トリニティ・トライアル制作概説

明治大学 理工学部 情報科学科 4年 林 友貴

目次

1. はじめに
2. ゲーム内容
3. 全体設計
4. 詳細設計
5. その他工夫点
6. 所感

はじめに
制作の目的

- ソフトウェア工学研究室 所属

  ソフトウェア開発における効率の良い開発手法の研究や
  それらを支援するシステムの設計開発を行う。

- 三年後期のゼミでデザインパターンなどのオブジェクト指向、
  リファクタリングの方法について学習したので、

ゲーム制作で実践

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
- まずは愚直に実装
  - 必要に応じてリファクタリング
- オブジェクト指向的に作る
  - 状況に応じてデザインパターンを適用
  - クラス間は疎結合に
  - 同じようなコードを繰り返し書かない

ゲーム内容
タイトル『トリニティ・トライアル』

- ジャンル
  - 横スクロールアクションゲーム
- 特徴
  - 自機を三体まで増やせる
  - 自機の操作は自由に切り替えられる
  - 全滅しなければOK

ゲーム内容
タイトル『トリニティ・トライアル』

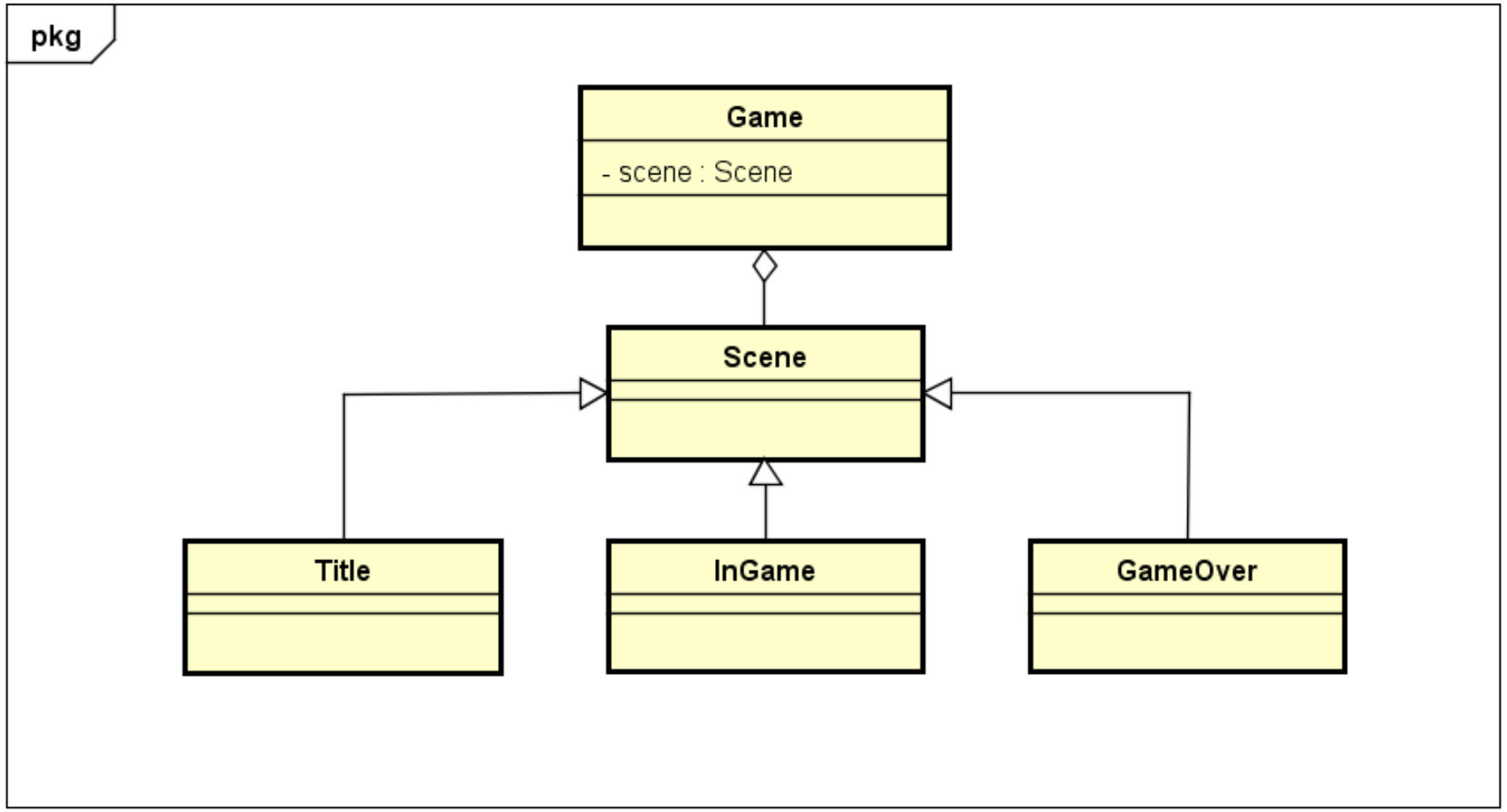
- アクション
  - 移動・ジャンプ
  - 操作する自機の切り替え
  - 仲間をぶん投げる
  - 自機を増やす
  - 仲間の集合・解散

俺を踏み台にしたり
俺の屍を超えていったりするゲーム

ゲーム内容
タイトル『トリニティ・トライアル』

- 開発言語・環境
  - Haxe, OpenFL
- 動作環境
  - Google Chrome, FireFoxなどのhtml5に対応したブラウザ

全体設計 クラス図　ゲーム全体



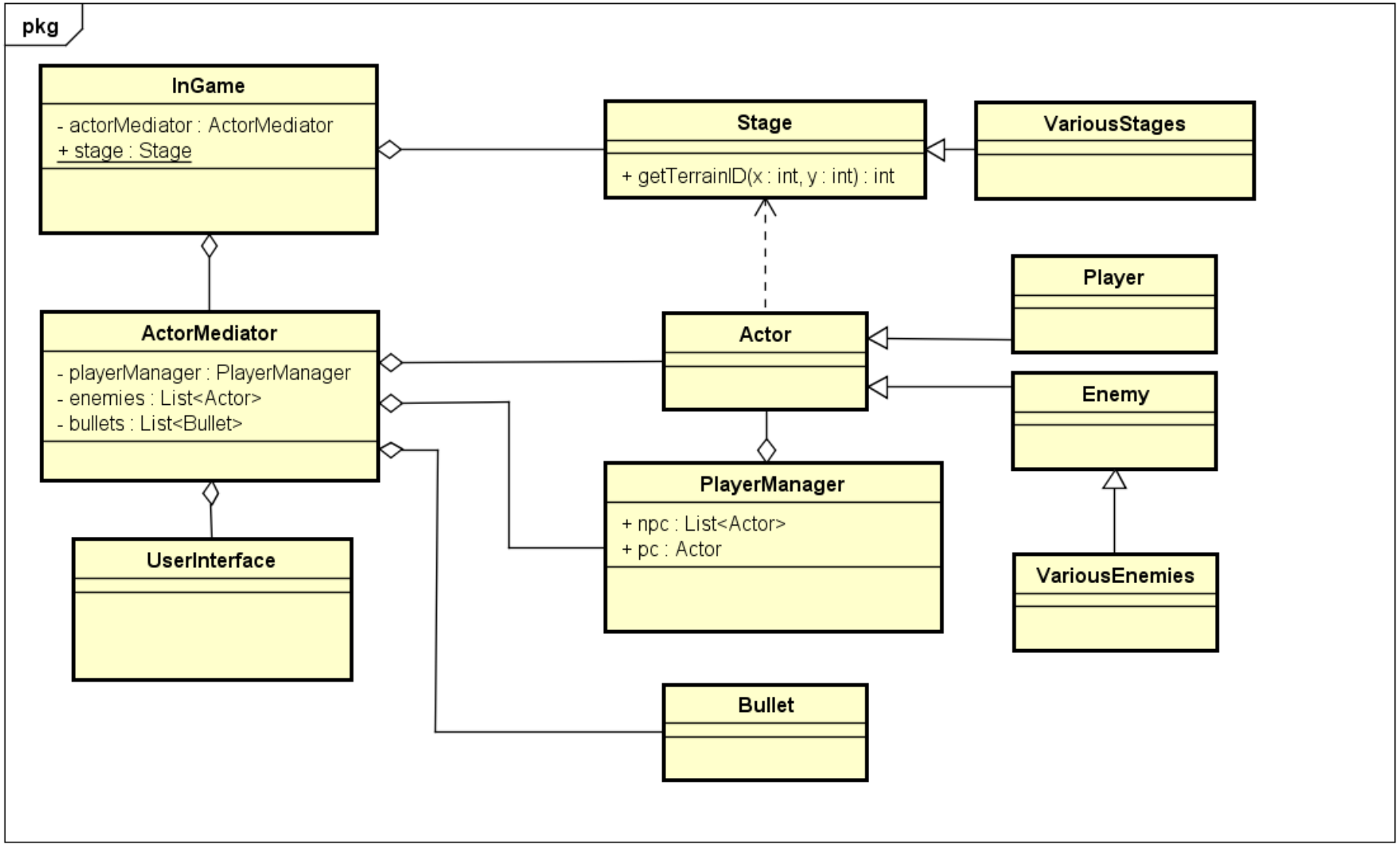
全体設計
Scene クラス

- 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

全体設計
InGame クラス

- InGame クラス
  - ゲームの本体部分とも言えるシーン
  - 次ページ以降からその設計を示す。

全体設計 クラス図　ゲーム部分

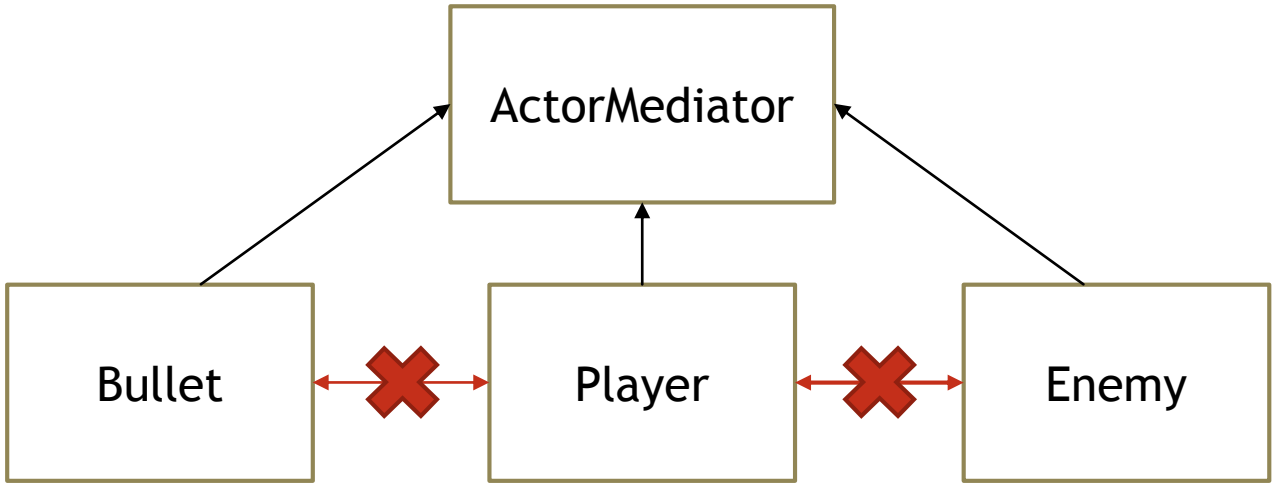


全体設計
ActorMediator クラス

- ActorMediator クラス
  - 自機(Player)と敵(Enemy)や敵弾(Bullet)の当たり判定などを管理する。
  - メンバ
    - List<Actor> enemies (敵をまとめたリスト)
    - List<Bullet> bullets (敵弾をまとめたリスト)
    - PlayerManager (自機の管理とその操作を行う)
      - Actor pc (操作する自機)
      - List<Actor> npc (それ以外の自機)

全体設計
ActorMediator クラス

- Mediatorパターン
  - Player, Enemy, 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



全体設計
Actor クラス

- Actor クラス
  - 自機(Player)や敵(Enemy)などのキャラクターの基底クラス。
  - 地形判定や重力による移動など
    キャラクターの基本的な挙動はここで定義されている。
- Enemy クラス
  - 敵(Enemy) のスーパークラス。
  - このクラスから派生して個々の挙動をする敵を実装していく。

全体設計
Bullet クラス

- Bullet クラス
  - 敵弾を表す。
  - インスタンス生成時に速度や威力など各種パラメータを設定することでその通りに動く。
  - 敵弾の生成はBulletGeneratorクラスが管理する。後述。

全体設計
Stage クラス

- Stage クラス
  - 地形や敵と自機の初期位置の情報を持っている
  - 地形の情報に関してはグローバルアクセスが可能

詳細設計
Factory パターン

- Factoryパターン
  - インスタンスの生成を別のクラスに管理させるパターン。
  - StageやEnemyなど
    サブクラスが多数存在するクラスに対して適用することで
    インスタンスの生成管理の手間を軽減している。
  - 例)　EnemyFactory
    StageFactory
    AnimationFactory

詳細設計
BulletGenerator クラス

- BulletGenerator クラス
  - Bulletのインスタンスの生成を管理するクラス。
  - 自機狙い弾、Nway弾など弾の出し方をここで定義する。
  - 他のクラスとBulletのリストの参照を共有して
    そこにBulletのインスタンスを生成して追加する。
  - Factoryパターンに近い？

詳細設計
Spritesheet クラス

- Spritesheet クラス
  - Actorのグラフィック管理するクラスの基底クラス。
  - 元々はグラフィックの処理もActorの各サブクラスで行っていたが肥大化したため分離して作成。
  - 対象のActorの参照をメンバに持ち、そのActorの状態に応じたグラフィックを表示する。

詳細設計
Module クラス

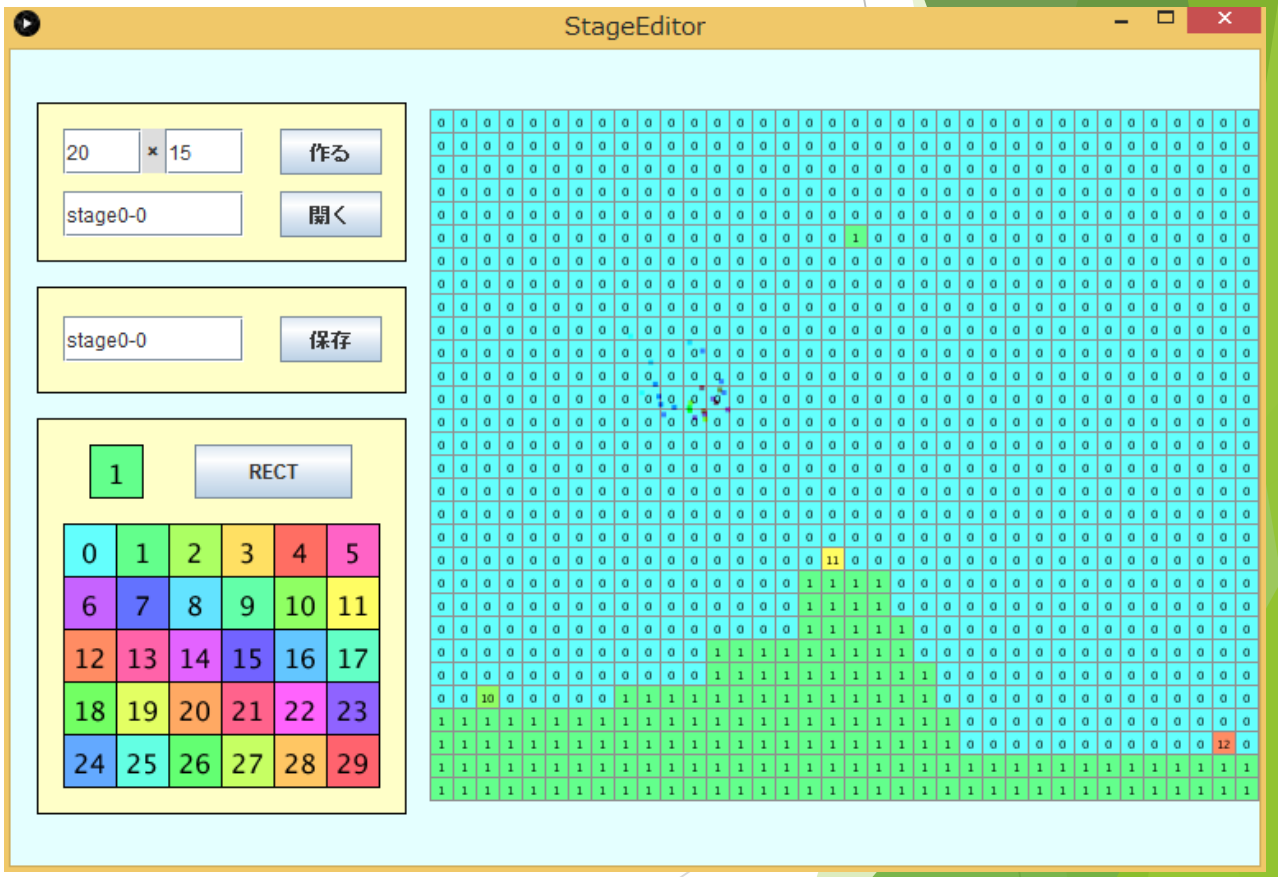
- Module クラス
  - キー入力など、ゲーム全体で必要になる機能を
    関数としてまとめたクラス。
- Facade パターン
  - 複雑な操作を隠蔽し、必要な機能のみを提供するパターン。
  - キー入力の処理のアレコレは Module内の private関数で完結させ、
    Moduleクラスからは「キーが押されたかの判定」などの
    必要な機能のみを関数として外部に提供している。
  - 先述したActorMediatorも見方によってはFacade パターン

詳細設計
Sequencer クラス

- Sequencer クラス
  - キューに関数(で包んだ文)を追加すると
    順々に実行してくれる。
  - UnityやLuaのコルーチンに近い機能を実現する。
  - 敵などの挙動を非常に簡単に設定できるようになった。
    - 例)　go_straight();
      wait(10);
      attack(1);

その他工夫点
ステージエディタ

- ステージエディタを自作
  - ステージをグラフィカルに作成できるツール
  - 二元配列に格納しやすいようにcsv形式で入出力



所感

- 現在ステージ１まで完成
- ゲームの土台部分は1カ月ほどで完成した
- ソフトウェア工学の『開発を効率よく行う』という目的は達成されているのではないか

その他

- Githubアカウント
  https://github.com/t-hayashi00
- トリニティ・トライアル リポジトリ
  https://github.com/t-hayashi00/TrinityTrial
- ステージエディタ リポジトリ
  https://github.com/t-hayashi00/StageEditor