ビデオゲーム記述をドメインとするプログラミング言語

企業科学専攻システムズマネジメントコース
主指導教員 久野 靖 教授
副指導教員 大木 敦雄 准教授
副指導教員 吉田 健一 教授
西森丈俊

1　研究の背景と目的

ビデオゲームは娯楽商品であることから,「面白さ」を要求されるが,「面白さ」の定義や合理的な評価は難しいため，制作前に面白さを十分に検討することができない．そこで，ビデオゲーム開発ではトライアンドエラーを繰り返したり複数のバージョンを作り比較する等，開発中に試行錯誤する探索的な開発手法が用いられる [新宅 03].

また近年，情報技術の進歩を背景に，ビデオゲーム開発には，3D グラフィクスや DVD 等の新しい技術的要素が取り入れられている．このため，より高水準で大量のコンテンツを実現できるようになり，ビデオゲームソフトウェア開発は大規模化してきている．

大規模化していく中でトライアンドエラー開発をすることは難しい．そこで，特に面白さの核となる，ゲーム中のインタラクション性，競合性，不確実性やゲームルール等の「ゲームシステム」と呼ばれる部分については，C/C++ 言語等の汎用的なプログラミング言語ではなく，ゲームルールやゲームの要素を表現するのに適した機能を持つゲームシステム記述ツールを用いることが，ビデオゲーム開発においては一般的となっている．

ゲームシステム記述ツールを用いることで，面白さに直結する要素の記述が容易になる．さらに，C/C++ 言語等を使った高度なプログラミング技術を必要としないので，プログラミングに精通していないゲームシステムの設計を担当する「ゲームデザイナー」が利用でき，プログラマーの協力無しに直接ゲームシステムの記述可能になる．このため，従来, 開発を進める上で必要であったプログラマーとのやり取りが軽減され，トライアンドエラーによる開発が容易になる．

しかし，現在利用されているゲームシステム記述ツールは，十分な記述性を提供していない．そのため，十分なトライアンドエラーを実施を難しくし，目的とするゲームの面白さを達成できない要因となる．

ゲームシステムを記述するために必要な要素としては次のものが挙げられる．よって，記述ツールはこれらを容易に記述できることが望まれる．

状態遷移処理 ゲーム中のキャラクタは様々な状態を持ち，様々な条件に応じた状態間遷移をする．

時間に沿った処理 ゲーム中のキャラクタは，時間の経過に従って，形状や挙動を変化させながら処理を進める．言語はゲーム時間やこのフレームにあわせたが記述しやすくなければならない．

並行処理 ビデオゲームでは複数のキャラクタが自律的に動作する．また，1 つのキャラクタが複数の処理を同時に実行する場合もある．

キャラクタ間の通信処理 多数の並行した処理がお互いに様々な通信をすることでゲームは進行する．

これらの要素は，数値の列挙やグラフィカルなユーザーインターフェースだけでは記述が難しい．そこで，記述ツールとしては，ゲームシステムの記述に適し，プログラミングに精通していないゲームデザイナーにも利用可能なプログラミング言語が適している．

本研究はこの 4 つ要素の記述性を達成する，実際の開発で役立つプログラミング言語の開発を行ない，その評価を行うことを目的とする．

2　関連研究

今日，ゲームシステムを記述するツールは，開発するゲームの種類やその利用目的等に応じて，様々なものが利用されている．それらを調査し，次の 4 つに分類した．

- ビデオゲーム開発を目的としたプログラミング言語
- 統合的なビデオゲーム開発環境
- 個々のゲームの開発専用に作られた記述ツール
- その他の開発ツール

ビデオゲーム開発を目的としたプログラミング言語は，アプリケーションに組み込んで利用するもので，汎用のプログラミング言語と違いゲームシステムの記述に適した機能を持つものも多い．しかし，先に述べた 4 つの要件全てを十分に満たしているものは無い．

近年多く利用されているビデオゲームのための統合開発環境は，ゲーム開発に有用な機能を多く持ち，グラフィカルなインターフェースを採用することで開発を容易にしている．しかし，ゲームシステムを記述するために採用されているプログラミング言語は多くが Javascript や C#等の汎用的なプログラミング言語であるため，先に述べた 4 つの要件を十分に満たしていない．

ビデオゲーム開発では，ゲームシステム記述ツールを独自に開発していることも多い [西森 03]．これらの記述ツールは，開発目的とするゲームに特化した，一般的なプログラミング言語等と違い非常に独特な文法や記法を持つ．そのため，目的とするゲームアプリケーションの記述には有効ではあるが，それ以外の異なるタイプのゲーム記述には適していない．

アニメーションオーサリングツールや，エージェントシミュレーションツール等のビデオゲームに類似したアプリケーションの開発を目的としたツールも多く存在する．これらは比較的容易にアプリケーション開発が可能であるが，要件を十分に満たさないため，記述ツールとして適切ではない．

さらに，既存のツールは，キャラクタの状態遷移や時間の流れに沿った処理，及び並行処理を平易に記述可能なものが多い一方で，キャラクタ間の通信については，適切な機能を提供しているものが無い．そこで本研究では，計算機科学分野でプロセスやスレッド間の通信方法として研究されている，Tuple Space や join calculus 等の機構や通信モデルの検討も行った．

Tuple Space は分散システムのための通信モデルであり，プログラミング言語 LINDA[Gel85] で最初に提案された．多くの並列／分散処理が Tuple Space を用いることで簡潔に記述でき，Tuple Space の機能はプログラミング言語 LINDA とは独立していることから，LINDA 以外の言語にも採用され，機能拡張に関する研究も複数なされている．

join calculus[CG96] も，分散システムのために提案された並行計算モデルである．複数の並列動作するプロセスがメッセージの送受信を行うチャネルを経由して通信しながら計算が進行する．また，join pattern と呼ばれる，複数の異なるメッセージを同時に受信できる記法により，プロセスの処理と通信処理を分けて記述でき，プロセスを配線で繋いでいくようなプログラミングが可能である．

3 ゲームシステム記述に適したプログラミング言語の開発

前章の調査を土台として，1 章で述べた 4 つの要件を満たすように設計した S540 という新しいプログラミング言語を開発した [西森 03]．

S540 は次の機能を持つ．

- キャラクタをクラスとして記述し，クラスの中で状態遷移処理を記述する機能を持つオブジェクト指向言語．
- 一定時間処理を停止したり，条件を満たすまで処理を停止する機能．
- クラスからキャラクタを生成すると自立的に処理を開始．クラス内で複数の自立した状態遷移処理を記述可能．
- キャラクタ間の通信処理記述のために，Tuple Space を導入．
- 日本語での記述機能．

S540 は，キャラクタをオブジェクト指向言語における「クラス」として記述する．クラスの中では，他のプログラミング言語にある変数や制御構造などの記述機能のほかに，状態遷移処理を記述しやすい仕組みを導入した．

また，指定した時間やある条件を満たすまで処理を停止する記述機能により，時間に沿った処理の記述も可能とした．

S540 は，各クラスは自律して動作するスレッドを 1 つ以上持ち，キャラクタとして生成されると，他のキャラクタとは独立して処理を開始する仕組みとした．さらに，複数の状態遷移処理を 1 つのクラスに記述可能にすることで，1 つのキャラクタで複数の並行処理を実行する記述もできるようにした．

キャラクタ間の通信を記述するためのモデルとしては，Tuple Space[Gel85] を採用した．Tuple Space は，非同期的な通信や，相手を特定しない通信も記述でき

る機能がありながら，Tuple の読み書きだけで表現するのでシンプルで理解しやすいので，開発時のトライアンドエラーにも有効と判断した．

ゲームデザイン上の仕様とプログラミング言語による記述の間の対応づけも，開発のしやすさに影響があると考え，開発時に使用する言葉をそのままプログラムとして記述できるようにプログラムは日本語で記述できる設計とした．

この S540 をゲームデザイナー 3 人のプロトタイプゲーム開発プロジェクトに導入した．導入初期は言語の理解や記述方法での試行錯誤が続いたが，その後は、3 人で記述範囲の分担が行われ、目的とするゲームのために S540 に必要な機能の要望も出るようになった．さらに開発が進むと、ゲームシステムはほぼゲームデザイナーが S540 を使って実装し、プログラマーはゲームデザイナーからの機能的な要望を実装していくという流れで開発が進むようになった．このことから S540 を使用することにより，ゲームデザイナーがプログラマーの助けなしにゲームシステムの実装を行えることが確認できた．

また，ゲーム開発プロジェクト終了後に記述されたプログラムの調査と使用者のインタビューを行った．その結果次の評価が得られた．

- トライアンドエラーの幅を広げる効果があった
- 言語の機能が十分でない．特に Tuple Space は，バグの発生原因になりやすく，通信タイミングが記述しづらいという問題点が判明した．

S540 を使用した開発では，S540 の利用者であるゲームデザイナーがプログラマーの助け無しにゲームシステムの実装を行っており，利用者がそれまでに経験した開発プロジェクトよりも，トライアンドエラーが容易で，本研究の目的がある程度達成されたことが確認できた．

一方，キャラクタ間の通信処理の記述のために導入した Tuple Space は，Tuple の読み書きの記述が冗長になり，誤りの原因になりやすいことや，通信遅延やタイミングを管理できないという問題点が明らかになった．

4　キャラクタ間通信の記述性の改善

S540 を使用した開発の結果から，Tuple Space に代わるゲームシステムの記述に適した通信モデルが必要であることがわかった．そこで，新しい通信モデル Join Token を提案し [NKa]，その評価のために実装したプログラミング言語 Mogemoge の開発を行った．

Join Token は，Tuple Space と join calculus[CG96] という通信モデルを組み合わせた新しい通信モデルである．Join Token は, Tuple Space の Tuple を読み書きするというわかりやすさを持ちつつも，Tuple Space よりもゲームキャラクタやその状態に依存した通信を，ハンドラと呼ばれるキャラクタ間の通信を明示的に記述する機能を持つ．

この Join Token を評価するために，数個のサンプルゲーム制作を行った [NKb]．制作は，目的とするゲームのルールを列挙し，複数のキャラクタが関係して通信を行うルールを Join Token を利用して記述し，その他のキャラクタ個別のルールは Mogemoge を使ったオブジェクト指向プログラミングで記述する，という方法で進めた．

その結果，どのゲームにおいても，キャラクタ間の通信処理が必要なルールは，ルールとハンドラを 1 対 1 で簡潔に記述することができた．キャラクタ固有の処理と，キャラクタ間の通信処理は分けて記述されることから，記述が複雑になることもなく，Join Token はゲームシステムの記述に適した通信モデルであると評価できた．

さらに，Join Token を使用せずに同じゲームをプログラミング言語 Ruby を使用して制作し，Join Token を使用した場合との比較検討も行った．制作した Ruby/Tk の Balloon ゲームプログラムは，Mogemoge によるプログラムの行数は 357 行であるのに対して，Ruby によるプログラムは 436 行となった．Mogemoge と Ruby の文法的な違いはあるが，キャラクタの定義等は類似したものとなっており，違いの多くは，グラフィクスの処理とキャラクタ間の関係処理の部分となった．

Ruby プログラムでも，Ruby の機能を利用して衝突検査と衝突処理を分離して記述して，衝突に関するルールは 1 対 1 でメソッドとして記述することができた．しかし，この実装をするためには，キャラクタの組み合わせを処理するプログラムを記述しなければならなかった．Join Token ではこの処理に関する記述は不必要である．

また，Ruby 版にはその他にも次の違いがあった．

- 衝突ではないルールは Join Token を用いた時と違い複雑な記述となった．
- Join Token ではキャラクタの状態まで記述することが可能であるが Ruby では難しい．

- 1 度衝突したら二度と他のキャラクタと衝突するような記述は Ruby では簡単ではない．Join Token ではハンドラがトークンを取り除く指定をすれば良い．
- Ruby では複数のルールが適用できる場合その優先度を実装するためには複雑な記述を必要とする．Join Token は記述順が処理の優先度なのでわかりやすく表現できる．

この結果，Join Token は，列挙したゲームルールのうち，キャラクタ間の通信を簡潔にわかりやすく記述でき，Join Token を使用しな場合と比較してもキャラクタ間の通信や相互関係が記述しやすいと評価できた．また，Tuple Space の問題点も解決できた．

5 結論

本論文の研究は，ビデオゲーム開発におけるトライアンドエラーを容易にするためのゲームシステム記述ツールの設計と開発を目的とした．

ビデオゲーム開発では，ゲームに求められる面白さの定義が難しいことから，トライアンドエラーによる探索的な手法で進められており，トライアンドエラーを容易にするために，ゲームシステム記述ツールが用いられているが，既存の記述ツールは十分な機能を提供しておらず，大規模化する近年の開発では，高機能なプログラミング言語が必要であることを議論した．

そこで，議論を土台として，新しいゲームシステム記述ツール S540 を開発し，実際の開発現場で適用して評価を行った．その結果，トライアンドエラーがより容易に行えることが確認でき，ビデオゲーム開発プロセスの改善に貢献できる結果を得ることができた．

しかし，S540 にゲームシステムの記述要素として必要な，キャラクタ間の通信処理の記述性に問題があることも判明した．これは採用した通信モデル Tuple Space によるもので，この Tuple Space に代わる新しい通信モデルが必要であるとわかった．そこで，新しく Join Token という通信モデルの提案と実装を行い，いくつかのシンプルなゲーム制作を行い，Join Token の評価を行った．その結果，Join Token は Tuple Space よりもゲームシステムにおけるキャラクタ間通信の記述性に優れており，特に複雑になりやすい複数のキャラクタ間のゲームルールを簡潔にわかりやすく記述できると評価できた．

以上のように本研究では，ビデオゲーム開発におけるトライアンドエラーを容易にするためのゲームシステム記述ツールの設計と開発を行い，ビデオゲーム開発プロセスの改善に貢献できる結果を得ることができた．

しかし，S540 は導入前後の比較検証ができていないこと，S540 の通信記述機能の改善として Join Token を実装したが，最初の 4 つの機能的要件を全て十分に満たす記述ツールの開発までには至っていないことや，記述ツールの機能的要件が 1 章で述べた 4 つで十分であるかどうかが明らかにできていない等, 課題も残されている．今後はこれらの解決に取り組むため，さらに研究を進めていきたいと考えている．

参考文献

[新宅 03] 新宅純二郎, 田中辰雄, 柳川範之．ゲーム産業の経済分析．東洋経済新報社, 2003.

[Gel85] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 1, pp. 80–112, 1985.

[CG96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A Calculus of Mobile Agents. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, pp. 406–421. Springer-Verlag, 1996.

投稿論文 (査読付き)

[西森 03] 西森丈俊, 久野靖．アクションゲーム記述に特化した言語．情報処理学会論文誌 プログラミング (PRO), Vol. 44, No. SIG15 PRO19, pp. 36–54, 2003.

[NKa] Taketoshi Nishimori and Yasushi Kuno. Join token: A language mechanism for programming interactive games. *Entertainment Computing*, Elsevier. (採録予定 `http://www.sciencedirect.com/science/article/pii/S1875952111000292`).

[NKb] Taketoshi Nishimori and Yasushi Kuno. Join Token-Based Event Handling: A Comprehensive Framework for Game Programming. *Software Languages Engineering*, Vol. 6940 of *Lecture Notes in Computer Science*. Springer. (採録予定).