Space Vector Refact 制作概説

明治大学 理工学部 情報科学科 4年 林 友貴

目次

はじめに
制作の目的

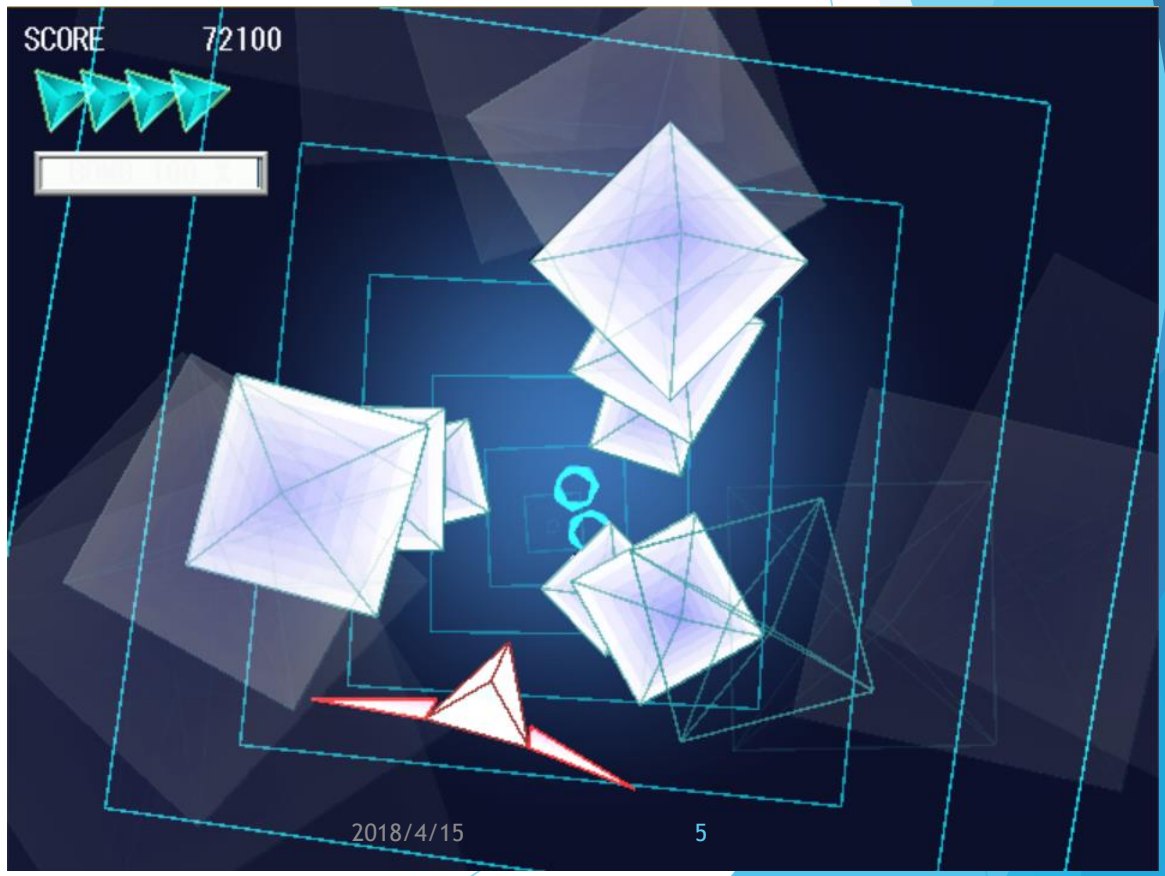
- Space Vector Refact
  - 大学一年次に作成したゲーム「Space Vector」(https://github.com/t-hayashi00/SpaceVector)を
    オブジェクト指向的にリファクタリング、改良した。
  - ※Space Vectorはファイル名こそcppだが
    中身は殆どCとして書かれている。

はじめに
意識したこと

- 他人(2ヶ月後の自分)が見たとき理解できるコードで書く
  - 関数や変数の命名は自明なものに
- オブジェクト指向的に作る
  - 元のコードからクラスとして分離できそうな部分を探す
  - 例によって状況に応じてデザインパターンを適用する

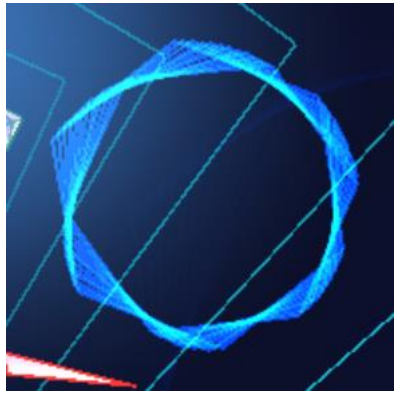
ゲーム内容
タイトル『Space Vector Refact』

- ジャンル
  - 3Dシューティングゲーム
- 特徴
  - 一点透視図法を用いた
    擬似3Dシューティングゲーム。
  - 画面奥から迫りくるブロックを
    破壊しながら長時間生存する。



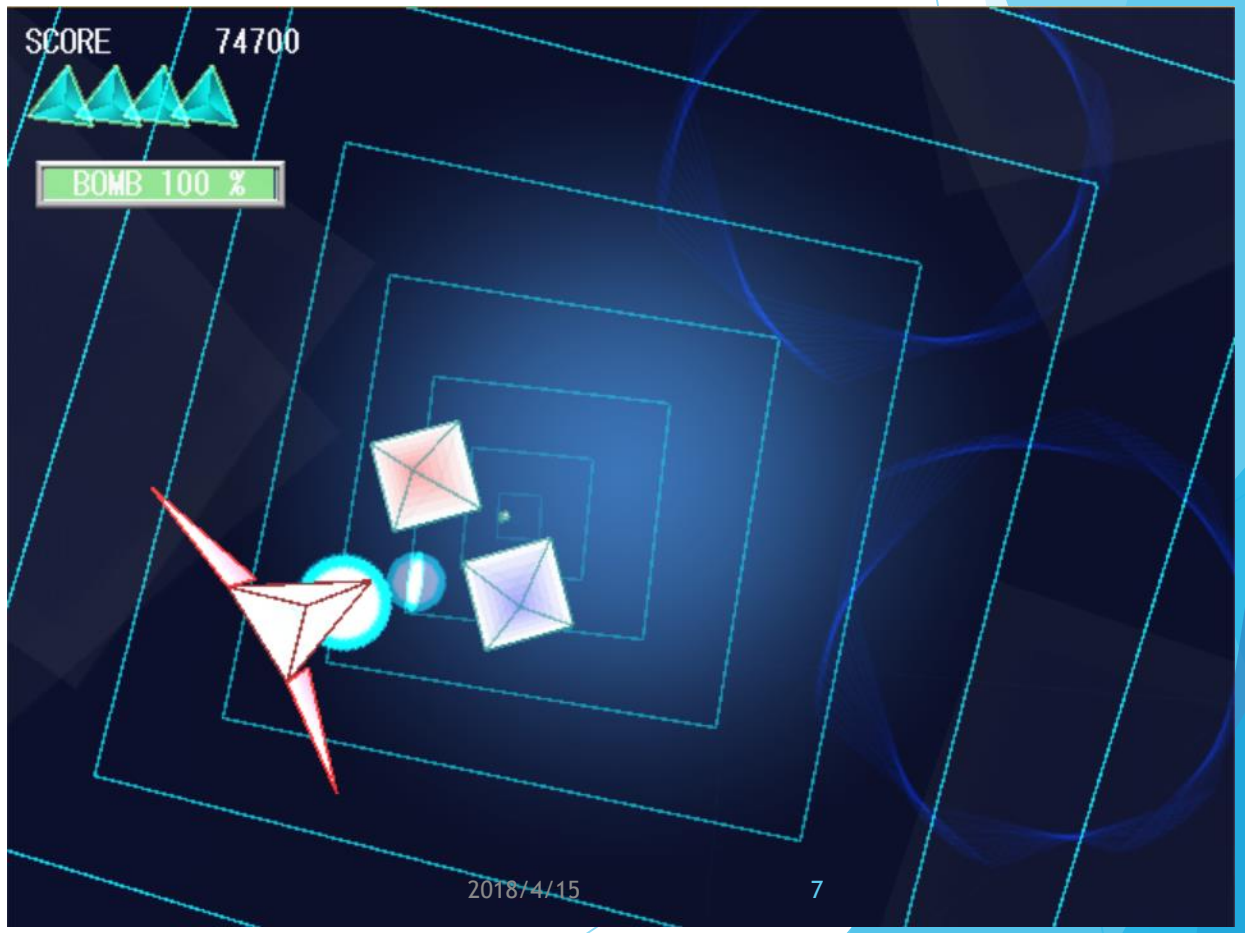
ゲーム内容
タイトル『Space Vector Refact』

- 特徴
  - 『ハイクオリティなドット絵や3Dモデルが無くたって
    見た目がすごいゲームは作れるんだ！』 (あるに越したことはない)
    という思いから生まれたゲーム。
  - 右図のリングなどは完全にDXライブラリ内の
    図形描画関数のみで描かれている。

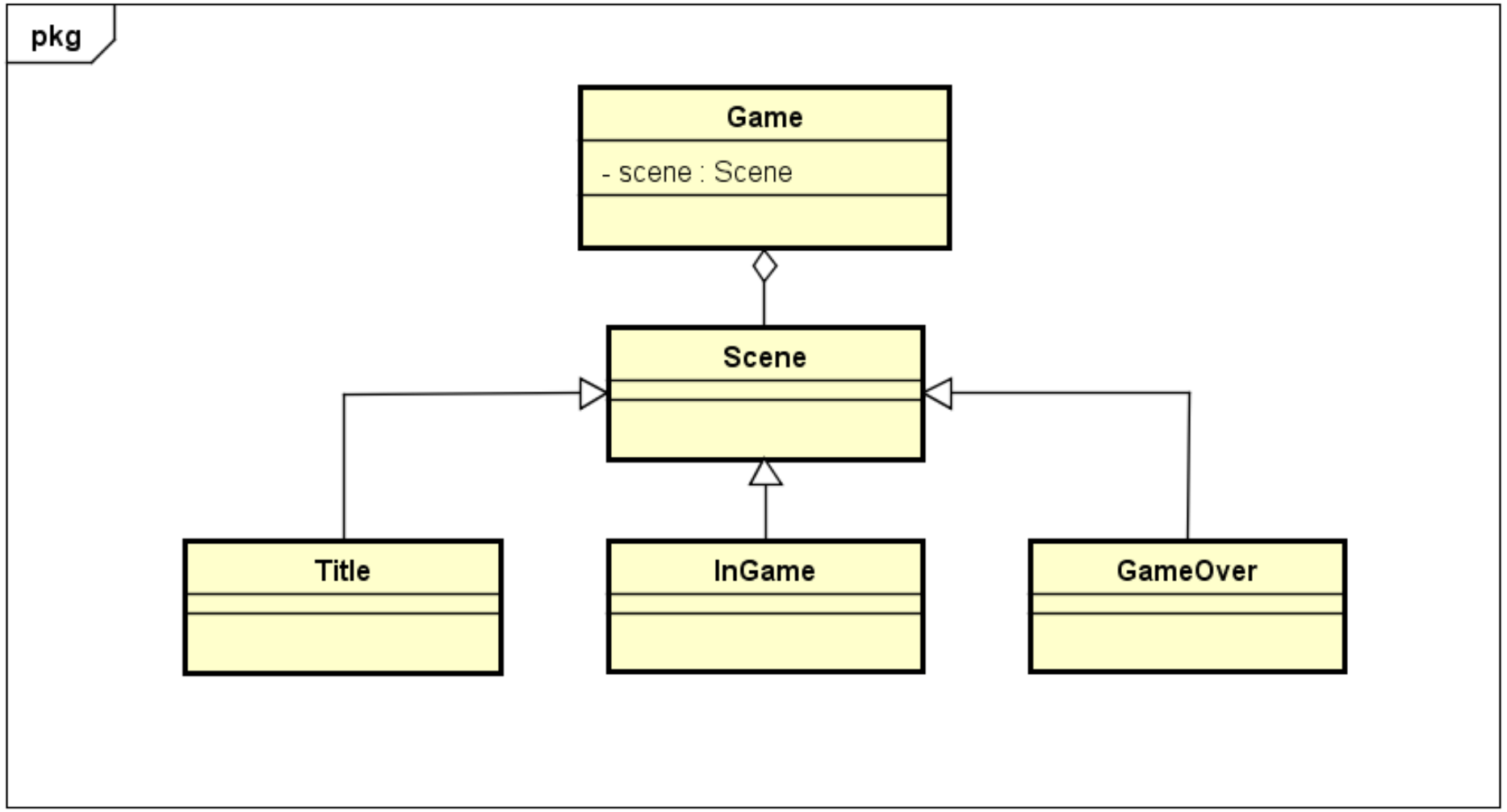


ゲーム内容
タイトル『Space Vector Refact』

- 開発言語・環境
  - C++, DXライブラリ
- 動作環境
  - Windows7, Windows8.1



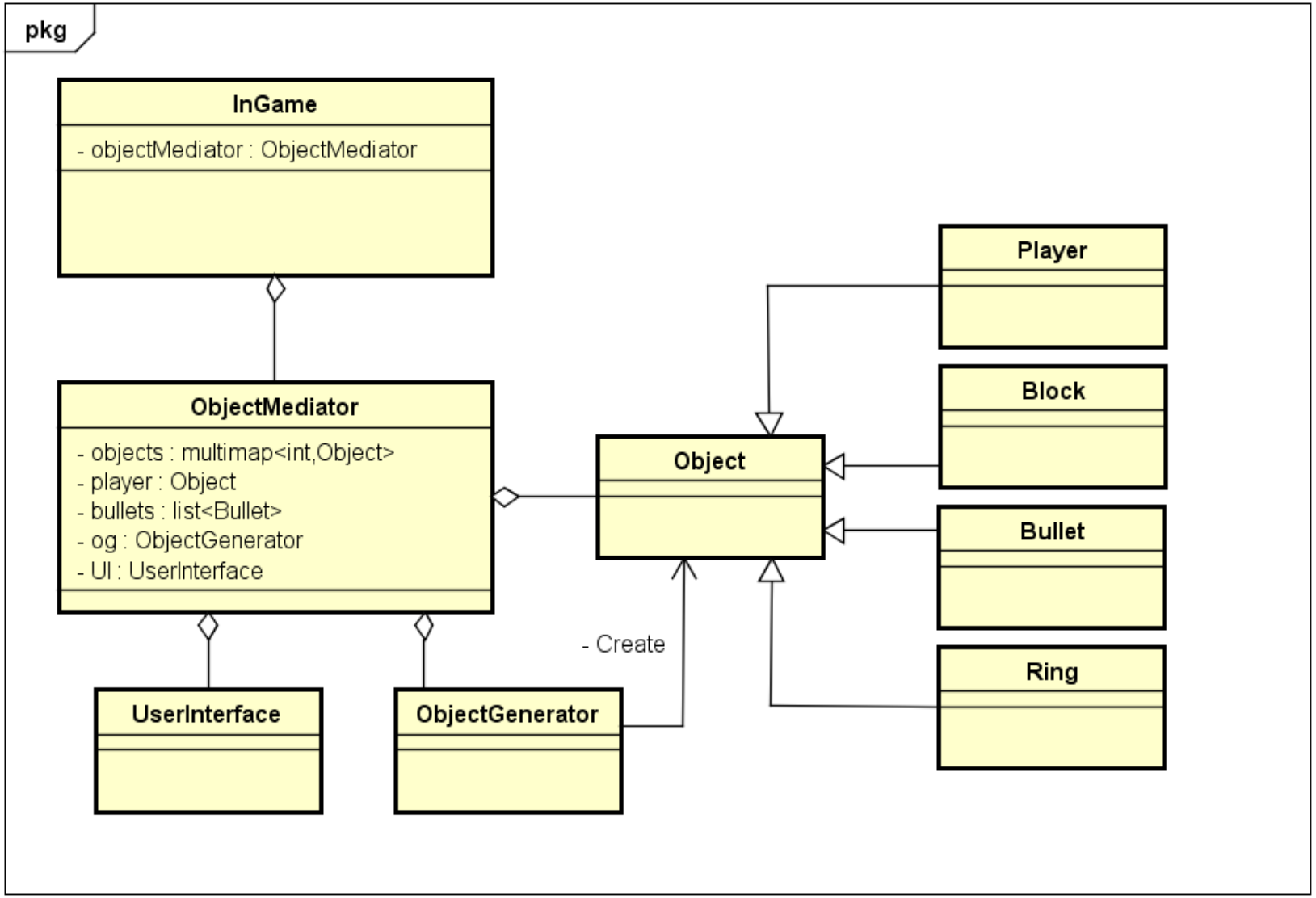
全体設計 クラス図　ゲーム全体



全体設計
Scene

- Scene クラス
  - ゲームのシーンとその遷移を管理するクラスの基底クラス
  - Title, InGame, GameOverのように各シーンごとに実装
  - Stateパターンを使用
  - Sceneオブジェクトを所持しているクラスは各Sceneオブジェクトのシーン遷移の知らせを受けてオブジェクトを付け替える
  - このパターンによりswitch文を使った冗長なコードを書く必要が無くなり、ソースコードの見やすさが向上

詳細設計 クラス図　ゲーム部分(InGameシーン)

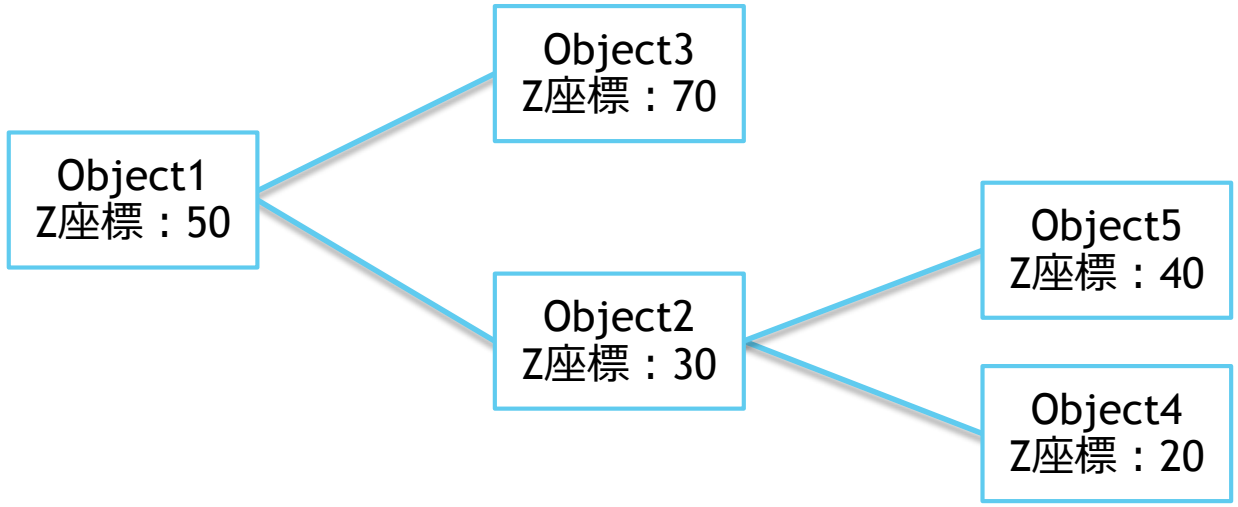


詳細設計
ObjectMediator

- ObjectMediator クラス
  - 自機(Player)と弾(Bullet),敵(Block, Ring)などの当たり判定などを管理する。
  - 描画順の管理もここで行う。
  - メンバ
    - multimap<Z座標, Object> objects (オブジェクトのマップ)
    - list<Bullet> bullets (弾のリスト)
    - Player (自機)

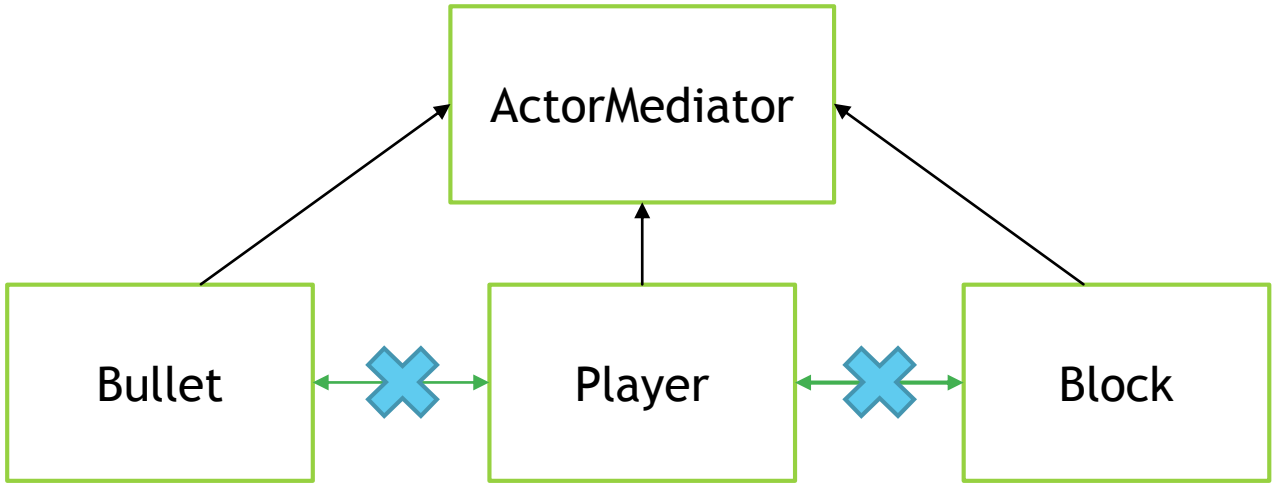
詳細設計
ObjectMediator

- 描画順のソート
  - multimapを用いてZ座標をkey,Objectをvalueにして管理。
  - multimap(TreeMap)のkeyの値で要素のソートが自動で行われるという特徴を用いて描画順を制御。
  - イテレータを使うと昇順で要素が取り出される。



詳細設計
ObjectMediator

- Mediatorパターン
  - Player, Block, Ring, Bulletの管理を一手に引き受けることで上記のクラス間の結合度を下げている。
  - コードの修正が最小限に



詳細設計
Object

- Object クラス
  - 自機(Player)や敵(Block)などのキャラクターの基底クラス。
  - 各オブジェクトの振る舞いはそれぞれの派生クラスで定義する。
  - ObjectMediatorにて一括で管理を行う。

詳細設計
ObjectGenerator

- ObjectGenerator クラス
  - Block, Ringのインスタンスの生成を管理するクラス。
  - 上記のオブジェクトの出現パターンをここで定義する。
  - ObjectMediatorのmultimapの参照を受け取り
    そこに生成したインスタンスを追加していく。

詳細設計

module.h

- module.h
  - 音の再生や画面効果など、ゲーム全体で必要になる機能を関数としてまとめたヘッダファイル。
- Facade パターン
  - 複雑な操作を隠蔽し、必要な機能のみを提供するパターン。

所感

- 制作期間は二週間ほど。オリジナル版の制作期間は一カ月。
- 過去の自分のコードを読み解くのに一番苦労した。
- Object* player = new Player()などとした場合、Objectのデストラクタを仮想関数にしなければPlayerのデストラクタが呼ばれないなど、C++独特の仕様に悩まされる場面が多々あった。
- HaxeやJavaで利用していたデザインパターンがC++でもスムーズに実装することができたのは良かった。

その他

- Githubアカウント
  https://github.com/t-hayashi00
- Space Vector Refact リポジトリ
  https://github.com/t-hayashi00/SpaceVectorRefact