ゲームシステムのアーキテクチャと開発環境

– 開発の柔軟性と安全性、効率化のために –

2014 年 1 月 20 日 初版

板垣 衛

■ 改訂履歴

版	リリース	担当	改訂内容
初版	2014 年 1 月 20 日	板垣 衛	（初版）

■ 目次

■ 概略

一連のドキュメントで示すゲームシステムのアーキテクチャと開発環境の概要を説明する。

■ 目的

本書は、一連のドキュメントの全体像の理解を得ることと、各ドキュメントへの誘導を目的としたものである。

■ 基本設計思想

ゲームシステムのアーキテクチャは、下記の基本思想に基づいて設計する。

・ 融通の利かない縦割りのシステムにならないようにする。

- これによって起こりえる問題点としては、「同じような処理が幾つも作られる」、「システムどうしの複雑な依存関係が生じる」、「タイトルに特化した処理との切り分けがしにくくなり、結果的に処理の汎用性が失われる」といった事がある。
  - 例えば、「キャラシステム」「マップシステム」「エフェクトシステム」といった区分けで処理を作成すると、「オブジェクトの親子連結処理」や「ビルボード処理」などをそれぞれの処理系で独自に実装したり、描画の優先度を解消するために処理系どうしの複雑で危うい依存関係が生じたりといった問題が起こることがある。
  - この対処として、「役割」をベースにしたシステムの切り分けを行う。各処理系は、徹底して役割を遵守して連携する。
    - 【役割例１】リソースの生成／削除の役割はリソースマネージャのみ

      他のシステムは、いかなる都合があっても直接リソースの生成／削除を行ってはいけない。そのような操作は必ずリソースマネージャに依頼する。リソースマネージャは、確実に安全なタイミングで依頼を遂行する。
    - 【役割例２】画面上に登場する座標を持った要素は全てシーン管理に登録する

      マップも、キャラも、エフェクトも、頭上アイコンも、SE も、見えないファンクションボックス（イベント発生ポイント）も、定点カメラも、全てシーン管理に登録する。

シーングラフが全てのオブジェクトの親子関係に責任を持つため、手に持った剣や炎エフェクトを体にまとまったモンスター、プロペラ音を出し続けるヘリコプター、動くエレベーター上に配置されたファンクションボックスなどの親子関係の解消は、全て共通処理でまかなわれる。

また、オブジェクト間の処理順序や描画順序もシーン管理に任せることにより、不整合が起きないように調整される。

・ 【役割例３】シーンの切り替えは、必ずメッセージキューに要求する

例えば、同一フレームで「メインメニューの呼び出し」「ゲームオーバー」「イベント発生点への到達」が同時に起こったとしても、それぞれを判定する各処理系で、直接このようなシーン切り替えを伴う処理を行ってはいけない。

メッセージキューに登録された要求をシーン管理が全てチェックした上で、状況的に可能で、かつ、もっとも優先度の高いものが採択される。

これにより、各処理系で状況判断することなく、かつ、競合処理の同時発生のような不具合の発生を防ぐ。各処理のプログラムもシンプルになる。

- 以上により、システム間の強い依存関係は生じるものの、各処理を必要最小限の構成にすることができる。

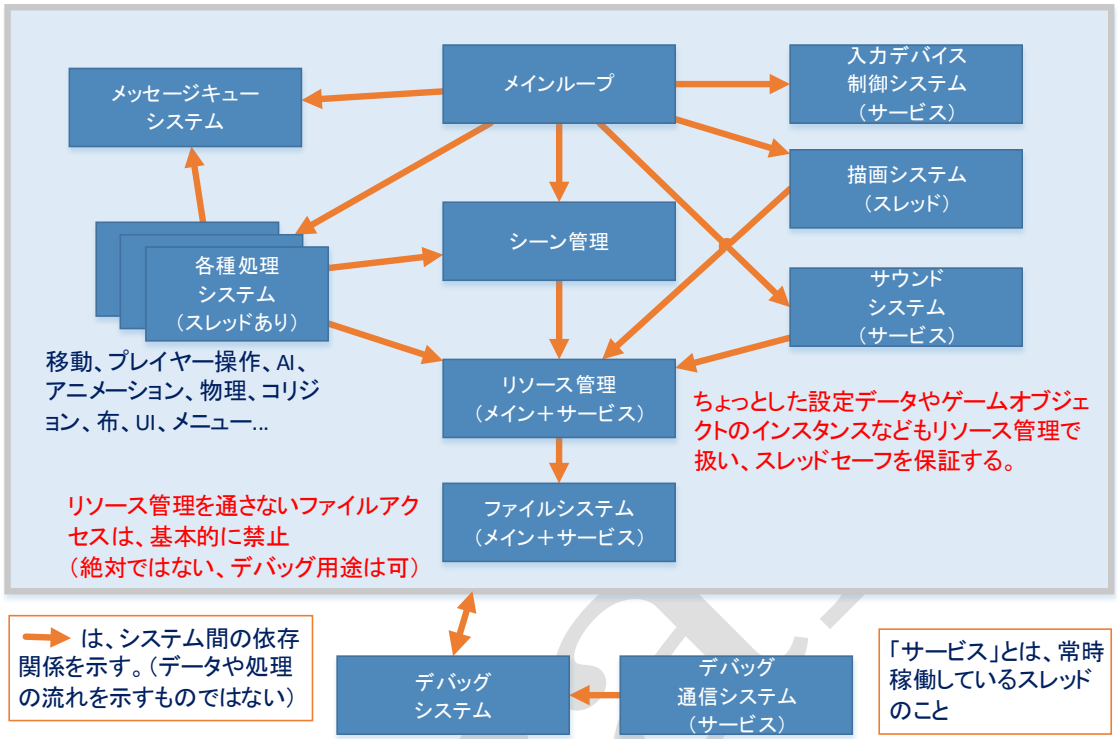
・ 生産性（開発作業効率）をできる限り考慮する。
  - ゲームを再起動せずにリソースを再読み込みする仕組みや、新たな処理を追加する際に極力手間がかからないような構造などを考慮した設計とする。
  - 役割分担の徹底は、各所で同様の処理を作らなければならなくなるようなプログラミングの手間と試行錯誤を軽減し、生産性の向上につながる。
  - 例えば、前述の「メッセージキュー」の利用のように、「シーン切り替えが必要な時は、余計な状況判断はせずに、メッセージキューに要求するだけでよい」というルールは、シンプルで生産的である。

・ 処理効率をできる限り考慮する。
  - 処理を作成するにあたって、配慮すべき「安全性」「生産性」「処理効率」の優先順位は、多くの場合はこの順序である。ただし、絶対のルールではないため、快適な動作のために安全性を犠牲にすることがあっても、それが大きく効果的であるなら、やむなしとする。
  - 「安全性」「生産性」を維持しつつ、「処理効率」を向上するために、「マルチスレッドの最適化」を特に意識した設計とする。
  - マルチスレッドを前提としたゲームループと、スレッドセーフな各システを設計する。
  - マルチスレッド処理を最大限に効率化するために、効率的なロック機構と、分かり易いスレッド同期（ロックなど）のルールを設ける。

・ 安全性をできる限り考慮する。
  - 役割分担の徹底は、各所に同類の雑多な処理が作られることを防ぎ、安全性の向上につながる。
  - 例えば、前述の「メッセージキュー」の利用のような処理は、不整合を起こさないための状況判断が一箇所にまとまり、それを利用するシステムは余計な処理を行う必要がないため、安全である。
  - マルチスレッド化する処理とスレッド同期のルールをあらかじめ明確にしておき、それ以外のスレッドの利用と同期方法（もしくは同期しないこと）を徹底して禁止することで、安全性を確保する。

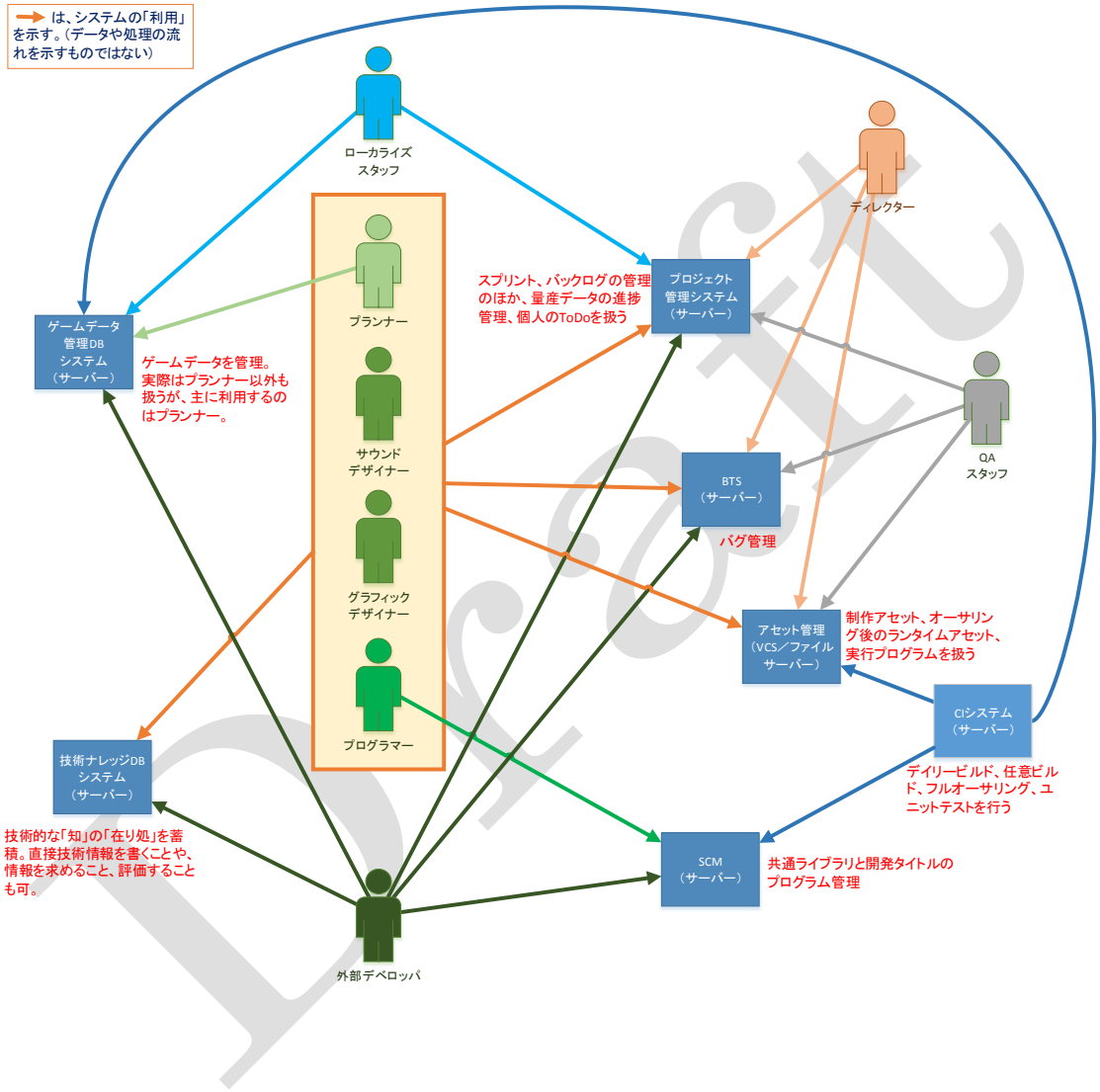
・ デバッグ効率をできる限り考慮する。
  - デバッグ機能の追加が行い易い設計にする。
  - 問題発生時にデバッグ情報を収集しやすい設計にする。
  - ユニットテストの環境を整え、誰でもすぐにテストの処理の追加と実行ができる環境にする。
    - ユニットテストによって継続的に繰り返し実施されるテストは、誤った修正によって他のロジックの挙動を変えてしまうような問題を早期発見し、結果として開発を効率化する。

■ ゲームシステムのアーキテクチャ

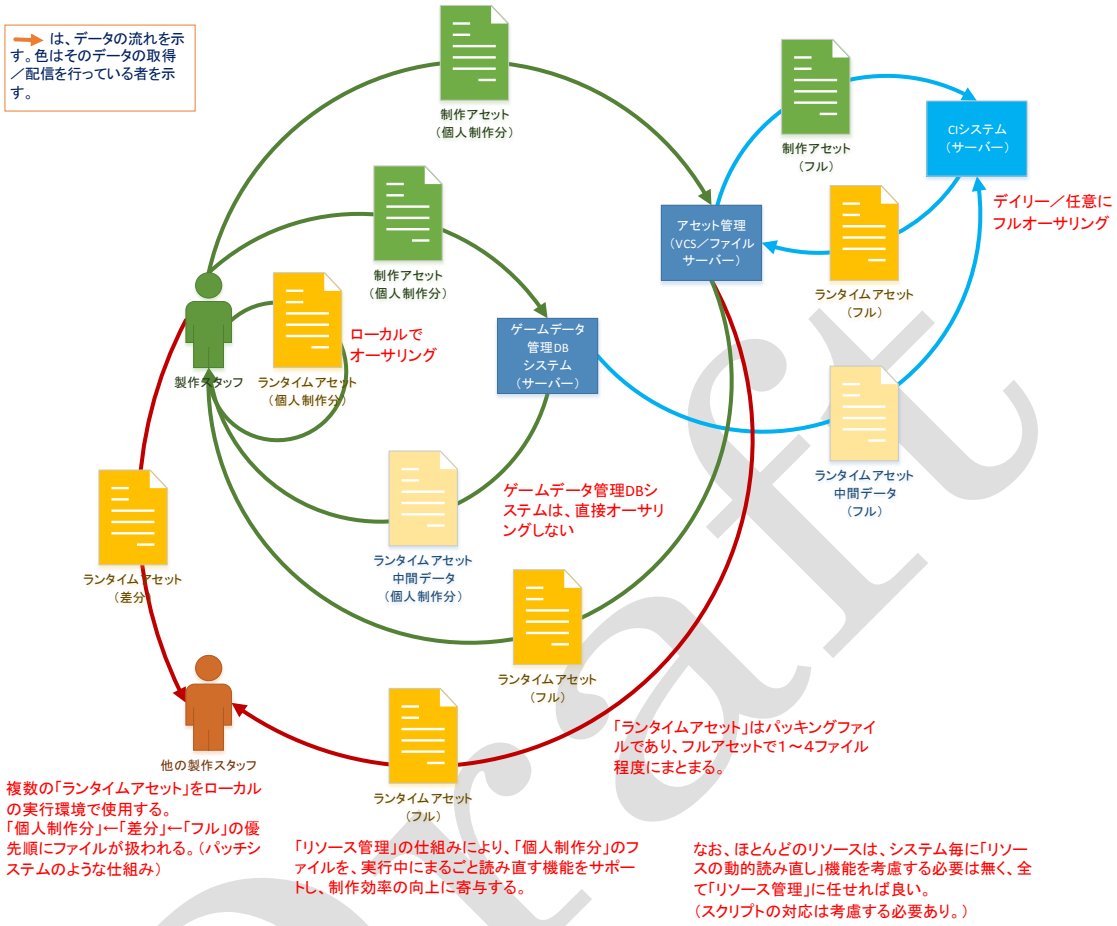


■ 開発環境

▼ 各種管理システムと制作スタッフの関係



▼ アセット管理



■ ドキュメント一覧

以下、一連のドキュメントをカテゴリ別に列挙する。
各ドキュメントは、カテゴリごとのフォルダに配置している。

なお、各システムの仕様・設計については、基本的に草案レベルのものであり、その動作や効果を確実に保証できる状態には至ってはいない。

▼ まとめ

・ ゲームシステムのアーキテクチャと開発環境（本書）

▼ プログラミング系

＜プログラミング Tips＞

・ コンパイルを効率化するための**コーディング手法**
・ 本当にちょっとした**プログラミング Tips**
・ オブジェクト指向と C++
・ プログラミング**禁則事項**
・ 効果的な**テンプレートテクニック**
・ **デザインパターン**の活用
・ プレイヤーに不満を感じさせないための**乱数制御**

＜マルチスレッド制御＞

・ **マルチスレットプログラミング**の基礎
・ 効率化と安全性のための**ロック制御**
・ **「サービス」**によるマルチスレッドの効率化

▼ ゲームシステム系

＜ゲームループ管理＞

・ マルチスレッドによる**ゲームループ管理**
・ 安全性をのための**メッセージキュー管理**とイベントドリブン

＜ファイルシステム＞

・ 開発を効率化するための**ファイルシステム**

＜リソース管理＞

・ 開発の効率化と安全性のための**リソース管理**

＜シーン管理＞

・ ゲーム全体を円滑に制御するための**シーン管理**

＜デバイス管理＞

・ 反応性と安全性を考慮した**入力デバイス管理**

＜メモリ管理＞

・ ゲーム制御のための**メモリ管理**方針
・ ヒープメモリとスラブアロケータを併用した**メモリ管理**
・ 様々なメモリ管理手法と**共通アロケータインターフェース**

＜ゲームデータ管理＞

・ **ゲームデータ**仕様
・ **ゲームデータ管理** DB システム
・ ローカライズのための**テキスト管理**構造

＜シリアライズ＞

・ セーブデータのための**シリアライズ**処理

＜スクリプト管理＞

・ スクリプトの生産性向上のための**プロパティマップ**

＜カメラシステム＞

・ **カメラ処理**の効率化手法

＜サウンドシステム＞

・ サラウンドとリソース管理を効率化するための**サウンドシステム**

＜AI＞

・ プランナーのための **AI システム**考察

＜レベル管理＞

・ オープンワールドのための**レベル管理**

＜イベントシステム＞

・ 効果的な**イベントストリーミングシステム**

＜デバッグシステム＞

・ **デバッグ制御システム**
・ **ユニットテスト**と継続的ビルド
・ 効果的な**デバッグログ**と**アサーション**

▼ 開発環境系

＜プロジェクト管理＞

・ ゲーム開発のための**プロジェクト管理**
・ **プロジェクト管理** Web システム
・ **技術ナレッジ DB** システム

＜アセット管理＞

・ 効果的な**ランタイムアセット管理**

＜開発環境＞

・ 複数タイトルにまたがる効率的な**フレームワーク管理**

■ 【目的別】ドキュメント一覧

以下、一連のドキュメントを目的別にまとめ直して列挙する。
複数の目的にまたがるものもある。

▼ ゲームシステムに関するドキュメント

＜基本システム＞

・ マルチスレッドによる**ゲームループ管理**
・ ゲーム全体を円滑に制御するための**シーン管理**
・ 安全性をのための**メッセージキュー管理**とイベントドリブン
・ 開発の効率化と安全性のための**リソース管理**
・ 開発を効率化するための**ファイルシステム**
・ 反応性と安全性を考慮した**入力デバイス管理**
・ サラウンドとリソース管理を効率化するための**サウンドシステム**

＜基本仕様＞

・ ゲーム制御のための**メモリ管理**方針
・ **ゲームデータ**仕様
・ **カメラ処理**の効率化手法

＜拡張仕様＞

・ オープンワールドのための**レベル管理**
・ 効果的な**イベントストリーミングシステム**
・ プランナーのための **AI システム**考察

＜デバッグシステム＞

・ **デバッグ制御システム**
・ 効果的な**デバッグログ**と**アサーション**

▼ 開発環境に関するドキュメント

＜アセット管理＞

- 効果的な**ランタイムアセット管理**
- 開発を効率化するための**ファイルシステム**

＜ツール関係＞

- **ゲームデータ**仕様
- **ゲームデータ管理 DB** システム
- ローカライズのための**テキスト管理**構造
- 技術ナレッジ DB システム

＜プログラム関係＞

- 複数タイトルにまたがる効率的な**フレームワーク管理**
- **ユニットテスト**と継続的ビルド

▼ プロジェクト管理に関するドキュメント

＜プロジェクト管理＞

- ゲーム開発のための**プロジェクト管理**
- 複数タイトルにまたがる効率的な**フレームワーク管理**

＜管理ツール＞

- **プロジェクト管理** Web システム
- **技術ナレッジ DB** システム

▼ プログラミングに関するドキュメント

＜プログラミングの基本と規約＞

- コンパイルを効率化するための**コーディング手法**
- 本当にちょっとした**プログラミング Tips**
- オブジェクト指向と C++
- プログラミング**禁則事項**
- **マルチスレッドプログラミング**の基礎
- 複数タイトルにまたがる効率的な**フレームワーク管理**

＜プログラミングテクニック／ライブラリ＞

- 効果的な**テンプレートテクニック**

・ **デザインパターン**の活用
・ プレイヤーに不満を感じさせないための**乱数制御**
・ セーブデータのための**シリアライズ**処理
・ スクリプトの生産性向上のための**プロパティマップ**

＜処理仕様＞
・ **ゲームデータ**仕様
・ **カメラ処理**の効率化手法
・ **「サービス」**によるマルチスレッドの効率化
・ 効率化と安全性のための**ロック制御**
・ ゲーム制御のための**メモリ管理**方針
・ ヒープメモリとスラブアロケータを併用した**メモリ管理**
・ 様々なメモリ管理手法と**共通アロケータインターフェース**
・ 開発の効率化と安全性のための**リソース管理**

＜デバッグ＞
・ **デバッグ制御**システム
・ **ユニットテスト**と継続的ビルド
・ 効果的な**デバッグログ**と**アサーション**

■ 一連のドキュメントでほとんど扱っていない事

一連のドキュメントはデータ処理とプログラミングの基本部分を中心にまとめており、それ以外の事はほとんど扱っていない。例えば、以下のような要素はゲーム制作に非常に重要なものであるが、一連のドキュメントではほとんど触れていない。

・ 3D グラフィック全般（ツール、アセット管理、データ構造、アニメーション、描画）
・ 2D グラフィック全般（ツール、アセット管理、データ構造、アニメーション、描画）
・ ライティング（ツール、アセット管理、データ構造、描画）
・ エフェクトシステム（パーティクルエフェクトシステムなど）
・ ポストエフェクト（アニメーション、描画）
・ グラフィック全般（シェーダー、描画）
・ シェーダー管理
・ UI／HUD（メニュー）システム
・ カメラ制御
・ サウンド（制御、データ）

・ ストリーミングシステム（音声、映像）
・ 物理演算・布処理
・ ネットワーク関係
・ 汎用／内製スクリプトエンジン
・ DCC ツール／グラフィックツール関係（プラグインなど）
・ サウンドツール関係
・ イベントシーン制作ツール
など...

■■以上■■

■ 索引

索引項目が見つかりません。

ゲームシステムのアーキテクチャと開発環境

以　上