反応性と安全性を考慮した入力デバイス管理

– シンプルな処理で柔軟な制御 –

2014 年 3 月 13 日 初稿

板垣 衛

■ 改訂履歴

稿	改定日	改訂者	改訂内容
初稿	2014 年 3 月 13 日	板垣 衛	（初稿）

■ 目次

■ 概略

処理落ちに左右されることなく、プレイヤーの操作に的確に反応するための入力デバイス管理のシステムを設計する。

また、複数のコントローラ操作系処理がアクティブになり得る状況で、処理間に複雑な相互依存関係を築くことなく、安全に処理するためのシステムを確立する。

加えて、操作方法の変更要求に柔軟に対応可能なシステムとする。

■ 目的

本書は、コントローラ操作の処理をシンプルにプログラミングできる状態にすることを目的とする。

なお、ポインティングデバイス（タッチ、マウス）、キーボード、ジェスチャー（タッチやマウスによるジェスチャー）、ボタンコマンド（波動拳コマンドやふりほどきガチャ操作など）は対象としないが、ほぼ同じ構造で扱うことが可能である。

■ 要件定義

▼ 基本要件

本書が扱うシステムの基本要件は下記のとおり。

- コントローラ（入力デバイス）の操作に対して、敏感に反応するものとする。
- 「プレイヤーの移動」や「カメラ操作」、ゲーム中に割り込んで出現する「HUD」、「チャットウインドウ」、「デバッグメニュー」など、シーンの切り替えもなく、様々な操作系処理が入り乱れような場面であっても、煩雑な依存関係のない整然としたプログラミングを可能なものとする。
- 「コントローラ」「振動装置」「ポインティングデバイス」などのデバイスが、物理的に一体化しているかどうかに関係なく、それぞれのデバイスを抽象化して、汎用的なプログラミングが可能なものとする。

▼ 対処すべき問題点

基本要件を「対処すべき問題点」としてまとめ直す。

・「ボタン連打」のような操作の反応が悪く、途中途中押したことになっていなかったり、操作の反応が少し遅れたりといった問題に対処する。
  - 「ボタンを押した」→「ボタンを放した」を 1/30sec で判定していたのでは反応が遅い。また、処理落ち時にはタイミングを取りこぼす可能性がある。

・複数の操作処理が同時に反応してしまうような問題に対処する。また、その対処のために、煩雑なプログラムの依存関係が生じないようにする。
  - 例えば、「移動シーン」の最中に、「チャットウインドウ」をいつでもアクティブにできる仕様があったとする。この時、ゲームは止まらず、敵も動くしオートランにも対応するものとする。文字入力のために、キャラの操作はできないが、カメラは動かせる。
  - このような処理の対応としては、プレイヤーのコントローラ操作処理で、チャットウインドウのアクティブ状態を確認するのが最も手っ取り早い。
  - しかし、そのようなプログラミングを行うとなると、チャットウインドウが使える全てのシーン（例えばミニゲームなど）で同様の対応が必要になり、各担当者に対応を求めるなど、手間がかかる。
  - また、「新たに割り込み表示される HUD を追加」、「デバッグ機能で NPC を動かして座標を取得するような機能が欲しい」といった要件が追加されるたびに、煩雑な依存関係が増えていく。
  - 対応を怠った場面があると、「チャットウインドウも反応するし、キャラもいっしょに動く」といった問題を起こす。

・例えば、「チャット画面を開くボタンを□ボタンから[SELECT]ボタンに変更」といった修正要件が発生した場合、各ゲームシーンの担当者にそれぞれ修正してもらう手間がかかる上、不確実、といった問題に対処する。

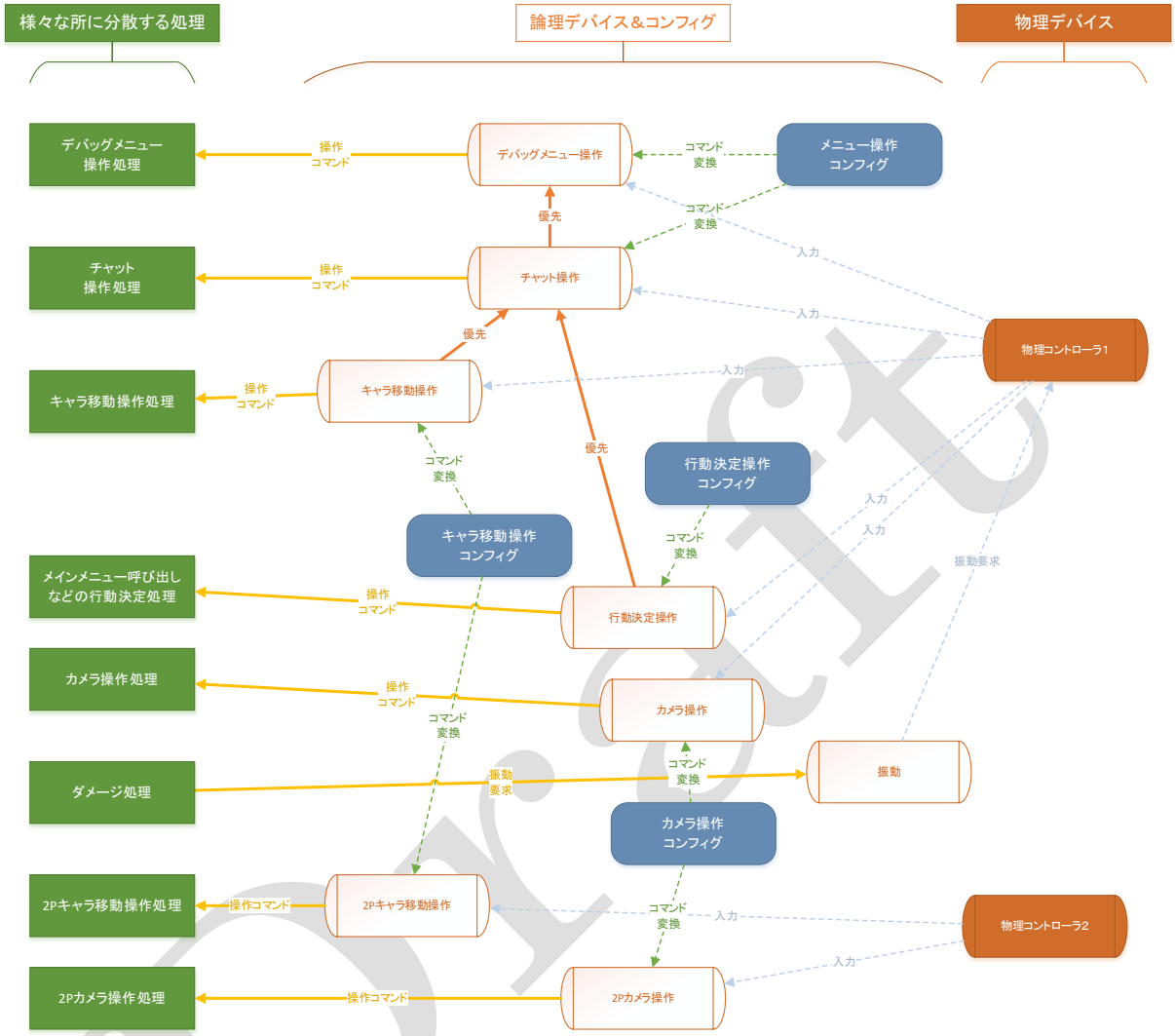
▼ 要求仕様／要件定義

以下、本書が扱うシステムの要件を定義する。なお、要件として不確定の要求仕様も併記する。

・入力デバイスの状態は、メインスレッドのフレームレートや処理落ちに関係なく、専用の「入力デバイス監視スレッド」で監視する。

・「論理デバイス」の概念を採用し、物理デバイスと無関係にゲームのために抽象化したデバイスで制御する。

処理と論理デバイスおよび物理デバイスとの関連図：



- 各操作系処理は、直接物理デバイスを扱わず、「論理デバイス」からコントローラの状態を読み込む。
- 「論理デバイス」では、コントローラの情報をコマンド化して扱う。
  - 各論理デバイスに関連づけられた「コンフィグ」に基づいてコマンド化する。
  - 「コマンド」は、「A ボタン」や「アナログ左 0.8」のような生の情報ではなく、「決定コマンド」、「攻撃コマンド」、「チャットメニュー呼び出しコマンド」といったゲームの要件に変換したものとなる。
  - 「コンフィグ」と「コマンド」は、ゲームタイトルごとに任意に作成する。

    ・ ユーザーによるボタン設定のカスタマイズを反映させても良い。

  - 「コンフィグ」を通すことにより、例えば、「△ボタン」は、「行動決定操作コンフィグ」により、「メインメニューコマンド」になる。

- また、例えば、アナログスティックは、「キャラ移動操作コンフィグ」を通すと、「2D ベクトル」と「長さ」というアナログ情報になる。「メニュー操作コンフィグ」を通した場合は、十字ボタンと同様の「上コマンド」「下コマンド」「右コマンド」「左コマンド」というデジタル情報になる。
- また、「攻撃コマンド押し始め」（最初にボタンを押し始めたことを検出したフレームで反応）、「攻撃コマンド：長押し中 0.5 秒」（ボタンを押している間毎フレーム反応）や、「攻撃コマンド：長押し完了 1.2 秒」（長押ししたボタンを離したフレームで反応）、「攻撃コマンドクリック」（短い時間押して離したフレームで反応）といったコマンドにも対応する。

  ・ 操作状態の時間計測も入力デバイスマネージャが行う。
  ・ 監視の頻度が高いので、正確な計測になる。
  ・ 「無操作時間〇秒」という計測も自動的に行う。

➢ 「論理デバイス」ごとに、毎フレーム発生したコマンドを記録する。

➢ 「論理デバイス」は、プログラム中で任意に「アクティブ化」することで情報を記録するようにする。
- ゲーム中の論理デバイスの追加／削除も可能だが、通常は必要な論理デバイスをまとめて登録しておき、非アクティブ状態にしておく。

➢ 「論理デバイス」には論理デバイス間の優先関係があり、優先度が高い論理デバイスがアクティブ化されると、低い論理デバイスは自動的に一時非アクティブ状態になる。
- 図では優先度の関係を親子関係のように表現しているが、実際は「系統」と「優先度」で扱う。
- 同系統で優先度の高い論理デバイスが有効化されると、低い論理デバイスが一時無効化される。
- 系統の異なる論理デバイスは影響を受けない。例えば、図ではデバッグメニューが有効になってもカメラは影響を受けない。

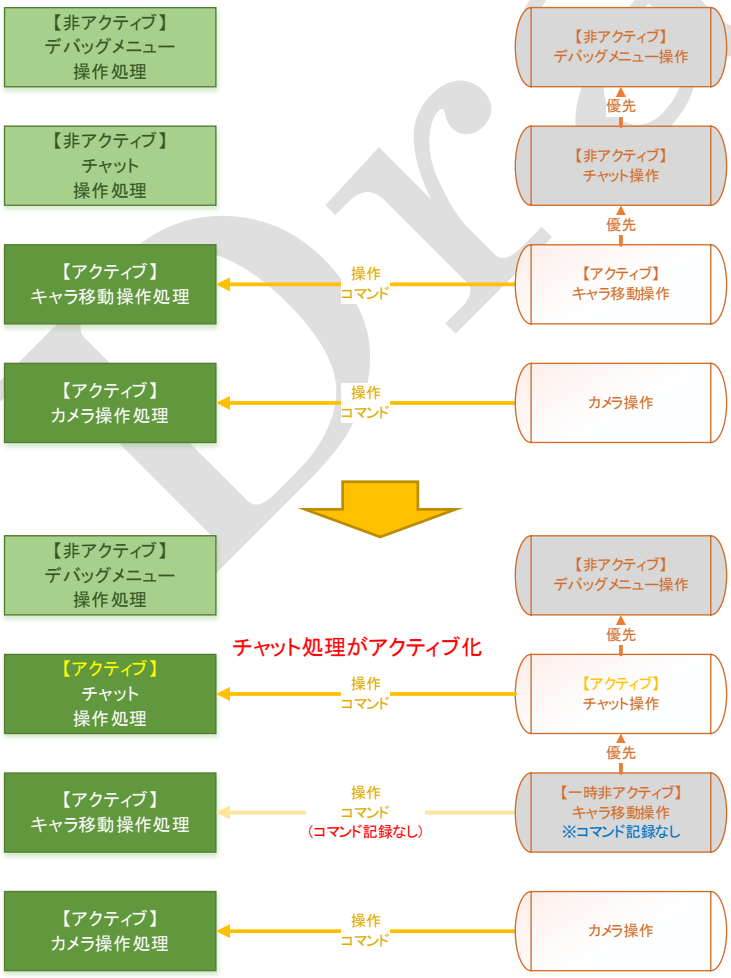
➢ 非アクティブ状態、もしくは、一時非アクティブ状態の論理デバイスは、一切コマンドを記録しない。
- つまり、非アクティブ状態、および、一時非アクティブ状態の論理デバイスにとっては、コントローラ操作をしていないのと同じ状態になる。

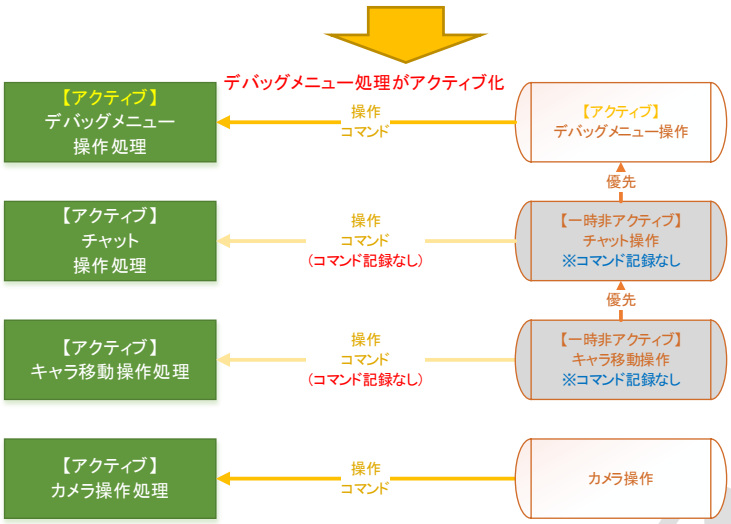
➢ このような論理デバイスの制御により、各操作系処理は、他の操作状態を気にせず、自分用の論理デバイスから情報を読み取れば、自動的に制御が変わる。
- 例えば、チャットメニューがアクティブになった時、キャラ移動操作を止める必要があるが、キャラ移動操作の処理ではとくになんの判定も行う必要はない。
- 論理デバイスの優先度に従い、キャラ移動操作の論理デバイスはコマンドを記録しなくなるため、結果的に操作が効かなくなる。
- チャット画面を閉じる際に、チャット用の論理デバイスを非アクティブにすると、自動的にキ

ャラ移動操作の論理デバイスがアクティブになり、操作が効くようになる。

- 「プレイヤー２」のように、使用するコントローラが異なる操作を扱う場合は、専用の論理デバイスを追加し、物理デバイスと関連づける。この時、「コンフィグ」は「プレイヤー１」用と共有できる。
- 「振動」を扱う場合は、専用の振動処理インタフェースを持ったオブジェクト（クラス）を用意し、物理デバイスと関連づけて操作する。
- 「論理デバイス」は、必要に応じて任意の処理を追加する。
  - 「振動」のほか、「キーボード」「マウス」「タッチデバイス」「コントローラスピーカー」なども、論理デバイスとして、必要に応じて追加する。
- 例えば、ゲーム中になんらかの理由によって「プレイヤー１」が使用するコントローラを「コントローラ１からコントローラ３に変更」といった要件が発生した場合、論理デバイスと関連づける物理デバイスを変更するだけで済む。

処理と論理デバイスのアクティブ化のイメージ：





※論理デバイスの優先順位に基づいて、最も優先される論理デバイスしかコマンドを記録しない状態になる。

※ただし、優先順位の系統が異なる論理デバイスはその影響を受けない。

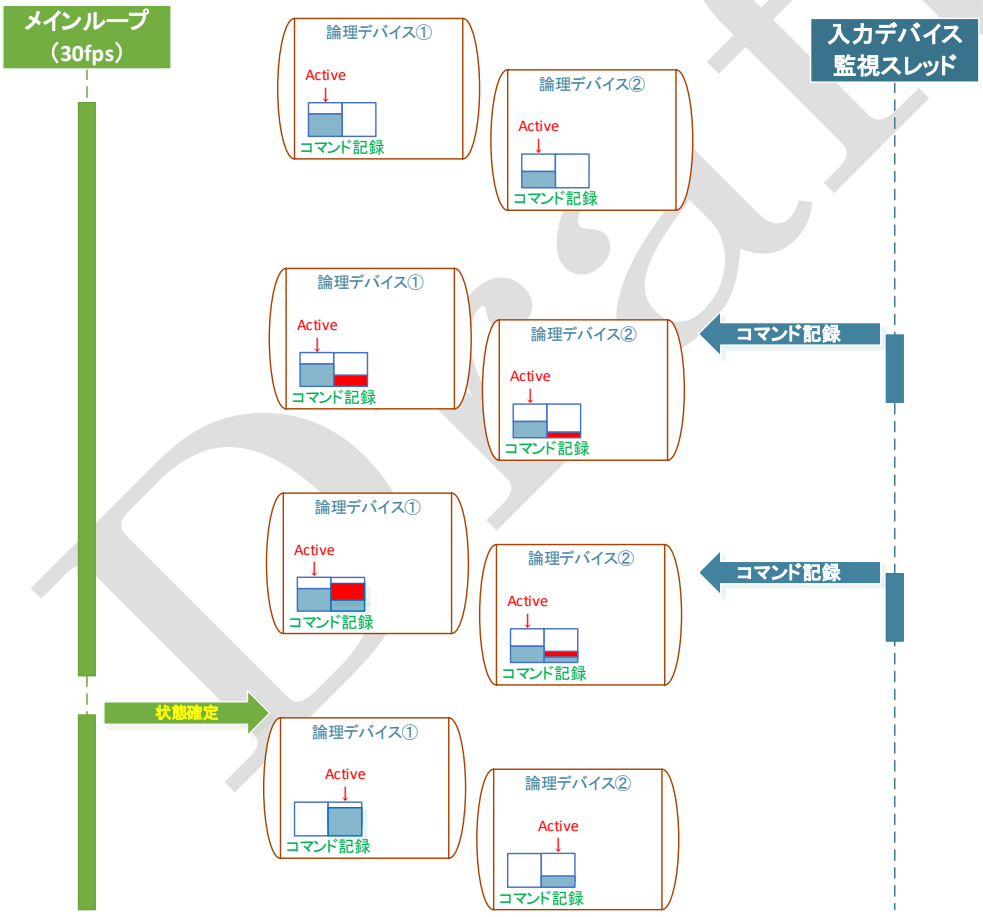
※また、各処理は自分がアクティブと思ってさえいればよく、他の処理を気にかけることなく普通に処理を行う。コマンドが得られないので、結果として操作が効かない。

・ 論理デバイスの要件として記載しているとおり、コントローラの操作情報は、「A ボタン」「左ボタン」のような生の情報ではなく、コンフィグを通してコマンド化して扱う。
  - ➢ これにより、操作方法の変更に柔軟に対応する。
  - ➢ 操作方法のユーザーカスタマイズにもそのまま対応できる。

・ 「入力デバイス監視スレッド」は、（メインスレッドのフレームレートや処理落ちと無関係に）常時 60fps（もしくはそれ以上）の頻度で物理デバイスの情報を収集する。
  - ➢ 情報収集のタイミングは、（60fps の場合の）フリップのタイミングに合わせる。
    - プレイヤーが画面を見て最速で反応することを考慮し、念のため、フリップ後、0.5/60 秒ぐらいタイミングをずらして収集する。
    - 情報収集以外のタイミングでは、スレッドは極力「スリープ」状態で待機する。
    - ビジーウェイトでポーリングしたりせず、「条件変数」などの「モニター」の仕組みを活用し、できるだけスリープでウェイトとし、フリップタイミングの通知を受けて処理をウェイクアップさせる。
    - 条件変数を使用した最適な待ち受けとウェイクアップ手法は、別紙の「「サービス」によるマルチスレッドの効率化」を参照。
  - ➢ 「入力デバイス監視スレッド」は常駐スレッドとし、メインスレッドより高めのスレッド優先度を設定する。

・ メインループからデバイスマネージャに「現在のフレームの入力状態を確定」という通知が来たら、各論理デバイスは、「現在のフレーム用のコマンドリスト」に情報を移し替える。
  - ➢ メインループよりも高頻度でコマンドを収集していることもあり、メインループから要求があったタイミングでは、多数のコマンドが記録されている。
  - ➢ 「押し続け」のようなコマンドは、後着優先として、一つのコマンドにまとめる。
  - ➢ 論理デバイス内のコマンド記録はダブルバッファで管理し、メインループからの要求に瞬間的に対応できるようにする。
    - ただし、コマンドを書き込んでいる最中に要求があると、それが終わるまで、処理をブロックする可能性がある。

コマンドの記録とダブルバッファ管理のイメージ：

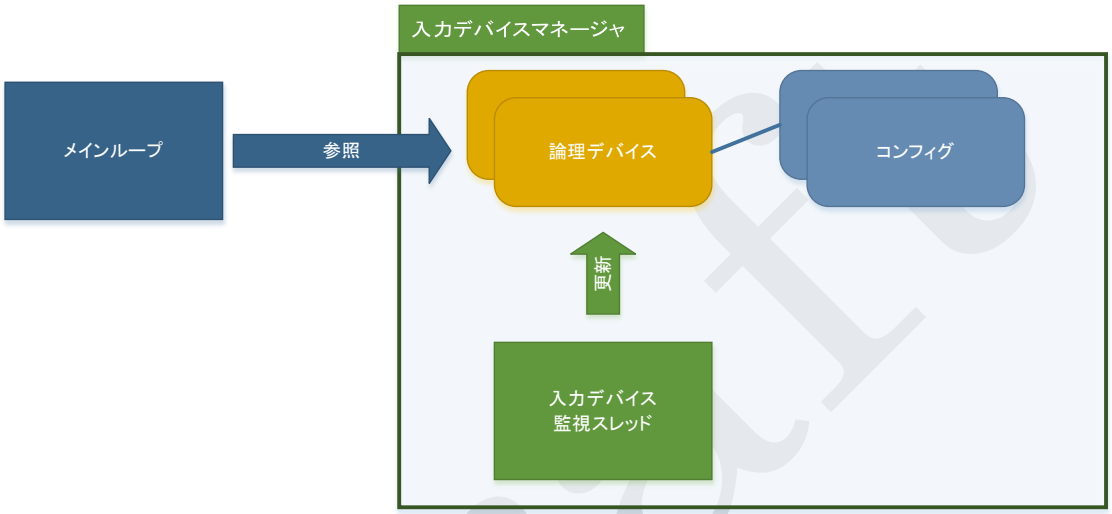


■ 仕様概要

▼ システム構成図

要件に基づくシステム構成図を示す。

入力デバイスマネージャのシステム構成図：



■■以上■■

■ 索引

索引項目が見つかりません。

反応性と安全性を考慮した入力デバイス管理

以　上