミズムの両方がお互いに学び合ったことを反映し、開発者とアカデミズム、学生が分野の理解を共有するためにお互いが話すための共通言語を確立していく進化もまた行わなければなりません。アカデミックのプログラムは、適切なスキルを持った学生を教育し、ゲームデザインと開発の新しい分野に使用できる研究を通じて産業を支援することができます。産業で確立された最高の実践は、教育プログラムの中にフィードバックされることで、教育を拡充していくことができるでしょう。

4. 理論と実践：カリキュラムフレームワークは、開発への批評的なアプローチを促す、理論と実践をどのように統合すればよいのかというガイドラインを作成する助けとなれます。これは反対に、社会科学や人文科学のゲームの理論的な研究により、ゲームの開発とビジネスの性質もまた、より注意を払っておかなければならないということでもあります。

5. ライブ性のあるドキュメント：「カリキュラムフレームワーク」は、ゲームデザイン教育において、将来の仕事や、イノベーションを促し、自由に考えるための論点を上げています。これは、最終的なドラフトではなく、ライブ性のあるドキュメントです。これを読んで、いい点であれ悪い点であれ内容についてコメントをする人が多くなればなるほど、このドキュメントは将来の改訂のときにより強力なものになるでしょう。

IGDA教育SIGのウェブページ（http://igda.org/education/）とIGDA wiki（http://igda.org/wiki/index.php/Game_Education_SIG）を通じて、メーリングリストや、ウェブサイト、書籍などより多くの情報と、また、ゲームコースのアウトラインやサンプルの学位プログラムについての幅広い収集を提供しています。

2　ゲームの学問分野（ディシプリン）

研究分野からゲームを定義することは、極めて難しいことです。ゲームはユニークで文化的なアイデンティティを持ち、独特な理論的でコンセプト的な原則を使っており、多様な要素を理解し、真価を認めるためには学際的な奥行きを必要とします。現在、学会誌や論文誌では、学問分野を特別な関心を持って定義しようすることが様々ありますが、分野をすべて明瞭に決めてしまうことにはしばしば議論が伴います。

カリキュラムフレームワークは、ゲームとして参照される分野のなかのコンセプトの幅、知識の詳細、一般的な研究からの推薦を通じて、我々が集めた知識の中で、何が共通分母だと考えられるのかを区別しようと試みています。

一方で、我々が着目しているデジタルゲームは、非電源ゲームや「遊び」の拡張であ

ると認識しています。ゲームの研究は、制作のすべて（マネジメント、デザイン、プログラミング、オーディオ、グラフィックスデザイン、ライティング、テスティング、QA）の段階を組み込まなければなりませんし、ビデオゲームの文化の文脈（マーケティング、社会学、理論、批評）も提供しなければなりません。このドキュメントはゲーム分野を形作っている範囲をアウトライン化しているのです。

それらの分野のすべてを統合する一つの教育プログラムは実現不可能であるため、我々はそれぞれの教育機関がそれぞれの独特なコンテクストを土台にして教育プログラムの深さと幅のバランスを取ることを期待しいています。

2.1 ゲームとは何か

このドキュメントではゲームには広大な可能な方向性があり、一つの学問分野や視点を優先することには、どんな一つの定義であれ限界があると考えています。大半の定義で述べられている一般的なこととしては、ゲームはシステムの状態を変化させる選択を行い、結果を導くプレイヤーが巻き込まれているシステムであるとされています。

ただ、「十分によい」機能する定義を持つ目的のために、我々は以下の定義を提案します。

*ゲーム*は、**ルール**を伴った**行動**である。それは**遊び**の形式であり、ゲームシステムそれ自体が伴う、もしくは、ランダム性／運命／運に関連する、しばしば、しかし常にではない、他のプレイヤーとの明快な、**対立**に関係している。

大半のゲームは**ゴール**を持っている、しかし、すべてではない（例、[「ザ・シムズ」、「シムシティ」）。大半のゲームは**開始と終わりのポイントが定義されている**、しかし、すべてではない（例、World of WarCraft, Dungeons & Dragons）。大半のゲームはプレイヤーの役割として**意志決定**が含まれている、しかし、すべてではない（例、Candy land、すごろく）。

*ビデオゲーム*は、何らかの種類や何らかの方法で、**デジタルなビデオスクリーン**を使った（上位概念としての）**ゲーム**である。

上記の定義はゲームのタイプを絞り込むことを意味していませんが、「機能する定義」として扱っています。読者にはゲームを構成する付加的な定義と奥行きについてのリファレンスの材料を参照して頂きたいです。

ゲームの研究は、この複雑なシステムの働きに影響し合う多く要素を理解するという内容が含まれています。ゲーム研究の三つの重なった領域を上げます。

- **ゲームデザイン**－インタラクションとインターフェース・デザインを第一に考慮されるゲームデザインは、プレイヤーの行動がゲーム環境の文脈を意味するようになる、遊びのシステムを作り出す過程である。[Salen and Zimmerman, Rules of Play,2004] ゲームデザインの高品質な製品をどのように作り出すのかという定義や、コンセプト、実践の組み合わせを包括しています。ゲームデザインプロセスでは暗黙の了解として、人間がプレイ可能なインタラクティブ環境でのゲームの実行が認められるようにトレードオフがデザインとして考えられています。

- **ゲーム開発**－ゲームの制作、特にゲーム製作で使われるテクノロジーが第一に考慮される
  ゲーム開発のソフトウェア・エンジニアリングのような技術的な学問分野や、プレイ可能な現実世界のフォーマットの中にあるゲームデザインを持ち込むためにアートや音楽といった創造的な学問分野との学際的な協力が包括されている過程です。[Rabin, Introduction to Game Development, 2005] ゲーム開発は、しばしば事前に成功するか失敗するか知ることなく潜在的なゲームの要素を開発を進め、付加的にテストを行うという場合がある。また、ゲーム開発は、使用可能なリソースと受け入れられる時間的な制約の中で、ゲームを完成させることを確実にするため、プロジェクト・マネジメントの知識を必要とする。

- **ゲーム研究**－文化的な人工物として、メディアの一部として、遊びの理論を探索するものとして、ゲームを調査することが第一に考慮される。
  ゲーム研究はゲームの研究と分析のために概念的な基礎と語彙を扱っています。ゲームの顧客に関係することや、ゲームの歴史やビデオゲームの歴史、技術／プラットホームの歴史、ゲームの批評、教育やそのプロセスのためのゲームといったものです。

上記の定義は、ドキュメントを読むためのガイドラインとすることを意図しているものです。これらが最終的で、決定的で、全世界的に受け入れられたということを意味していません。それぞれの領域では、複数の定義が争っているのを見つけることができますし、もちろん、ゲームを区分して研究するには非常に様々な方法があります。一方で、次のセクションで述べるそれぞれのコアトピックスは、複数の領域にまたがっています。ただ、概念的には、コアトピックは通常は一つの領域にとして考えた方が有用です。

3 コアトピックス

ゲームは学際的な多くのレベルで構成されています。ゲームを創るためには、オーディオとビジュアル・デザインから、プログラミングからプロジェクト・マネジメントといった、異なった存在している分野間でのコラボレーションを必要とします。同時に、デジ

タルゲームは、ゲームデザインとインタラクティブ・ストーリーテリングのような、混成した新しい種類のディシプリンの登場を引き起こしています。ゲームを文化的な人工物として考えるとき、ゲームを完全に批評的に理解するには、我々は、社会的、心理学的、歴史的、美学的な複合体であるすべてを、ゲームとして正しく理解する必要性があります。

この理由のために、我々は　ゲームに関連する教育への分野をまたいだディシプリンによるアプローチを強く推奨しています。我々にとって、これは確立されている分野がゲームに持ち込まれるということと、ゲームが可能にする研究の新しい分野への注目を得るという二つの側面を持つ教育的なアプローチであることを意味しています。

ゲーム産業からの批判の一つには、ゲームコースの卒業生たちのなかには、ゲームデザインと開発のそれぞれの部分の断片的なわずかな情報しか知らない、しかし、特定の分野のどんな深い知識も持っていない点があります。それぞれの教育機関では、注意深く、対象とする学生や特別な環境に基づいて、教育機関が提案する教育プログラムの幅と広さのバランスを取る必要があります。

以下のコアトピックを我々が規定する目的は、このアプローチを反映したものです。コアトピックのいくつかはコンピュータ・サイエンスようにすでにあるディシプリンから直接持ってきたものです。他のものには、ディシプリンを組み合わせたものや、新しいものとして作り出したものがあります。我々は、それらの中には、知識分野が重なっている形を違ったあり方で扱ったものもあることを知っています。しかし、以下にリスト化されたコアトピックの規定によって、直感的にユニークな実践的で理論的なゲームの項目を示していると我々は感じています。つまり、コアトピックはゲームに関係する教育の広大な俯瞰的な視点を提供しているのです。

トピックス

1. 批評的なゲーム研究
2. ゲームと社会
3. ゲームデザイン
4. ゲーム・プログラミング
5. ビジュアル・デザイン
6. オーディオ・デザイン
7. インタラクティブ・ストーリーテリング
8. ゲーム・プロダクション
9. ゲームのビジネス

指摘しておかなければならないのは、このリストのなかには両立する重なり合う部分があることです。それは、コアトピック以上にサブトピックの部分に見ることができるかもしれません。これがまた意味していることは、異なったディシプリンから持ち込むことができる項目がいくつかあるということです（例、プレイテスト。デザインの部分としてのプレイテストと、ソフトウェア開発のプレイテストと、マーケティングでのフォーカステストの一部としてのプレイテスト）

また同時に、すべてのゲーム教育のプログラムに盛り込む必要がある、コアトピックを選択した組み合わせに、何らかの同意があるということではありません。アート重視のゲームコース（もしくは、プログラミング重視、デザイン重視、もしくはビジネス重視）に学んでいる学生でさえ、把握しなければならないことに同意があるわけではありません。事実、ゲームプログラマーと3Dゲームアーティストのための重要なトピックのうちいくつかは重なり合っているだろうことを、また、そこには多くの違いがあるだろうということを予期しています。コアトピックの組み合わせがはっきりとしてきたら、我々は変更を反映して、このドキュメントをアップデートするつもりです。

それぞれのコアトピックのために、我々は関連する IGDA のリソースへのリンクを可能なときに提供しています。しかし、それらのリソースは、長期間にわたって、成長し変化するため、関心のある読者は、追加的なリソースを探すために直接 IGDA http://igda.org/ のページを調べることを勧めます。

以降は、トピックのそれぞれについての、一般的なディシプリンです。

3.1 批判的ゲーム研究

電子ゲームと非電子ゲームの批評と分析と歴史

学際的なコアトピックは、歴史、文学、メディア研究、デザインからのアプローチを統合したものです。*批評的ゲーム研究*によって、ゲームの美学をはっきりと述べるために、批評的な語彙を開発し磨きをかけています。これには、ゲームにとって固有で独特な機能や、他のメディアや文化と重なり合っているものの両方が含まれています。例えば、批評的ゲーム研究は、ゲームプレイのテキストの分析から得られる洞察や、文学、映画、テレビ、演劇、インタラクティブアートといった、他のメディアの確立された仕事から、豊富な批評のフレームワークを提供することができます。また、次のようなものを含んでいます。コンピュータやデジタルゲーム、オモチャの歴史。重要で影響のあるゲームの基準の構築と批評。そして、ゲーム批評とゲーム・ジャーナリズム。

3.1.1 ゲーム批評

ゲーム研究

- ルドロジー－ゲームと遊びの活動を研究すること
- 批評的な理論と調査
- ゲームと遊びを議論するための批評的な語彙。ゲームマシンの進化や、ゲームプレイ、ゲームのフロー、ゲームデザイン、ゲームデザインに影響するゲームプレイの経験の形式といったものが含まれる。
- 影響力のある、もしくは、重要なゲームの基準の確立と批判

経験中心の批判（プレイヤーを中心にしたアプローチ）

- インタラクティブ性の研究、人間とのインタラクションに関連するテクノロジー
- 仮想空間での探索の機能と使い方
- プレイヤーの“代理”を促しサポートする
- プレイヤーの没入を作り出し維持する
- 信じられないと判断することを食い止めるサポートをする
- 人間の仮想の社会的なインタラクションの研究

消費者志向の批評

- ゲームプレス（雑誌等）の機能と現在の状況を分析し理解すること
- ゲームレビューの機能と現在の状況
- 活字やメディアジャーナリズムのツールやテクニック、標準
- 立法、司法のゲーム産業へのインパクト
- ゲームの宣伝

ジャンル分析

- 何のジャンルが存在しているのか？
- どのようにしてゲームジャンルを定義するのか？
- ゲームジャンルの歴史（登場して、なくなってしまったジャンル）
- ジャンル分けは使いやすい？　ゲームに適応しているジャンルが他のメディアでは違ったものになっているとき、ジャンル分析の適応はどのように異なっているのか

作者分析

- ゲーム開発がコラボレーション性を持たされている際、誰が実際にゲームを創りだしているのか？
- 作家性のコンセプトはそれぞれのゲームに適応できるだろうか？

- 作家性のコンセプトは全体の個々の部品に適応できるだろうか？
- 一人の作家の働きによるゲームの「ブランディング」

ゲームデザインの分析

- ゲームプレイ
- ナラティブ／ゲームライティング
- ストーリーとプロット
- キャラクター開発
- アートデザイン
- サウンドデザイン
- インタラクションデザイン（どのように新しいインタラクションはプレイの形式に影響を与えているか？）
- シミュレーションの手法

3.1.2. メディア研究

非ゲームメディア、文学、ラジオ、映画、テレビ、アート、演劇、マンガ、建築、インターネット。

メディア研究手法

- データ収集手法
- 民族誌学（エスノグラフィー）
  - 質的
  - 量的
- テクノロジー調査（違ったテクノロジーの研究と比較、それらのパフォーマンスと、潜在力）
- 実験的なテクノロジー（新しいゲームテクノロジーを確立する、特にハードウェア）
- マスメディアやポップカルチャーの研究への紹介
- 一般的なメディアの影響の調査
- ゲームに特化した調査
- プレイヤーにフォーカスした調査

コアとなる経験

- ゲームのレビューを書く
- ゲームの批評を読む
- ゲームの批評を書く

3.2. ゲームと社会

どのようにゲームが個々人やグループに影響し、構築されているのかを理解することは、どのようにゲームが、個々人やグループによって影響され構築されているのかを理解するのと同じことでもあります。

このコアトピックでは、社会学、人類学、文化研究、心理学により全世界のゲーム文化への重要な洞察を扱います。*ゲームと社会*はオンラインの経済やコミュニティ構築、ファンカルチャー、ゲームコンテンツの創造的な改変、人間の文化のなかでの遊びの役割、そして、オンラインやオフラインのアイデンティティの関係性といった学術的な研究も含みます。また、ゲームに関係する表現や、イデオロギー、レトリックも、このトピックで見いだせます。このコアトピックでは、メディアの影響の研究や個々人やグループの心理学的なインパクトについて継続されている議論といったものを含んでおり、ゲームの心理的な側面も範囲にしています。

また、このコアトピックでは、個々人やグループがゲームをどのように構築しているのかを調べます。どのように価値観や、アイデンティティ、文化的なイメージがゲームの制作を形作っているのかを探索します。最後に、このコアトピックでは、どのようにテクノロジーや、法的機関、政府の政治、企業が、ビデオゲームの制作を形作っているのかを探索します。国境線のなかにゲームが位置づけられながら、社会的、政治的、経済的なコンテクストを通じて、ゲームがどのように発展してきたのかについての洞察を得ることができます。

プレイヤーと影響

ゲームの人口統計

- プレイヤーのジェンダーと多様性
- 子供時代、教育と子供の成長
- 小売店とプレイヤーの好みとパターンの理解
- 情報のソースと、ゲームに関連する組織

ゲームの「文化」

- ポップカルチャー：偶像としてのゲームと文化的な人工物としてのゲーム
- ファンカルチャー：ゲームコミュニティとそのメンバー
    - なぜコミュニティは形成されるか
    - ファンコミュニティの創造がどのように促され、それが、維持されるのか
    - ゲームのマーチャンダイジング
    - 関連するメディアからのファンコミュニティ

- オンラインコミュニティ：デザインと力学
- 大衆文化：ゲームについての文化的な対話
  - 他のメディア（映画、テレビ、本など）のなかのゲーム
  - コンピュータに文化的なインパクトを与えた広い視点でのゲーム

歴史

- 分野を定義してきた有名なデザイナーや、人々や、出来事
- 電子ゲーム／非電子ゲーム／オンラインゲーム
- コンピュータ／プラットホーム研究
- デジタル技術の保存
- 他の国からのゲーム

プレイ（遊び）の経験

プレイ（遊び）の経験の歴史的側面

- 遊びの歴史
- 遊びの比較文化人類学
- 国境を越えたゲームの共通性と違い
- 遊びの歴史での経済の役割（レジャーの時間、オモチャのために時間を使う）

社会的側面

- ソーシャルゲーム、オンライン大規模マルチプレイヤーゲーム
- どのようにゲームに、実験的な遊びのために「安全な空間」を創るのか
- どのように社会的な設定を使うか
- どのように伝統的な社会の役割をサポートし、壊すか
- チート行為の影響（ゲームの間と実習／学習の間との違い、チート行為を確立して使用する）
- ゲーム内のステレオタイプ（キャラクター、設定）
- ゲーム内の倫理的で社会的な問題

心理学的側面

- ゲームにより、どのように感情的な反応が引き起こされ、操作されるか
- 認知理論
  - 心理モデル
  - 問題解決
- 知能の原理
- 開発モデルの適応

- 他の者によるゲームへの反応（コミックやロック音楽、政治的な立法行為、裁判）
- ゲームが、どのように私たち自身や他者による理解に依存し影響を受けているか
- ゲームと暴力の関係についての研究
- ゲームと中毒についての研究

経済的側面

- より大きな販売への強い求めー成功した製品の続編や、ライセンスされた製品の追求
- ゲームの質の役割と、80年代の供給によるクラッシュ
- 人口統計の変化、新しい機会

人間と機械とのインタラクション

- ユーザービリティの問題（例．ゲームのインターフェースは、学びやすく、使いやすくする必要がある）
- アクセサビリティの問題（例．特別なニーズを持つユーザーのための対応）

ゲームとゲームテクノロジーの構築

ゲーム産業を構成するテクノロジーと慣習の歴史的側面

- ゲームテクノロジーの歴史
- ゲーム企業の歴史
- ビデオゲームの訴訟と特許の歴史

ゲーム産業の人類学

- ゲーム産業の政治的、経済的コンテキスト
- ゲーム開発の実践
- ゲーム開発の経済的コンテキスト
- ゲーム開発者の「文化」
- ゲーマーの文化とゲーム制作者の文化のインターセクション（関係積）
- ゲームとゲーム技術の国境を越えた制作

3.3. ゲームデザイン

ゲームのルールとプレイの裏側にある原則と方法論

このコアトピックでは、電子的ないし非電子的なゲームのデザインの背景にある基礎となる考え方を示しています。

ゲームデザインには、ゲームプレイ、ストーリーテリング、チャレンジの設定や、基

礎的なインタラクティブデザインといったものが含まれており、さらに、インターフェース・デザイン、情報デザイン、ゲーム世界のインタラクションが含まれています。もしかするとゲームデザインとって最も重要なことは、どのようにゲームが、経験を構築するために機能しているのかの詳細な検討ではないかといえます。

その機能には、ルールデザイン、プレイメカニクス、ゲームバランスの調整、ソーシャルゲームのインタラクションといったものや、ビジュアル、オーディオや文字の要素といった、全体的なゲームの経験へと統合されていくものが含まれています。さらに、ゲームデザインの実践的な側面には、ゲームデザインのゲームデザインドキュメントやプレイテストのようなものも対象となります。

ゲームデザインは、コアトピックの中でも、ゲームの最も固有の要素が集まっているものともいえ、そのため、我々がここで示すアウトラインはカリキュラムフレームワークの心臓といえるような部分があります。他方で、最近理解されるようになったものであるために、訓練された教員や質の高い参照できるリファレンスは痛ましいほどに不足しています。そのため、コアトピックの中でも教育に使うにはチャレンジングなものになっています。

コンセプト的ゲームデザイン

ゲームの元素パーツの理解

- ゲームの使用されるオブジェクト（トークン）とゲームセッティング
- ルール
- 原動力
- 遊びのメカニクス
- ゴール
- 対立
- テーマ／色

遊びのメカニクス

- ゲームの「ルール」とは何か？
  - ルールはどのように構成されるべきか？
  - 障害と救済、ペナルティと報酬の正しいバランスをどのようにすれば創れるか？
  - 「世界」の性質とインタラクション
- 核となるメカニクス：それらは何か、それらはどのようにゲームプレイを形作るか
  - 遊びのメカニクスのタイプ：個別の／連続のインプット、決められている／ランダムな結果、など

- システムデザインの鍵となる成分としての情報の流れ
  - プレイヤーからの入力
  - システムのアウトプット
  - インフォメーションのフィードバックループ
  - フィードバックループが強固な情報として持続される重要性
- ゲーム理論：二人のプレイヤー用ゲームと戦略、利得行列、ナッシュ均衡
- どのようにゲームメカニクスが、ゲームジャンルやプラットホームによって形作られ影響を受けているか？
  - どのようなときに、ゲームはあまりに難しすぎる、もしくは、あまりに簡単すぎるか？　そして、それはなぜか？
  - 何がゲームの因果関係をあまりに難しくするのか、もしくは簡単にするのか？
- プレイメカニクスのどのような性質が、人々のどのような性質に最も機能するだろうか？
- 競争的、協力的な環境での戦略的な意志決定の研究（囚人のジレンマなど）
- ゲームデザインのなかでのバランスの役割
  - 状況ごとにバランスを取るためのテクニック（エリア効果 対 ポイント効果、防御装備 対 攻撃装備）
  - 均等性バランス（一秒当たりのダメージ、正確性□対□パワーなど）
- 推移的なメカニクス 対 自動的なメカニクス
- モデリング手法

ゲームデザインに対してのアプローチ

- アルゴリズム的なゲームデザイン思考
  - ボトムアップのデザイン対トップダウンのデザイン
- プレイヤー経験のアプローチー瞬間の時間デザイン
- 世界デザインーストーリーと設定に含まれるゲームプレイの構築
- どのようにゲームメカニクスが、ゲームジャンルやプラットホームによって形作られ影響を受けているか？
- プレイメカニクスのどのような性質が、人々のどのような性質に最も機能するだろうか？

ボードゲームとロールプレイングのデザイン

- ウォーゲーム
- ロールプレイングゲーム
- トレーディングカードゲーム（Collectable Card games）

- チャンスと確率の役割
- 物語とその雰囲気 対 メカニクス

アイデア

- 新しいアイデアを創出する
  - 個人やグループのブレインストーミング
  - まわりの世界のシステムを観察すること
- アイデアをゲームコンセプトに変えていくこと
- デザインドキュメントを使ったゲームコンセプトとゲームのプロトタイプの評価

おもしろさ

- 「おもしろさ」とは何を意味しているのか？
- 様々な種類の「おもしろさ」：探検、キャラクターの出世／成長、社会的経験、挑戦など
- ゲームは「おもしろい」ものでなければならないのか？
- なぜ人は遊ぶのか

抽象的なゲーム要素

- ポジティブとネガティブを持つフィードバックシステム
  - ゲームバランスを取るためのツール
  - プレイヤーへの報酬と罰
  - 挑戦と「フロー」
- 創発的な複雑さ
  - プレイヤーの経験をユニークなものに導くシステムのインタラクション
  - ゲームを破綻させないように創発的な複雑さをコントロールする
  - プレイヤーの意図と、プレイヤーが十分に理解でき、予測でき、コントロールできるような明快なシステムを作ること
- シミュレーションと模倣
  - 柔軟な反応を許すシステムを使うこと 対 前もって決められていた状況への決められた行動
- コミュニケーションシステム
  - プレイヤーが必要な情報はどれぐらいだろうか？
  - プレイヤーが情報を得るための最善の方法は何だろうか？
  - レイヤー化したコミュニケーション
  - 潜在意識的なコミュニケーション

心理学的なデザインの配慮

- オペラント条件づけ
- フロー状態
- ゲーム中毒
- 報酬とペナルティ
- 難易度曲線
- 多様性のある社会システムを作り出す
- ゲームの中にプレイヤーを居続けさせる／そのうちプレイヤーをゲームから連れ戻す
- ゲームプレイのスタイルの多様性を育てる

インターフェース・デザイン

- インターフェース・デザイン理論／コンピュータUI理論
- 人とコンピュータ間のインタラクション
  - 奇抜な、もしくは、特化したインターフェース
- インフォメーションの視覚化
- ユーザータスクモデリング
- プレイヤーコントロールスキームのバランスかー単純性 対 表現力
- 特別なハードウェアの組み合わせによるインパクトーコントローラー、キーボード、ヘッドセットなど

ゲームデザインの反復的性質：創造、テスト、変更、その繰り返し

シリアスゲームデザイン

医療、訓練、治療などエンターテインメントでないアプリケーションでのゲームの使用
教育
訓練
治療的な使用
シミュレーション
政治的なメッセージのためのゲーム利用
アート的なメディアとしてのゲーム利用
コンテンツの専門家との共同作業
計測機器的なデザイン
評価ー教育やトレーニングツールとしてのゲームの評価

実践的ゲームデザイン

空間デザイン

- ゲームプレイの空間
  - 具象的な空間
  - 抽象的な空間
  - 空間と歩調合わせ
  - 空間と物語
- 密度の高いインタラクティブと、高い反応がある世界を作る
- 空間デザインを通じたゴールへのコミュニケーション

タスクデザイン

- アクションとインタラクション
  - 世界／幾何学のインタラクション
  - キャラクターインタラクション
  - パズル
- プレイヤーへの適切なフィードバックの提供

デザインの統合

- 空間とタスクの混合
- アートとゲームプレイの統合
- プラットホームの選択によって決まってくる要素のデザイン

コントロールのスキーム

- 直接的／間接的操作
- 移動とナビゲーション
- アイテムとアイテムの操作
- アイテムの品目一覧
- コントロールするプレイヤーによる自然なマッピング

カスタムツールの使用

- ゲームデザインのコンセプトをゲームの基礎を成すシステムにしていく

トレーニング

- プレイヤーにゲームの遊び方を教える／ゲームの中で何ができるのか：ゲームの中にチュートリアルを統合する
- 一貫性のある挑戦と適切なフィードバックによって学習をサポートする

- ゲーム世界内の挑戦、行動、能力についてプレイヤーとコミュニケーションを取る
- プレイヤーがゲームで何を行ったのかという経緯を追う／残っているゴールについてのフィードバックを提供する

ゲームチューニング

- 動的なシステムとしてゲームを理解する
- バランスが取れたゲームとは何か
- 実際のゲームプレイからフィードバックを参照しながらゲームをチューニングする戦略を決めていく
- 挑戦の達成具合から、プレイヤーの進展速度のバランスを取る

ゲームプレイヤーの分析

- 顧客は誰であるのかを理解する
- 多様な人口属性のためのデザイン
- 与えられた顧客で成功を計るためどのような基準を使うか
- 品質保証（QA）と共に仕事をする
  - バグトラッキング、バグアサインメント
  - 他の人へのフィードバックの書き方を理解する

プレイテスト（制作の中で頻繁に使われるが、デザインのフェーズでも同じように使うことができる）

- 人的対象のテストの際の倫理的配慮
- 考えていることを声に出す（Think-aloud）方式の手順
- ユーザビリティテスティングとプレイ・テスティングの違いと類似性
- インタビューとアンケート
- 観察
- ベータテスト
- 異なった制約下でのテスト：自分自身でテストする、親しい友人や学校の友達によるテスト、完全な知識のない人によるテストの監督、プロトタイプの情報をまったく提供しないで行うテスト
- ペーパープロトタイピング
- 速く、軽いレベルのコンピュータ上でのプロトタイピング
- ターンベースのビデオゲームのための物理的なプロトタイプの作成
- リアルタイムビデオゲームのための物理的なプロトタイプの作成
- 個別のシステムやメカニクスのデジタルプロトタイプの作成

ゲームデザインのドキュメンテーション

- ゲームデザインのドキュメントの執筆と保守
- コンセプト、提案、ルールドキュメント、デザインドキュメントの執筆
- デザインのアイデアをチームと明快に伝達すること
  - ディティールについての適切なレベル
  - デザインの要求をアーティストとプログラマーが理解可能なものにすること
  - 変更履歴

コンテンツデザイン

- レベルデザイン

3.4 ゲーム・プログラミング

伝統的なコンピュータ・サイエンスとソフトウェア・エンジニアリング、特に、そのゲームの技術的側面について取り上げる

コアトピックとしては、物理、数学、プログラミング技術、アルゴリズム・デザイン、ゲームに特化したプログラミングや、ゲーム・テスティングの技術的側面を取り上げます。

こういった領域の多くは、これまでのコンピュータ・サイエンスやソフトウェア・エンジニアリングのカリキュラムにおいて教えられて来たものです。しかしながら、ゲーム開発は非常に特殊でチャレンジングな一連のプログラミングを要求します。パス検索や、ソーティングなどの主要なアルゴリズムの最適化、リアルタイム 3D レンダリングなどです。

数学と科学的技術

- 基本的なニュートン力学の物理
- 計算力学
- 確率と統計
  - 幾何学、離散数学と線形代数ベクトルと行列
  - 座標空間と変換
  - 衝突判定
- 計算幾何
- 基本的な計算と微分方程式

スタイルとデザイン原理

- コヒーレンシー
- オブジェクト指向プログラミングのパラダイム
- デザインパターン

- ○ ゲームデザインパターン

インフォメーション・デザイン

- データ構造 ー　データ・アーキテクチャ、ファイルフォーマット
- データ組織、データ圧縮
- アセット・パイプライン
- 計算幾何
- 環境モデル、空間データ構造
- データベース
- マシーン・アーキテクチャ
- 最適化（CPU、GPU）
- 組み込みシステム開発
- コンフィグレーション・コントロールとソースコントロール・システム
- ソフトウェア・アーキテクチャ
- ソフトウェア・エンジニアリング

ゲームエンジン・デザイン

- 目的と重要性
- アーキテクチャとデザイン
- データ・パイプライン
- スタンドアローンのゲーミング・アプリケーションを製作するための方法と実践
  - ○ クロスプラットホーム技術の実装の限界
- 3Dエンジンのプログラミングにおける固有の問題、普遍的な問題
  - ○ グラフィック・ライブラリと 3D ハードウェアの問題
  - ○ プログラミング・オブジェクトとカメラモーション
  - ○ 衝突判定と衝突レスポンス
  - ○ パフォーマンス分析
  - ○ 特殊エフェクト

プロトタイピング

- 素早くくり返しができるツールとスキル
- 外部からコンフィグレーションできるフレキシブルなシステムの構築

プログラミング・チーム　-- 構造とワーキング・リレーション・シップ

- 分野を超えた共同チームによる開発
- プログラマー、アーティスト、デザイナー、プロデューサーなど間の議論

- チーム・プログラミング・プロセスと方法論

デザイン・技術の統合

- プレイヤーのゴールとアクションのサポート
- 知的で一貫して整合性のある、リアクティブなゲーム環境の構築
- プラットホームの問題

リアルタイムなゲーム環境のためのシステム・アーキテクチャとシミュレーション

- コンカレント（並行性を持った）プログラミングの技術
- サブシステムの統合（物理、衝突判定、AI，インプット、レンダー、スクリプティング）
- ゲームエンジンにサードパーティのシステムを拡張して組み込むこと
- リソース割り当て（CPU、GPU、メモリ）

コンピュータ・アーキテクチャ

- プログラム・デザインから見たCPUの構造
  - （例. ブランチを避ける）
- プログラム・デザインから見たメモリ階層構造
  - （例. メモリにおけるデータ構造のアライメント、参照の局在性）
- CPUとGPUを使い分けるアルゴリズム・デザイン

ツール製作

- 「ツール開発」
- GUI製作
- マルチメディア・コンテンツ製作、修正とマネジメントのためのツール
- カスタム・デザインツール
- プログラマー以外の開発者の使用に耐えるフレキシブルなシステムの構築

グラフィック・プログラミング

- レンダリング
  - 変換、ライティング、テクスチャリング
  - クリッピング、オクルージョン、透明化
  - レベル・オブ・ディティールの取り入れ
  - レンダリング時間の最適化のためのデータ構造
- アニメーション
  - フォーワード、インバース・キネマティクス

- 変換表現
- 補完テクニック
- カメラ・アニメーション

- グラフィック・システムデザイン
- プロシージャル・コンテンツ・ジェネレーション（テクスチャ、モデルなど）

サウンド／オーディオ プログラミング

- サウンドの物理と人間の聴覚
- 3Dポジショナル・サウンドのプログラミング
- オーディオ・チャンネルの利用
- オーディオ優先付け

人工知能

- ゲームAIと伝統的なAIの目的の違い
- パス検索、探索アルゴリズム
- エージェント・アーキテクチャ
- 意思決定システム
- 状態マシンのデザイン
- 統計的な機械学習

ネットワーク

- ネットワークとサーバーデザイン
- パフォーマンス測定
- トポロジー
- プロトコル － TCP／IP, UDP, …
- セキュリティ
- ゲームサーバー
- ゲーム・プロトコル開発
- 利用可能なネットワーク・ライブラリ
- オープンソース・ネットワークゲームのケーススタディー

ゲームロジック

- コンパイラ
- スクリプト言語

プレイ分析

- プレイヤーのフラストレーション、進行、楽しさをモニターするプレイ・テスティング
- プレイヤーの状態をモニタリング -- ゲームプレイ・データロギング
- プレイヤーの計測

3.5 ビジュアル・デザイン

ゲームにおけるビジュアル要素のデザイン、製作、分析。

本トピックは、コンピュータに直接、関連する、しないに関らず、広いメディア領域における、ビジュアル・デザインの基礎について取り扱う。取り上げる領域としては、「絵画、線描、彫刻と言った伝統的なアート・メディアにおける歴史、分析、製作」、「イラストレーション、タイポグラフィー、グラフィックデザインと言ったコミュニケーション・フィールド」、「建築や工業デザインといった他のデザイン分野」、「アニメーションや映画製作と言った時間を基本とするメディア」です。

特にビジュアルにおける美学がゲームの体験において、果たす役割に重点を置きます。2D、3D グラフィクスの課程の導入は、ビジュアル・デザインのカリキュラムで重要です。しかしながら、特定にソフトウェア・パッケージを習得することよりも、基本的なビジュアル・デザインの原則に重きを置きます。

基本的なビジュアル・デザイン

- アートの歴史と理論
- ビジュアル・デザインの基礎
  - コンポジション
  - 光と色彩
  - グラフィックデザインとタイポグラフィー
- 線描の基本
- ペインティング技術
- 彫刻
- 解剖学と人体描写
- 生理学と運動学

非叙述的グラフィクス／表現ツールとしての抽象

インタラクティブなコンテクストにおけるビジュアル・デザイン

ビジュアルな叙述：絵画、コミック、写真、フィルム

モーショングラフィックス

- アニメーション
- 映画
- カメラアングルとフレーミング
- ビジュアルな叙述／絵コンテ
- 映画制作：フレーミング、ショットのタイプ、カメラ制御、編集
- 物理的運動学

映像素材作成

- 2Dグラフィックス
  - ピクセルアート
- 3Dモデリング
- テクスチャ
- インターフェース・デザイン
- キャラクターデザイン
  - コンセプト設計
  - キャラクターモデリング
  - キャラクターアニメーション

ワールドデザイン

- 環境のモデリング

建築物

- 建築の基本原理
- 建築の歴史
- 現実の空間とゲーム空間の違い
- 空間のデザイン
- ナビゲーション
- 素材

3D ハードウェアの活用

- プロシージャル・シェーディング
- ライティング
- エフェクト

ゲームアート（ゲームコンテンツについてのデジタルなアート）

- カスタムツールの使用－ゲームアートをゲームエンジンへ組み込む

インフォンメーションの可視化

プロシージャル・コンテンツ

3.6 オーディオ・デザイン

サウンドとサウンド環境をデザインし創作する

このトピックの中心は、理論的、または実践的にオーディオに関係する分野である。音楽理論、歴史、作曲、美学的な音楽の分析、レコーディング・スタジオにおけるスキル、電子サウンドの作成である。特にデジタルゲーム技術に関する音楽は、3D サウンド・プロセッシングや生成的オーディオ構造などを含む。しかしながら、ここではゲームというより大きなコンテクストにおける音楽がもたらす体験の役割について特に重点を置く必要がある。ビジュアル・デザインと同様に、特定の技術的知識よりは、デザインの気尾基本に重点を置く。

オーディオの歴史と理論

基本的な技術スキル

基本的なスタジオスキル

- ハードウェアとソフトウェアに精通（例：マイク、ミキサー、アウトボード・ギア）
- レコーディング、ミキシング、マスタリング
- スタジオ編成

オーディオ・プログラミング

オーディオ・アセット

オーディオ・ツール

オーディオ・デザインの基礎

- ムードを決め、テンションや解決を運用する。

- 美しいサウンドのエフェクトを処理し、ミキシングし、コントロールする。
- ゲーム製作の一般的なワークフロー
- オーディオ・エンジンの専門用語と機能

インタラクティブ・オーディオ入門
- 音とインタラクティビティーをデザインする
- サウンド・エフェクト
- 音楽
- ボイス・レコーディング

サウンド・エフェクト
- サウンド環境のシミュレーション
- サウンドトラックにおけるアンビエンスと音楽性の対立

音楽
- 作曲
- インタラクティブ・スコアリング

3D オーディオ
- 3Dのマルチチャンネル・サウンドの基礎
- エフェクトのモデリング、エコー、ルームサイズ・シミュレーション

3.7 インタラクティブ・ストーリーテリング

伝統的なストーリーテリングとインタラクティブな物語への挑戦

インタラクティブな作品のライターとデザイナーには伝統的な物語理論、キャラクターの開発、プロット、会話、物語の背景、そして、世界の創造に対する理解が必要とされる。同様に、ゲームと関連して文学、演劇、映画における実験的なストーリーテリングへアプローチへの理解も必要である。加えて、インタラクティブなストーリーテリングは、新しいツールや技術に精通していることが必要です。それは、新しいメディアに対するライティング、アルゴリム的なストーリーテリング、コラボレーションによるストーリー構成などです。このコアトピックでは、特にこれらのアプローチをゲームにおけるインタラクティブなストーリーテリングに応用する。

非インタラクティブ・メディアにおけるストーリー
- 文学理論と物語学
    - 伝統的な物語の幕構成

- 抽象的に、そして具体的に物語について考えること
- 伝統的な物語（民話）
- 構造主義と物語論
- ポスト構造主義（バルト、ボードリヤール、等）
- ポストモダン文学

- 演劇
  - 演戯理論
  - 理論家（アリストテレス、ブレヒト、アルトー、ボアール等）
- 物語創作
  - 設定：時間、場所
  - キャラクター：アクション、モチベーション、会話
  - イベント
- 会話
  - スタイル
  - 声と視点
  - イベント構造
- 小説、映画、演劇における性格描写
- 映画と文学理論へのイントロダクション
- ゲームとの理論の理論
- コンテキスト・セッティングと伝統的なストーリーテリング
- 物語の背景と架空の設定のデザイン
- 説得力のあるキャラクターの創造

インタラクティブメディアにおける物語

- 理論的課題
  - 代理(agency)と没入(immersion)
  - インタラクティビティーと物語における対立
  - サイバーテキスト
  - アルゴリズム的なストーリーテイングとプロセス中心主義
  - 結束性(cohesion)と「よく整えられた」物語
- コンピュータを用いないメディアにおけるインタラクティブなストーリー
  - ロールプレイングゲーム
  - 発話によるストーリーテリング
  - 文学における例　―　ウリポ、ナブコフ『Pale Fire』、など
  - 演劇における例　―　フォーラム・シアター、被抑圧者の演劇、など
- インタラクティブ・ゲームに代わる固定したストーリー

- ビジュアル・ノベル（日本のジャンル）
- 探索的な物語
  - ハイパーテキスト
- 分岐ツリー：物語分岐、会話の分岐
- 創発的な物語のアプローチ
  - ストーリー生成
- インタラクティブな小説
- コラボレーションによるストーリーテリング
  - ウェブを使ったコラボレーションによるストーリー
  - 現実代替ゲーム
  - MUDs, MMOGs

その他のメディアのためのライティング

- 小説のライティング
- 劇のライティング
  - 映画のライティング
  - 芝居のライティング
  - ラジオのライティング

抽象的なオーディオビジュアルの物語

- 記号学と象徴学
- ムードとドラマを音楽と音で創造する

3.8 ゲーム・プロダクション

ゲーム開発のマネジメントにおける実践的な挑戦

ゲームはソフトウェア製作において最も複雑な形式の一つであり、ゲーム開発とパブリッシングは複雑でコラボレーティブな仕事です。ソフトウェア開発におけるあらゆる技術的挑戦と共に、ここではコアトピックとして、設計文書の問題、コンテンツ製作、チームの役割、グループのダイナミクス、リスク分析、人材管理、開発過程管理について取り上げます。ゲーム製作に関する著作はますます増加しているが、ソフトウェア開発とプロジェクト管理における豊富な蓄積から、以下の項目をコアトピックとしてピックアップします。

人材管理とコラボレーションによる開発

開発プロジェクトに予算をつける

標準的な産業の情報源、産業情報　－　取引、産業の異なる分野との取引、他のメディアとの取引

典型的な予算と予算の分類

チーム形成

- 仕事の説明書き
- リクルーティング、研修
- 才能、経験、予算のバランシング

ゲーム開発のライフサイクル

- プリプロダクション ／ プロダクション ／ テスト
- 出荷と、顧客のロイヤリティーの維持
- 製作過程の様々なアプローチ
    - ウォーターフォール、スパイラル、V 型、進化型、スクラム／アジャイル、イテレーティブ型／インクリメンタル型開発、ラピッド・プロトタイピング、など
    - 強みと弱点
    - ゲーム開発特有の問題

ワークフロー

- どのツールをいつ使うかを知ること
- コンピュータ支援によるコラボレーティブ・ワークツールを評価し使用すること
    - バグトラッキング・システム
    - Wiki
    - スプレッドシート
    - メッセージボード／フォーラム
    - データベース
    - バージョン管理
- 問題評価と適切なリソースの調査
- タスク分析
    - バックログの作成
    - 特徴を抽出

グループ・ダイナミクス

- チーム構築

- はっきりした役割を構築しゴールを明確にする
- 開発チームの現状
- 実行力のあるチームを構築する
  - 統一されたゲームプレイ・ビジョンを実現するためにチームとして連携する
  - リーダーシップ、委任、責任
  - チームメンバー間のやり取りを定義する

デザインと開発ドキュメント

- なぜドキュメントが必要か？
- 何をドキュメント化するべきか？
- どこまでドキュメントを作成すれば十分であるか？／余分であるか？
- ドキュメントを読むのは誰か？
- 絵コンテに対する文書は必要か？
- テキストでないドキュメント：プロトタイプを使用する、物理的モデル、絵...
- デザインと開発文書
  - コンセプト文書／プロポーザル
  - ゲーム仕様書
  - デザイン文書
  - ストーリー・バイブル
  - スクリプト
  - アート・バイブル
  - 絵コンテ
  - 技術設計文書
  - スケジュールとビジネル／マーケティング文書
  - テストプラン

テスティング

- コードレビューとテスト環境の構築（Test harnesses）
- テスト設計と品質管理からのフィードバックとの連携
- バグ修正、バグデータベース、安定したコードベースの構築

スケジュールとタイムマネジメント

- スケジュール作成
- スケジュールの目標　－　マイルストーン
- クオリティーと現実とのバランス
- スケジュールの活用、出荷への調整

- 典型的なスケジュール
- 完成間近の期間の問題
- 生活の質の問題

コミュニケーション・スキル

- レトリック
- 同僚とも、スーパーバイザーとも部下ともコミュニケーション
  - 文書においても会話においても明確にコミュニケーションをすること
  - コラボレーション・スキル　－　同一の用語で話すこと
  - コラボレーション・スキル　－　分野をまたいで話すこと
    （「同じ用語」で話そうとすること。用語のギャップを埋めること）

開発、品質管理、セールス、マーケティング、パブリック・リレーションズ（public relations）と経営

ローカリゼーションの問題、プロセス、スキル

- ゲーム周辺のライティング
  - パッケージング
  - プレイヤーマニュアル、ウェブサイト、など。

製品のポストモーテム

- 意思決定の事後評価
  - デザインにおける決定
  - プロセスにおける決定
  - ビジネスにおける決定

品質管理

- プランニングと品質管理プラン

欠陥のトラッキング

テクニカルなレビューと検査

アーキテクチャ

- ソフトウェア・テスティング
  - ベータ・テスティング

- システム・テスティング
- コードレビューとテスト環境の構築
- テスト設計と品質管理からのフィードバックとの連携
- バグ修正、バグデータベース、安定したコードベースの構築

- ゲーム・テスティング

マーケティングの活用

- マーケティング・プランとスケジュール
- アセットのニーズのマーケティング

3.9 ゲームのビジネス

ゲームの経済、法律、政策の側面

ゲーム産業の経済－ゲームはどのように資金を得るか、市場に出されるか、販売されるか、そして、パブリッシャー、ディベロッパー、卸業者、販促者、小売業者、そして、その他、さまざまな企業の関係について取り上げます。

マーケットと産業のトレンド、ライセンシング・マネジメント、企業の動向と商品価値、そして、主要なゲーム・プラットホームにおけるビジネスの形の違い、これらはすべてゲームのビジネス的側面において重要なものです。加えて、ゲーム、開発者、プレイヤーに関する法律的な問題、知的財産と契約法は本章におけるコアトピックです。最後に、ゲームコンテンツに対する法律制定と制限に対する社会的、或いは政府の力についても取り上げます。

ゲーム産業の経済

- 小売業者、陳列棚のスペース、デジタル・ディストリビューション：どのようにユーザーはゲームにたどり着くか？
- プラットホームの選択 － コンソール、PC、携帯ゲーム機、モバイル開発の間のトレードオフ
- 開発の国際化 ／ グローバル化
  - 海外への委託 ／ アウトソーシング
  - 参入障壁の変化 （知識、技術、マンパワー）
  - 文化、距離、タイムゾーンにおける課題
- 販売チャンネル
- アイテム課金、一括払い、月額料金、基本的にフリーだが、幾つかのサービスは料金を払ったメンバーのみに公開など
- バーチャル・ワールドやMMOにおけるリアルマネーの取引
- さまざまな購入方法とストリームによる購入（マイクロソフト・アーケード、

PS Home...）

- 独立系のゲーム開発、 パブリッシャー／ディベロッパーのゲーム開発の相違
- 違法コピー（海賊）行為

ユーザー

- マーケティングとセールス：現在における、ゲームのユーザーへの届き方
- さまざまなゲームジャンルにおけるユーザーを理解する
- ユーザーを捉える方法、そして、一度捉えたユーザーを維持する方法
- 消費者の行動とその心理（多様な種類、多様な広がりを持つ消費者は何を欲するのか？）

パブリッシャー／ディベロッパーの関係

- 取引
  - 扱う領域
  - どのように行うか
  - 予測される検討内容
  - グリーンライティング・プロセス（Grennlighting process）
- 日々の活動：一度サインした後、発生するインタラクションとプロセス
- マイルストーンのレビュー

知的財産

- テクノロジーとコピーライト
  - キーケース
  - 主要なプレイヤー
- コンテンツ
- ライセンス
  - ライセンスの獲得
  - ライセンスの使用
  - ライセンス所持者との仕事
- コピー（海賊）行為

特許とゲーム産業

契約

- パブリッシャー／ディベロッパー
- 雇用主／被雇用者

- 契約人

コンテンツ規制

- ゲームレーティングと分類
  - ESRB (North America)
  - PEGI (Europe)
  - CERO (Japan)
- 政府による規制
  - 北米
  - ヨーロッパ ／ オセアニア
  - アジア

4 学位取得プログラムと大学卒業者の未来

ほとんどの学生にとって、ゲーム教育を受けることはゲーム産業におけるキャリア形成の一つのステップです。しかし、多くの学生は（そして学術関係者の何割かは）ゲーム産業におけるポジションについて包括的な視野を持っていません。ゲーム教育は決して、特定のゲーム産業におけるポジションに結び付いているものではありません。しかし、仕事のタイプを理解しておくことは重要であり、これらは大学卒業者にとって有用な情報です。

ゲーム産業におけるポジションは、ゲーム産業で働く人々に対する Gamasutra の年次調査をもとにしています。より包括的な仕事のリストは、IGDA Credits and Awards Committee によって編纂されており、以下の文書で参照できます。
http://www.igda.org/wiki/I_%E2%80%93_IGDA_STANDARDIZED_ROLES

1. プログラミング
2. アート & アニメーション
3. ゲームデザイン
4. ゲームライティング
5. プロダクション
6. 品質管理（QA）
7. オーディオ
8. ビジネス & リーガル
9. マーケティング
10. コンシューマ・サポート
11. コミュニティ・サポート

我々はそれぞれの仕事に対して、期待される役割、責任、教育をより詳細を記述して行くつもりです。

5 制度への配慮

ゲーム教育のプログラムは制度のサイズの違いによって全く異なります。つまり、そのプログラムが学部生か、院生か、専門向けなのか、あるいは、どの学部がどのプログラムを主導しているのか、そのプログラムが産業におけるエキスパートの協力を得ているのか、どのように学生がプログラムに参加しているのか、によって異なっています。

数個のコースや、コースのクラスターは、広い世界観を持ち、十分な能力を得て成熟した学生を育てています。その一方でゲームに特化したプログラムも、また、ゲーム産業の仕事に就くために、十分な素養を身につけた情熱を持つ生徒を生み出しています。

それぞれの制度は、自分たちが持つリソースや、現在の構造、特有の背景の上にプログラムの改善を必要としています。強力なプログラムを構築するために必要な要素は、以下のようなものがあります

1. アドバイザリーボード（可能ならばその地域のプロフェッショナル）
2. ポートフォリオ開発へのフォーカス（卒業要件、プロ開発者、学術関係者の判定）
3. 開発スタジオ、企業、NPO を含むコミュニティ組織とのインターンシップの連携
4. 各地の IGDA 支部との連携（学生メンバーシップ）
5. 産業での経験を持つ教員（特に開発プログラムにおいて）
6. 実験室とライブラリ（生徒が持っていないハードウェ／ソフトウェア／ゲームへのアクセス）
7. 講演プログラム（現役のプロ開発者をキャンバスへ招聘する）
8. 複合クラス（プログラマーとアーティストが幾つかのプロジェクトをチームベースで行うコース）
9. 特別プロジェクト（生徒による mod 製作、教室外でのプロジェクト）

新しくコースを始めようとする教育者へのアドバイス

1. 懐疑的な同僚がいることを前提に進めるべきでしょう。ゲーム開発教育は新しい教育の領域なので、しばしば、学術系の同僚は単に学術的なフィールドとして正しく受け入れることができません。
2. あなたが作ろうとしているプログラムが、自分の所属している場所に合っているかどうかを確かめるべきでしょう。学部生のプログラムは、既存の学部教育の中に創設されることが多いです。これは、プログラム設計の性質に、明らかに大きなイン

パクトを与えるものであり、組織は、プログラムを開始する前に、自分たちがそのプログラムを実行できるだけの能力を持っているかを確かめるようにきちんと忠告を受けるべきでしょう。

3. 生徒（または企業）が、あなたのプログラムが真剣なものと捉えているかどうかを、すぐに見極めることができます。数多くのプログラムが利用できるようになるにつれて、プログラムが設置されている組織に非常に批判的になります。その理由は、学術的に信頼できるかどうか、というよりも、それが就職に結び付くかということからです。我々の経験から言えば、学生は（もしくは、より重要なことに従業員は）こういった見方に同調し、すぐに、こういった目的を通して組織を見ることになるになります。
4. ゲーム開発において用いられるテクノロジーとツールは、高価であることが多いことを理解するべきです。
5. ゲームをプレイするために、教育プログラムに参加したいと思う同僚にもゲームをプレイさせる機会をつくるべきです。コアゲーマーになる必要はなくても、ゲームをプレイするという最初の経験は、ゲーム分野を教える、あるいは、ゲームを研究する人間にとって本質的なものだからです。

6 次のステップ

「カリキュラムフレームワーク 2008」は GDC 2008（Game Developers Conference）の IGDA Education Summit で発表されます。そのフィードバックは、このドキュメントに反映されます。次のメジャーバージョンは、2010 の終わりか、2001 年になる予定です。

7 さらに詳細な情報

さらに詳細な情報については以下を参照してください。

IGDA web site： http：//igda.org/

IGDA Game Education SIG： http：//igda.org/education/

IGDA Game Education Wiki： http：//igda.org/wikiGame_Education_SIG

IGDA Game Education Listserv：

http://seven.pairlist.net/mailmanlistinfo/game_edu

8 謝辞

この教育委員会は、このドキュメントのために貢献し、支えて下さったすべての学術研究者、開発者、学生の皆様に心から感謝を申し上げます。もし、以下に名前のリストに抜けがあるようでしたら、お詫びします。

Mark Baldwin, University of Advancing Technology
Katrin Becker, University of Calgary
Robin Burke, DePaul University
Philip Carlisle, University of Bolton
Rob Catto, Full Sail
Doug Church, Electronic Arts
Jason Della Rocca, International Game Developer's Association
Mark Doughty, University of Lincoln
Tom Dowd, Columbia College
Michael Edwards, Parsons The New School for Design
Clara Fernandez, Massachusetts Institute of Technology
Tracy Fullerton, University of Southern California
Paul Gadi, Anino Games
Kimberly Gregson, Ithaca College
Susan Gold
Mark Harmon, Westwood College Online
Ian Horswill, Northwestern University
Robin Hunicke, Electronic Arts
Katherine Isbister, Center for Computer Games Research
IT University of Copenhagen
Michael J. Katchabaw, The University of Western Ontario
Jay Laird, Northeastern University
Jack Lew, Electronic Arts
Samuel Lewis, Cartoon Network
Martin Masek, Edith Cowan University
Bruce Maxim, University of Michigan-Dearborn
Gary Miller, Full Sail
Alex Mitchell, National University of Singapore
Frans Mäyrä, University of Tampere
Andrew Nealen, Technische Universitaet Berlin
Lennart Nacke, Blekinge Institute of Technology
Magy Seif El-Nasr, Simon Frasier University
Casey O'Donnell, Rensselaer Polytechnic Institute
Kevin O'Gorman, American InterContinental University
Lisa Patacchiola Bristol Community College

Yusuf Pisan, University of Technology, Sydney
Mike Reddy
Damon Redmond, Shaba Studios
Norb Rozek, Frozen Code Base Studios
Katie Salen, Parson the New School for Design, Institute of Play
Ian Schreiber
Marla Schweppe, Rochester Institute of Technology
Warren Spector, Junction Point Studios
Tom Smith, THQ
Jolyon Webb, Blitz Games
Elias Wyber, Murdoch University
Catherine Wyman, DeVry Institute
Eric Zimmerman, gameLab

付録 A

CMU Program

寄稿者：Drew Davidson

Carnegie Mellon University のエンターテインメント・テクノロジーセンター（ETC）は、様々な分野に応用されるインタラクティブエンターテインメントのための最初の大学教育プログラムです。ETC は、2 年間のエンターテインメント・テクノロジーの修士課程を提供します。コンピュータ・サイエンス学科と、美術学科の修士号を同時に取れるという、他に類を見ないものです。

ETC では、学生は異なる学問分野にまたがるチームで、効率的な作業の仕方を学び、魅力的でインタラクティブな経験を得ます。彼らには、技術者とアーティストからなる 1 つのチームでともに作業する環境が準備されます。この環境には、たとえばテーマパーク、子供と科学の博物館、Web サイト、モバイルコンピューティング、ビデオゲームなどがあります。

我々のカリキュラムはプロジェクトベースで、講義がベースとなる学習過程はほとんどありません。1 期は、バーチャル世界の構築、ETC の基礎、即興での演技、ビジュアルストーリーテリングといった必修コースだけです。その後の 3 つの期では、CMU が教えられる様々なコースを、学生が自由に選択します。残った時間は、プロジェクトコースに当てます。

プロジェクトコースには指導者が割り当てられます。異なる学問分野にまたがる学生のチームは、1 期の間、1 つのオフィスを共有します。それぞれのチームには、作品制作を通して学生を導く、アドバイザー（指導者）がいます。作品は、試作品の場合もありますが、時折、展示に値する完成品となります。 いくつかのチームには、外部のスポンサーやクライアントがつきます。これらの長期プロジェクトは、外部クライアント、指導者の調査、あるいは、学生の発案から生じます。

学期の間中、学生には徹底的な評価が与えられます。これらの評価は、過程と、成果物に分けられます。過程の評価では、独創的なとりくみや、挑戦を重視します。成果物の評価では、グループがチームとして共同で作成した成果物を評価します。

ピッツバーグは、（まだ）インタラクティブエンターテインメントやビデオゲーム業界の中心ではありません。我々は学生を、学生自らが手続きを行った、カンファレンスや会

社訪問に送り出しています。ETC の全学生は、在学中の 2 年間の夏期に、会社でのインターンシップが奨励され、サポートされています。いくつかの成功した ETC プロジェクトが、起業を望んだ ETC の卒業生に地元でスピンオフ会社をはじめるきっかけを与えたことは、特筆すべきことです。それらの企業は、インタラクティブメディアとビデオゲームの未来を作っています。ピッツバーグでは、現在 6 つのスピンオフ会社が操業しています。

今後の話として、ETC は ETC Global へと発展しつつあります。ETC は 2006 年のアデレード、オーストラリア、カリフォルニアを皮切りに、2007 年には韓国のソウル、シンガポール、日本の大阪といった世界中にキャンパスを設立しています。これらのキャンパスは、ETC 全体の一部であり、学生は 2 年間の単位を取るために、学期やプロジェクト単位でキャンパスを移動することができます。めまぐるしく動くグローバルな世界で、我々の学生にとって、この大きな経験は、複数の大陸に分散したチームを有することが当たり前になっていると思われる未来の仕事のための、準備になると考えています。

ETC は非常に特異です。ETC では、コラボレート、実験、疑問解決の繰り返し、の方法を学ぶことを通して、学生が制作と挑戦の経験をつみ、リーダーシップ、革新性、コミュニケーションを身に付けることを重視します。我校の卒業生は、彼らが目指す分野に、有望な影響を与えるための十分な準備ができています。 ETC は、学生に産業界への興味をもたせるための場所です。

Full Sail Program

寄稿者：Rob Catto

Full Sail Real World Education は、フロリダのオーランド郊外にある大学です。当大学は、エンターテインメント産業のキャリアを実現させるための革新的教育の先駆者です。1998 年に、大学での社会人教育の需要を満たすために、ゲームのデザインと開発の科学準学士が設立されました。2004 年に、このプログラムはゲーム開発の科学学士のためのものに昇格されました。

学生は 1 日 8 時間構成の専門的なクラスと、24 時間連続のスケジュールで、"real world"教育を経験します。この教育方針は、フロリダ高等教育協会（the Florida Association of Postsecondary Schools and Colleges）から、"最も革新的なプログラム"の賞を与えられています。

ゲーム開発の学位は、ゲーム開発に特化したソフトウェア・エンジニアリングのプログラムです。カリキュラムは、計算機械（Computing Machinery）協会から提供されたソフトウェア・エンジニアリングのフレームワークに合わせています。学生は 1 週間で 2 つのコースを履修し、それは 4 週間、あるいは 8 週間続きます。学生は 1 週間のうち、5 日間はクラスに出席します。クラスは、4 時間の講義と、4 時間の実習からなります。

コースカリキュラムは次に示す主なセクションと履修単位時間からなります。

●一般教育－24 履修単位時間
●専門的な演習－8 履修単位時間
●コンピュータの基礎－24 履修単位時間
●ソフトウェア・エンジニアリング 42 履修単位時間
●ゲームデザイン－10 履修単位時間
●プロジェクト開発－26 履修単位時間

プロジェクト開発コースは、学生をゲーム開発プロジェクトに集中させます。このプロジェクトでは、プランニングやドキュメンテーションと同様に、しっかりとしたチームワークが特に重要視されます。学生は、同級生の評価と適切な欠点調査メカニズムを重要視する、ソフトウェアの品質保証サイクルも学びます。また、学生には、テクニカルデザインドキュメンテーションの制作と管理、ゲームテクノロジーの実装、品質保証サイクルのデザインと実現も課せられます。それらは、学生に強固な基盤を与えるためにデザイン

されており、受講するコースにおけるマイルストーンになります。

Full Sail のゲーム開発プログラムは、毎年 12 回の開始時期がありますが、秋期が最も人数が多いです。

ゲーム開発プログラムに関するさらなる情報と、他の Full Sail のプログラムは、以下で確認することができます。
http://www.fullsail.edu./

Northumbria University

イギリス、ニューカッスルにある、Northumbria University の、コンピュータゲームソフトウェア・エンジニアリングの学位取得のためのプログラム
寄稿者：Dan Hodgson

Northumbria University の、コンピュータゲームソフトウェア・エンジニアリングの理学士のための特別コースは、伝統的なコンピュータ・サイエンス学科と、ゲーム開発に関する特別なモジュール（短い単位時間）を組み合わせます。学習計画の主目的は、産業界でプログラマーの役割を担う優秀な卒業生を育てることと、コンピューティング分野の広範囲にわたる具体的な課題を支援することに向けられています。ゲームデザインの入門的な紹介と、アセットの制作ついても教育された卒業生たちは、彼らを取り巻く仕事をよく理解するでしょう。この学習計画には、4 年以上の期間が用意されています。ここで示すのは、学生が学ぶモジュールのアウトラインです。

1 年目

- ゲーム・プログラミング 1&2：プログラミングの原則と共に C++を基礎から学ぶ、DirectX による小さな 2D ゲーム開発の演習
- コンピュータゲームのための数学 1：三角関数、複素数、行列、ベクトル、微積分、パーティクルダイナミクス
- コンピュータゲームデザイン：良いゲームデザインの原則、産業の本質、産業の問題点と倫理、ゲームデザインドキュメントの創造
- リレーショナルデータベース：SQL、Oracle を用いたデータベースデザイン
- Web デザイン：HTML、JavaScript、PHP など
- コンピュータシステムの基礎：基礎的なプロセッサのアーキテクチャ、アセンブラ

2 年目

- ゲーム・プログラミング 3&4：DirectX によるさらなる 2D プログラミング、コンパイラの記述、GameCube devkits 上での 3D プログラミング
- コンピュータゲームのための数学 2：微分方程式、リジッドボディダイナミクス、計算技術、3D 表現、3D でのアフィン変換
- ゲームシステムのアーキテクチャ：中級レベルのプロセッサのアーキテクチャ、アセンブラ（pong on 6800）
- HCI（Human Computer Interaction～人間とコンピュータの相互作用）の基礎：相互作用の原則、インターフェース・デザイン、周辺機器の使用、情報のプレゼンテーション

●システムデザインとアーキテクチャ：UML（Unified Modeling Language～統一モデリング言語)を用いたトップダウンによるシステムデザイン
●オブジェクト指向のゲーム開発： UML を用いた、ゲームシステムのためのオブジェクト指向デザインパターンから見たボトムアップ
●プロジェクト・マネジメント、専門能力

3 年目
産業界での実習

4 年目
●ゲーム事例の課題：大がかりなゲームデモ制作のグループ課題
●上級レベルのゲームシステムのためのアーキテクチャ：ゲームシステムのためのアーキテクチャの促進、GameCube と ARM アーキテクチャを含む
●上級レベルのゲームのためのプログラミング課題：最新のゲーム開発のための有意義なプログラミング技術に関する上級レベルのモジュール、分散および、ネットワークプログラミングにも言及する
●マルチメディア制作：3D モデリングと映像制作
●コンピュータゲームの人工知能
●レベル 3 オプションモジュール

さらなる情報は、以下を参照してください。
http://www.gamesdegree.com

USC Program

寄稿者：Tracy Fullerton tfullerton [at] cinema.usc.edu

Southern California University の映画美術学部は、2002 年にインタラクティブメディアの美術修士課程を、2005 年にインタラクティブエンターテインメントの美術学士課程を作りました。これらの学習計画は、Electronic Arts 社から提供されており、ゲームとインタラクティブ産業のための次世代のデザイナーとプロデューサー育成に焦点を絞っています。学習計画で特に重要視しているのは、手続きに乗っ取った開発能力の習得、革新的なゲームデザイン、コラボレーションによる制作作業です。

2006 年に、Southern California University の Viterbi School of Engineering（工学部）では、ゲームに特化したコンピュータ・サイエンスの修士課程と学士課程を作りました。これらの学習計画には、映画美術学部の核心をなす制作カリキュラムを統合したものが含まれています。学習計画は、チームワークとコラボレーションによる制作を経験したゲームプログラマーの育成を重視しています。

2007 年に Roski School of Fine Arts（美術学部）は、2 つの小さな学習計画を作りました。これらの学習計画は、映画美術学部と、工学部の両方に統合されました。これらは、正式な学位取得のための学習計画に加えられました。Annenberg School of Communications は、ゲーム文化研究のコースを提供しました。大学全体にわたる調査集団である the Games ORU も、カリキュラムのさらなる統合、調査実習、ゲームとゲームデザインに取り組むすべての Southern California University の学習計画間のコラボレーションを活性化するために、2007 年に設立されました。

統合された制作コースは、映画美術学部と工学部の両方の、美術修士（Master of Fine Arts）、理学修士（Master of Science）、美術学士（Bachelor of Arts）、理学学士（Bachelor of Science）の学生のためのものです。

ゲームデザインのワークショップ－最初のゲームデザインコースは、革新的なゲーム構造のデザインに焦点を絞ります。 デザイン工程を繰り返し、試作品制作とテストプレイを行います。中級レベルのゲームのデザインと開発では、小さなデジタルゲームのデザインと開発を担当する 2 つのチームを置きます。重要視するのは、試作品制作、テストプレイ、プロダクションマネジメントです。

上級レベルのゲーム課題－2 セメスター（2～6 月）の上級レベルの制作クラスでは、6

～10 人の学生で構成されるチームが、制作現場の専門家の指導で、革新的なゲームを制作します。このクラスの学生は、制作メンバーか、プロジェクトリードのいずれかとして加えられます。

プロジェクトリードやゲームコンセプトの担当者は、能力も含めた成績的な競争の結果を勘案して、前回のチームリーダと産業側のメンターが選びます。

この中核となる制作コースに加えて、映画美術学部の学生は、映画とビデオの制作、脚本、インタラクティブメディアの歴史と理論、ゲームビジネス、サウンドデザイン、ビジュアル・デザインなどのクラスも受講します。美術学士の学生も、これらすべての教育条件を 4 年間で満たさなければなりません。美術修士の学生は、卒業制作と、この分野に新たな知識を提供できる研究を記した論文を作成しなければなりません。

科学修士と科学学士の学生は、統合された制作コースと、人工知能やグラフィックプログラムのようなゲームに特化したコンピュータ・サイエンスのテーマに加えて、中核となるコンピュータ・サイエンスのカリキュラムを履修します。すべてのコースのリストは、以下で確認できます。

B.A. can be found at : http://interactive.usc.edu/about/
http://www.cs.usc.edu/admissions/graduate/msgames.htm and
http://www.cs.usc.edu/current/undergrad/default.htm

UTS Program

寄稿者：Yusuf Pisan ypisan at it.UTS.edu.au

University of Technology, Sydney（UTS）では、ゲーム開発の科学修士コースを 2006 年に開始しました。オーストラリアの大学課程は、一般的には 3 年です。ゲーム開発の科学修士課程は 4 年で、学生は 2 年間を TAFE（専門技術後続教育：Technical And Further Education～地域大学や訓練機関に似ています）で学び、その後の 2 年間を UTS で学びます。最初の 2 年間は、プログラミングスキルに焦点を絞った学習、ゲームエンジンの働きの学習、シリアス・ゲームを制作するチームでの実習を行います。UTS での次の 2 年間は、情報技術学部で、コンピュータ・サイエンスと、コンピュータゲームの知識を深めることを目指します。

TAFE では、学生は毎週 30 時間のクラスを受講します。クラスの大部分は、ワークショップかスタジオ形式で行われます。UTS では、一般的に、講義を 8 時間、実習とチュートリアルを 8 時間、毎週受講します。UTS では、学生はゲームに関する次のコース（他のコンピュータ・サイエンスのコースと似ています）を受講します。

●コンピュータ・グラフィックスー伝統的なグラフィックスのコースです。少しばかりゲームについてのオリエンテーションも含みます。

●ゲームデザインーテキストベースのゲーム、2D ゲーム、デザインドキュメントの記述、を通して生じるデザインに焦点を絞ります。最低限のプログラミングの知識を必要とします。

●ゲーム・プログラミングー高度なプログラミングコースです。学生は、ゲームエンジンを 1 から構築します。BSP ツリーやいくつかのシンプルな人工知能技術といった、高度なグラフィック技術を重要視します。(有限状態機械、LUA をベースにしたスクリプトとルールの作成)

●3D コンピュータニメーションーキャラクターの動きとショートストーリに焦点を絞ります。学生は、Maya を用いて、5 分間のアニメーションを制作します。

●CG レンダリング技術ー学生は、1 から、レイトレーシングエンジンを構築します。

●システム開発プロジェクトー学生は 2 セメスター以上の時間をかけて、グループプロジ

ェクトを完成させます。グループは、大抵、大人数でのチームワークが要求される 10～15 人の学生で構成されます。

●CG プロジェクト－学生は 1 からアイデアを考え出して、独自プロジェクトを実行します。

各年とも、TAFE では 30 人、UTS では 20 人を上限としています。学生は適応することを要求され、TAFE から UTS に進学できる者が選ばれます。学生は TAFE からゲーム開発の学位を受け取ることはできません。UTS では、学生は一般に，それぞれ 14 週をかけて、4 つのコースを受講します。2 年間で、最大 16 コースを受講します。

UTS コースの公式情報は、下記で確認できます。
http://www.handbook.UTS.edu.au/it/ug/c10229.html

WPI Program

寄稿者：David Finkel　（dfinkel@WPI.edu）

マサチューセッツ州、ウースターの、Worcester Polytechnic Institute のインタラクティブメディアとゲーム開発専攻（IMGD：Interactive Media and Game Development）(http://www.WPI.edu/Academics/Majors/IMGD/)は、2005 年の秋に始まりました。この専攻は、コンピュータ・サイエンス学科と、人文美術学科が共同で運営しています。これは 4 年間の学部プログラムで、科学学士号が与えられます。

WPI の学部では、28 の直接指導時間と合わせて、7 週間のターム（期間）が与えられています。大学での 4 年間には 4 つのタームがあり、学生は、典型的には、それぞれのタームで 3 つのコースを専攻します。学部ではプログラミングに重点を置いたテクニカル専攻と、ビジュアルアーツ、音楽、脚本に重点を置いたアート専攻の 2 つの進路を提供します。

次の 2 つが、学習プログラムの案内指針です。

1）すべての学生が、それぞれの専攻で、いくつかの実習コースを受講する。

2）アートコースの学生と、テクニカルコースの学生は、複数のコースとプロジェクトの中で、共同実習を行う。

IMGD 専攻の学生は、つぎの 3 つの核となるコースのなかから、2 つを専攻することを要求されます。

●IMGD の重要な学習－ゲーム要素の重要な全体像を学びます。また、ゲーム分析のための共通のボキャブラリを確立します。

●ゲーム開発過程－このコースでは、ゲーム制作のための異なる貢献者の役割を話し合います。学生はシンプルなゲームを制作します。

●ゲームの脚本

学生は、社会と倫理問題に重点を置いた 2 つのコースのうちの 1 つも受講します。1 つは、インタラクティブメディアとゲームに関する社会問題のコース、もう 1 つは、コンピュータゲームの哲学と倫理に関するコースです。

これらのコースに加えて、学生は、自分達の進路のための専門的なコースも受講しま

す。

テクニカル専攻の学生は、コンピュータ・サイエンスの 10 のコースを受講します。これらのコースは、ソフトウェア・エンジニアリングや CG など、ゲーム開発に関するコンピュータ・サイエンスの分野に特化しています。学生は、IMGD の 2 つの上級技術コースも受講します。最初のコースでは、低いレベルのゲーム・プログラミングに焦点を絞ります。2 番目のコースでは、人工知能やゲームのネットワークなどの、高いレベルのテーマを扱います。

アート専攻の学生は、人文と美術に関する 10 のコースと、IMGD の 2 つの上級美術コースを受講します。これらのコースは美術アセットの制作と、それをゲームに組み込むことを含んでいます。 最も重要な到達点は、専攻資格の認定プロジェクトです。このプロジェクトは、3 つのコース分のボリュームがあります。アート専攻とテクニカル専攻の学生は、ゲームか、他のインタラクティブメディアのプロジェクトのデザインと制作を共同で行います。

財　団　法　人　Ｊ　Ｋ　Ａ
平成 20 年度デジタルコンテンツの保護・活用に関する調査研究等補助事業

デジタルコンテンツ制作の
先端技術応用に関する調査研究
報　告　書

発　行　平成 21 年 3 月

発行者　財団法人デジタルコンテンツ協会
〒102-0082　東京都千代田区一番町 23－3
日本生命一番町ビル LB
TEL.03(3512)3900
FAX.03(3512)3908