

2020-後学期

# 令和2年度 プログラミング演習

## グループプログラミング レポート

### 「タワーディフェンスゲーム」

学科	総合情報学専攻
クラス	J1
グループ番号	1
1910002	青木 辰磨
1910097	上田 達也
1910110	遠藤 誉士

# 1 プログラムの概要

## 1.1 目的

電通大の4Kマップを使ったタワーディフェンスゲームの作成

## 1.2 機能

ゲームは「タイトル・ゲーム・ポーズ・リザルト」の4画面から構成される。

### 1.2.1 タイトル画面

- ゲーム画面への移行とゲームの終了が出来る。

### 1.2.2 ゲーム画面

- ポーズ画面への移行が出来る。
- 電通大内の主要な建物の内、防衛拠点となるタワーをゲーム毎にランダムに2つ選ぶ。
- タワーを陥落させるために電通大へと攻めてくる4種類のエネミーと、エネミーを撃退するための3種類のタレットがある。
- エネミーは、ウェーブ毎にランダムに選出された3つのスポーン地点から出現し、防衛拠点へと侵攻、攻撃する。
- エネミーの数は、ウェーブ毎に増加する。
- エネミーを撃退する為に、タレットを購入し、指定されたいくつかの地点に配置できる。
- タレットは、レベルに応じた金額を支払うことで、「攻撃力・攻撃範囲・攻撃速度」のレベルを上げて強化することが出来る。所持金は、エネミーを撃破することで増やすことが出来る。
- 購入したタレットは、必要に応じて売却することが出来る。売却値はタレットのレベルによって変わる。
- エネミーがタワーを攻撃し、2つのタワーが陥落したらゲームオーバー。一定のウェーブ数をしおぎ切ればゲームクリアとなる。
- エネミーの撃破状況とゲームクリア時に残ったタワーの数を参照してスコアを決定する。

### 1.2.3 ポーズシーン

- ゲームの進行を一時停止する。再開時には、ポーズした時点からゲーム画面に移行する。
- ゲームの再開(ゲーム画面への移行)とゲームの終了(タイトル画面への移行)が出来る。

#### 1.2.4 リザルトシーン

- ゲームオーバーとゲームクリアの2種類の画面があり、ゲームのスコアを表示する。
- タイトル画面への移行が出来る。

### 1.3 作業分担

プログラムをMVCモデルに従って構成したため、作業分担も主に「Model・View・Controller」の3つに分けて行った。それとは別に、タワー・エネミー・タレットといった「ゲームオブジェクト」のプログラムをそれぞれが担当した。

以下が、それぞれが担当したプログラムの詳細である。

- 上田

- ゲーム画面の Model
  - タレットが配置可能な場所
  - エネミーの出現
  - マップ製作(タワー・エネミー侵入場所、グラフの頂点の座標を定義)

- 遠藤

- ゲーム画面の View
  - ゲームシステム
  - ゲームオブジェクトの各種エネミー
  - ゲームオブジェクトのタワー
  - スタート、ポーズ、タイトル画面の Model・View・Controller

- 青木

- ゲーム画面の Controller
  - ゲームオブジェクトの各種タレット
  - エネミーのサーチ(索敵)
  - ゲームサウンド
  - ゲームオブジェクトと各シーンの画像探し

始めにゲームシステムの構築を行い、ゲームを作る上で活用する機能を実装した。ゲーム本体のプログラムは、始めにControllerと各ゲームオブジェクトの基盤となるクラスを実装し、実装が出来たものをViewとModelに追加していく。ゲームとして動作するようになった後は、基盤となるクラスを継承した各ゲームオブジェクトを複数種作成し、ゲーム性の拡張とゲームバランスの調整を行った。最後に、スタート、ポーズ、リザルト画面の実装とゲームサウンドの実装を行い、完成へと至った。

プログラムの統合は、Githubを用いて行った。これにより、「誰がどのプログラムを書いたのか」「どこのプログラムを変更したのか」などの確認が非常にスムーズに出来た。また、万が一途中でプログラムが動かなくなってしまったとしても、直前の状態へと戻すことが出来る為、安心してコードを共有することが出来た。

(文責: 青木)

## 2 設計方針

まず、プログラム全体のシーン遷移などを実現するための設計について述べる。

大まかにプログラムはシステム部とシーン部に分けられる。

システム部はウインドウ作成やシーン管理などゲーム内容に関係なく行われる処理を行うクラスと、ベクトルの計算など様々な場面で汎用的に用いるクラスからなるシーン部はタイトル画面やゲーム画面、ポーズ画面など、場面ごとの動作を記述するクラスからなり、各シーンはシステム部の Scene, SceneModel, SceneView, SceneController クラスを継承して作る。

ゲーム内に登場するオブジェクトは基本的にすべてシステム部のクラスの一つである GameObject を継承して作る。

シーン部は MVC パターンを用いて構成しており、Model はデザインパターンの一つである Composite パターンを利用して作られた GameObject の木を持つ。ゲーム内オブジェクトは再帰的な木構造を持つため、これによってゲームに適した粒度にクラスを分割することが可能になり、分担を行うことができた。

システム部とゲーム画面のシーンの関係を示すクラス図を図 1 に、ゲームオブジェクトのヒエラルキー図を図 2 に示す。

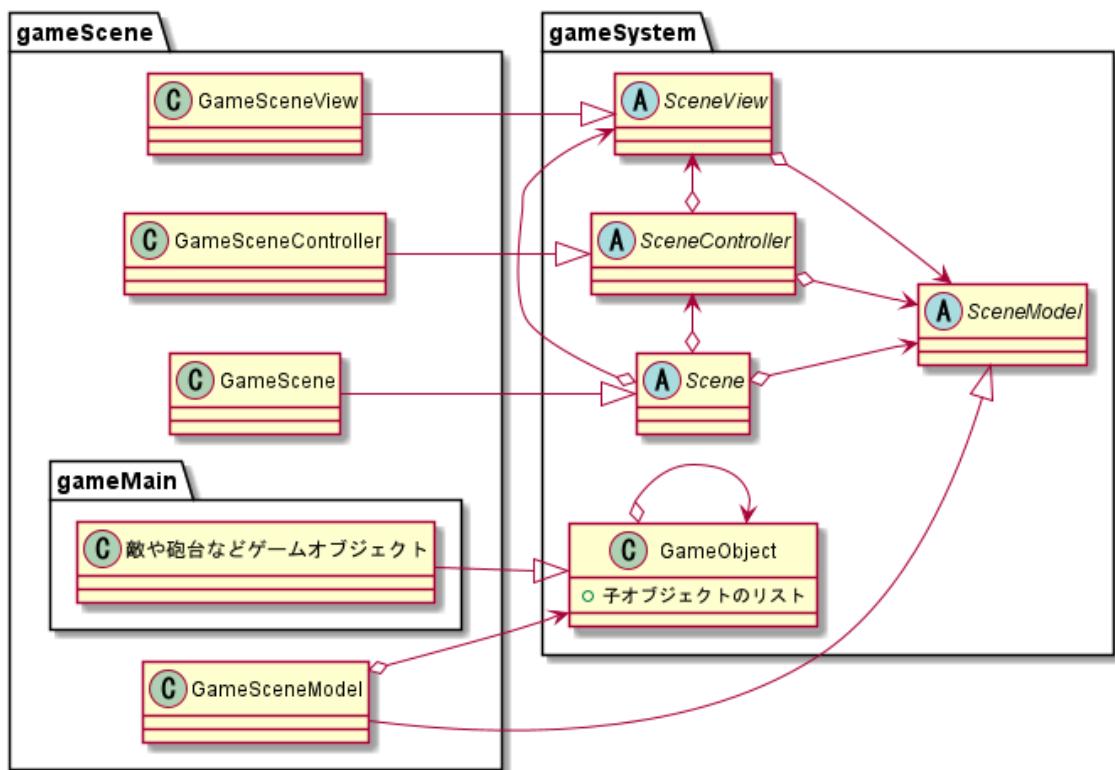


図 1: システム部クラス図

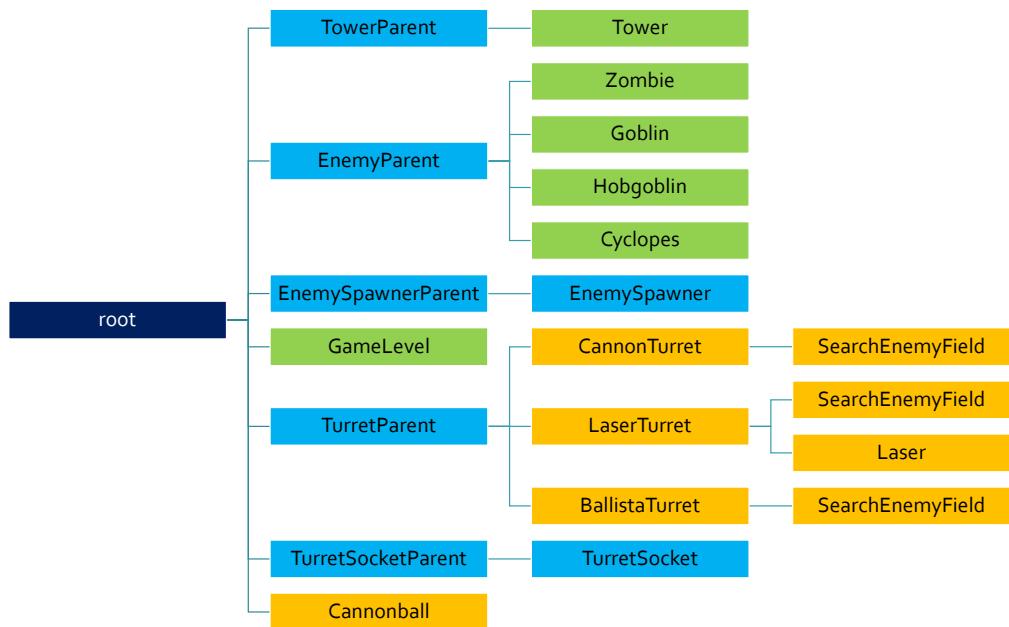


図 2: ヒエラルキー図例

## 2.1 ゲーム画面

ゲームを表現するために必要なオブジェクトは、大まかには以下の物がある。

- タワー
- 敵
  - 敵が移動するための道情報を持つマップ。
- タレット
  - 各タレットから発射される砲弾や矢など

これらと、ルールの大まかな実装方法について述べる。

### 2.1.1 タワー

タワーを表すクラス `Tower` は、`TowerArranger` クラスによって選出、配置される。また、`Tower` クラスは攻撃を受けることが可能であることを示すインターフェース `IDamageApplicable` を実装しており、`applyDamage` メソッドが実行されると攻撃を受ける。耐久値が無くなると陥落し、陥落したことを受け取りたいクラスであることを示す `TowerFallListener` の `onTowerFall` メソッドが実行される。

### 2.1.2 敵

敵は `BaseEnemy` クラスを継承して作られ、ゲーム内では `EnemySpawner` オブジェクトから出現する。`EnemySpawnerManager` はこれを管理するクラスであり、インスタンスの生成、敵の種類や出現間隔、ス

ポナー自体の起動時間などの情報の `EnemySpawner` への入力等を行う。管理のアルゴリズムは `Strategy` パターンにより柔軟に変えることが変えられるようにした。

敵もタワーと同じく攻撃可能であることを示す `IDamageApplicable` を実装しており、攻撃されると耐久値が減り、無くなると敵が死亡したことを受け取りたいクラスであることを示す `EnemyDieListener` の `onEnemyDead` メソッドが実行される。

敵自体の行動アルゴリズムも `Strategy` パターンによって変更することが可能になっている。実際に実装されているアルゴリズムは

- 襲撃対象をランダムに選択する。
- 現在の地点から対象までの最短経路を探索する
- これに沿って移動し、対象を攻撃する
- 対象が陥落した場合は、別の対象が残っている限り最初から同じことを繰り返す

というものである。敵の移動、経路探索等は地図情報を持つオブジェクト `GameLevel` に任せられている。地図情報は `Graph` クラスが持つ。データ構造としては隣接リスト表現でグラフを持っており、頂点情報、辺情報が格納される。グラフは `GraphBuilder` クラスによって外部ファイルから情報を得て生成される。

### 2.1.3 タレット

タレットは `BaseTurret` クラスを継承して作られる。配置可能な場所を示すオブジェクトである `TurretSocket` がクリックされたときに、購入可能な状態であれば配置される。

配置されたタレット自体を選択することが可能であり、選択されている間は能力の強化、売却ができる。

### 2.1.4 ルール

`GameSceneModel` クラスはゲームの状態、データを管理する。ゲームの主な状態には「次のウェーブを待っている状態」「ウェーブ中」、「ゲームオーバー」、「ゲームクリア」が存在し、時間経過やゲーム内の状態によって遷移する。また同クラスは `TowerFallListener`, `EnemyDieListener` などのインターフェースを実装し、敵の撃破やタワーの陥落などを監視し、状態遷移やスコアの計算などに使用する。ゲームオーバー、もしくはクリアの条件を満たしたら一定時間ののち、リザルト画面に遷移する。また、ポーズ入力があった場合、ポーズ画面に遷移する。

(文責: 遠藤)

## 3 プログラムの説明

### 3.1 遠藤担当分主要クラス

#### 3.1.1 MainFrame

##### 3.1.1.1 仕様

`static void main()` の存在するクラス。`JFrame` を継承しており、ゲームのメインウインドウとして機能する。また、インターフェース `SceneChangeListener` を実装しており、シーン遷移の制御を行う。

### 3.1.1.2 フィールド

---

```
private JPanel mainPanel; // 基本的に全てのSwingの要素はこのパネルの下に置く  
private CardLayout layout; // カードレイアウトでシーンを切り替える  
private Stack<Scene> sceneStack; // シーンを保存しておくスタック
```

---

### 3.1.1.3 メソッド

#### 3.1.1.3.1 createWindows()

Swing の機能で新しいウインドウを生成する。

#### 3.1.1.3.2 void sceneChanged(Scene scene, boolean stackClearFlag, Intent intent), sceneChange(scene, Intent intent)

SceneChangeListener インタフェースのメソッド。

stackClearFlag が True ならスタックにあるシーンをすべてクリアする。False なら一番上にあるシーンをポップして新しいシーンに遷移する。False がデフォルト。

#### 3.1.1.3.3 void scenePopped(Intent intent)

SceneChangeListener インタフェースのメソッド。

一番上にあるシーンをポップする。ポーズシーンから戻るときなどに使う。

#### 3.1.1.3.4 void scenePushed(Scene scene, Intent intent)

SceneChangeListener インタフェースのメソッド。

一番上に新しいシーンをpussh する。ポーズ画面への遷移などに使う。

## 3.1.2 Scene

### 3.1.2.1 仕様

MVC 各インスタンスと SceneChangeListener を持ち、一つのシーンとして機能する。コンストラクタ以外に特にメソッドはない。

### 3.1.2.2 フィールド

---

```
protected SceneModel model;  
protected SceneView view;  
protected SceneController controller;  
protected SceneChangeListener sceneChangeListener;
```

---

## 3.1.3 SceneModel

### 3.1.3.1 仕様

MVC の Model を表す基底クラス。

### 3.1.3.2 フィールド

---

```
protected SceneChangeListener sceneChangeListener;  
protected GameObject rootGameObject;
```

---

### 3.1.3.3 メソッド

抽象クラスのため、ゲームオブジェクトの root を機能させる処理以外具体的な実装は存在しない。

#### 3.1.3.3.1 start(Intent intent), stop(), pause(), unpause()

シーン開始時、終了時、ポーズ時、ポーズ解除時に実行されるメソッド。start は前のシーンから情報を受け取ることができる。

#### 3.1.3.3.2 void addGameObject(GameObject gameObject)

root のゲームオブジェクトの子オブジェクトを追加する処理。

## 3.1.4 SceneView

### 3.1.4.1 仕様

JPanel を継承している、MVC の V を表す基底クラス

### 3.1.4.2 フィールド

---

```
protected SceneModel model;
public Camera camera; // カメラを表す
GameObject。描画にこのオブジェクトの情報を用いることもできるが、用いないこともできる。
```

---

### 3.1.4.3 メソッド

#### 3.1.4.3.1 start(), stop(), pause(), unpause()

シーン開始時、終了時、ポーズ時、ポーズ解除時に実行されるメソッド。

#### 3.1.4.3.2 draw(Graphics g)

描画を行うメソッド。paintComponent 内で呼び出される。GameObject のヒエラルキーを DFS して描画できるものだけ取り出し、depth でソートして描画を各オブジェクトにさせる。

#### 3.1.4.3.3 public Vector2 convertScreenPosToWorldPos(Vector2 screenPos), public Vector2 ConvertWorldPosToScreenPos(Vector2 worldPos)

カメラの情報から、スクリーン座標とゲーム内の座標（以下ワールド座標とする）を変換するメソッド。

## 3.1.5 SceneController

### 3.1.5.1 仕様

MVC の C を表す基底クラス。javax.swing.timer を用いて短い時間ごとに model への更新処理と view への描画処理を通知する。

### 3.1.5.2 フィールド

---

```
protected SceneModel model;
protected SceneView view;
protected javax.swing.Timer timer;

private long prevTime = 0; // フレーム時間計算用、前のフレームの時刻
```

---

### 3.1.5.3 メソッド

#### 3.1.5.3.1 start(Intent intent), stop(), pause(), unpause()

シーン開始時、終了時、ポーズ時、ポーズ解除時に実行されるメソッド。model,view に同様のメソッドを実装させる。

#### 3.1.5.3.2 void actionPerformed(ActionEvent e)

タイマーから一定時間ごとに呼び出され、model に更新処理を、view に描画処理を行わせる。

#### 3.1.5.3.3 IClickable findClickedObject(Vector2 position)

このクラスを継承したクラスで使う、クリックされているオブジェクトを探すメソッド。メソッドのないマーカーインターフェースである IClickable を実装している中で最も depth が浅いものを探す。

## 3.1.6 GameObject

### 3.1.6.1 仕様

- ゲーム内に存在するオブジェクトの基底クラスである

ゲーム内に存在するオブジェクトの基底クラスである

- ゲーム内での位置を表す座標を持つ

- 短い時間ごとに実行される処理、シーン開始時、ポーズ時、ポーズ開始時には各処理が再帰的に呼び出される

- オブジェクトが木構造に追加されたとき、削除されたときに処理が呼び出される。この処理が再帰的には呼び出されないのは、システム側がかかわらない状況だからである。

### 3.1.6.2 フィールド

---

```
protected GameObject root; // 木構造の根のGameObject
public GameObject parent; // 同じく親
public List<GameObject> children; // 同じく子

private boolean enabled; // オブジェクトが有効であることを示すフラグ。有効な時のみ処理が行われる。
public Vector2 position, offset; // オブジェクトの存在する座標の基準点とそこからずれる距離。
// 相対座標を用いて表現した方が適切な場合はoffsetを用いる。
public int depth; // 深さ。Z座標とも言い換えられる。

private List<GameObject> addObjectList, removeObjectList; // 子オブジェクトを追加・削除する
// 時のバッファ

public Collider collider; // 当たり判定
```

---

### 3.1.6.3 メソッド

#### 3.1.6.3.1 start(), \_start(), calc(float deltaTime), \_calc(float deltaTime), pause(), \_pause(), unpause(), \_unpause()

スタート時、毎フレーム（または十分に短いペース）、ポーズ時、ポーズ解除時に呼び出されるメソッド。アンダーバー付きのメソッドが内部的には呼び出され、その内部からアンダーバーなしのメソッドが呼び出される。アンダーバーなしのメソッドはここでは実装が存在しないが、オーバーライドされたときに実装される可能性がある。（必要でないなら実装しないことも可能）

### 3.1.6.3.2 onSummoned(), onDestroyed()

木構造に追加されたとき、削除されたときに呼び出されるメソッド。これもオーバーライドされたときに実装される可能性がある。

### 3.1.6.3.3 destroy()

このオブジェクトを木構造から削除するメソッド。再帰的に子オブジェクトにも適用される。

### 3.1.6.3.4 addChild(GameObject child), removeChild(GameObject child)

子オブジェクトを追加・削除するメソッド。実行した瞬間に処理を行うとデータ構造が破壊されるため、一時的に追加・削除するリストに入れられ、次の\_calc() 実行時に処理される。

### 3.1.6.3.5 getEnabled(), setEnabled(boolean enabled), onEnabled()

オブジェクトが有効であることを示す enabled フラグは private 指定されているため、このゲッター、セッターを通して操作する。有効になった時に onEnabled() が実行される。

### 3.1.6.3.6 findChild(Class<G> class\_), findChildFromChildren(Class<G> class\_)

子オブジェクトの中である派生クラスに当たるものを探す。find...From... という名前のものは直接の子だけでなく全ての子孫から再帰的に探す。該当するもののうち最初に見つけたもののみ返す。

### 3.1.6.3.7 findChildren(Class<G> class\_), findChildrenFromChildren(Class<G> class\_)

前項と同様に子オブジェクトの中である派生クラスに当たるものを探す。該当するものを全てリスト形式で返す。

例えば root.findChild(Camera.class) などとすることで root の直接の子に Camera クラスが存在すれば探すことができる。

### 3.1.6.3.8 clone()

オブジェクトを複製する際に実行するメソッド。

## 3.1.7 GameSceneView

### 3.1.7.1 仕様

GameSceneModel から情報を受け取り、防衛対象や次のウェーブまでの残り時間など、表示すべき情報を表示する

タレットを強化するための UI を定義し、表示する

配置するタレットを選択するための UI を定義し、表示する

### 3.1.7.2 フィールド

---

```
private Font font; // UI 表示に用いるフォント

private InformationPanel informationPanel; // 情報表示のためのパネル。
public UpgradeTurretPanel upgradeTurretPanel; // 強化対象のタレットとその情報、タレット強化
                                              のためのボタンを配置したパネル
public AddTurretPanel addTurretPanel; // 配置するタレットを選択するためのボタンを配置したパ
                                              ネル
private GameSceneModel gSceneModel; // モデルのインスタンス
```

---

いずれのパネルもオブザーバーパターンを用いて Model から情報の変化を受け取り、表示する情報を変更する。

#### 3.1.7.2.1 InfomationPanel

- 次のウェーブ開始までの秒数
- 現在のウェーブ数
- 所持金
- 防衛対象

を表示するパネル。GridBagLayout を使用している。

#### 3.1.7.2.2 UpgradeTurretPanel

- 現在強化対象になっているタレットの画像と名前
- 現在の強化対象の範囲レベル、連射速度レベル、威力レベルの表示
- 現在の強化対象の範囲強化、連射速度強化、威力強化ボタンの配置

を行うパネル。これも GridBagLayout を使用している

#### 3.1.7.2.3 AddTurretPanel

配置するタレットの種類を選択するためのボタンを配置するパネル。

### 3.1.7.3 メソッド

#### 3.1.7.3.1 update(Observable o, Object arg)

Java 標準の Observable クラスを継承したクラスからゲームのウェーブ数を受け取り、それによって BGM を変える。

## 3.1.8 BaseEnemy

### 3.1.8.1 仕様

このクラスを継承して敵クラスを作成する。基本的に敵としての行動のほとんどの処理は BaseEnemyActStrategy クラスに委譲する。

耐久値などのパラメータと行動パターンさえ渡せば簡単に敵の種類を増やせるようにしている。

### 3.1.8.2 フィールド

---

```
// 耐久、攻撃、スピード、ドロップ金額
protected double hp, attackPower, speed;
protected int dropValue;

protected BaseEnemyActStrategy mover; // Strategy パターンを使用して実装、敵の行動パターンを表すクラス
protected GameLevel level; // ゲームのマップデータのオブジェクト。経路探索に使う
protected ArrayList<Tower> targets; // 襲撃対象のリスト
protected Graph.Vertex initialVertex; // 最初に出現したマップ上の地点

protected ArrayList<EnemyDieListener> enemyDieListeners; // この敵が撃破されたときの通知を受け取るリスナーの一覧
```

---

### 3.1.8.3 主なメソッド

3.1.8.3.1 `getHp()`, `getAttackPower()`, `getSpeed()`, `getDropValue()`, `getDropScore()`, `getInitialVertex()`  
体力、攻撃力、スピード、撃破時の賞金、撃破時のスコア、初期地点のゲッター

3.1.8.3.2 `calc()`, `onSummoned()`, `onDestroyed()`

`GameObject` に存在するメソッドをオーバーライドして、行動パターンのクラスに委譲している。

3.1.8.3.3 `addEnemyDieListener(EnemyDieListener enemyDieListener)`, `removeEnemyDieListener(EnemyDieListener enemyDieListener)`

敵撃破時のイベントを受け取るリスナーの登録、解除

---

```
public double getHp() {}
public double getAttackPower() {}
public double getSpeed() {}
public int getDropValue() {}
@Override
public void calc(float deltaTime) {}
@Override
public void applyDamage(Damage d) {}
public void addEnemyDieListener(EnemyDieListener enemyDieListener) {}
public void removeEnemyDieListener(EnemyDieListener enemyDieListener) {}
public Graph.Vertex getInitialVertex() {}
```

---

## 3.1.9 Graph

### 3.1.9.1 仕様

マップのデータを表すデータ構造としての役割を持つクラス。重み付き無向グラフを隣接リスト表現で持ち、各頂点は座標や固有の識別番号などの情報を持つ。辺はどの識別番号の頂点からどの頂点へ行くのかという情報、距離を持つ。

### 3.1.9.2 フィールド

---

```
public ArrayList<Vertex> vertexes; // 頂点の情報のリスト
public ArrayList<ArrayList<Edge>> data; // 隣接リスト表現のマップデータ
```

---

### 3.1.9.3 メソッド

3.1.9.3.1 `addVertex(Vector2 position, int idx)`

座標と固有の番号を指定して頂点を追加する。

3.1.9.3.2 `addEdge(int from, int to)`

2 頂点の番号から辺を追加する。距離はそれぞれの頂点の座標から求める。

3.1.9.3.3 `newarestVertex(Vector2 position)`

ある座標に最も近い頂点を返す。

3.1.9.3.4 `Stack<Vertex> shortestPath(Vertex s, Vertex t)`

ある頂点からある頂点までの最短経路をたどるために必要な頂点をスタックに入れて返す。具体的な実装は二分ヒープを用いた Dijkstra 法。

(文責: 遠藤)

## 3.2 上田担当分主要クラス

### 3.2.1 GameSceneModel

#### 3.2.1.1 仕様

- Controller が操作を受け取り、Model に反映できるような機能を持つ
  - カメラの移動と拡縮
  - マウス操作
- ゲームの状態遷移を行う
  - ゲームクリア・ゲームオーバーのチェック
  - タイマーによる制御
  - 遷移時の処理

#### 3.2.1.2 フィールド

---

```
private final float cameraZoomSpeed = 0.3f; // カメラの拡縮スピード
private final float cameraZoomMax = 2.0f; // カメラ拡大の最大値
private final float cameraZoomMin = 0.2f; // カメラ縮小の最小値
private BackGround backGround; // 背景
private Camera camera;
private Vector2 cameraMoveDirection; // 正規化された、カメラが動くベクトルが入る変数
private double cameraSpeed; // カメラの移動速度に緩急をつけるための変数
private int cameraZoomOperation; // カメラの拡縮操作を
    Controller から受け取るための変数。範囲 [-1, 1] の整数が格納される。
private GameTimer nextWaveTimer, waveEndTimer, gameOverTimer, gameClearTimer;
// 次
    wave を待機するためのタイマー、wave 開始から終了までのタイマー、ゲームオーバーとゲームクリア遷移まで
public ObservableComponent<GameState> currentState; // 現在の GameState
public ObservableComponent<Integer> countdownNum, waveNum, num0fEnemies, balance,
    maxWaveNum, score;
// wave 開始前のカウントダウン値、現在ウェーブ数、マップ上の敵数、所持金、最大ウェーブ数(この
    ウェーブを凌げばクリア)、スコア
public ObservableComponent<ArrayList<BaseTurret>> availableTurretList; // 利用可能なタレ
    ット群
public ObservableComponent<BaseTurret> selectingTurret; // 画面右にタレット情報を表示させ
    るための変数
public ObservableComponent<BaseTurret> purchasingTurret; // 購入対象のタレット
private EnemySpawnerManager enemySpawnerManager; //
    wave ごとに EnemySpawner を配置するためのクラス
public ObservableComponent<ArrayList<Tower>> targets; // 襲撃目標群
// 新しく作成されるゲーム内オブジェクトが属する、ゲーム内オブジェクトのヒエラルキーの親
private TurretParent turretParent; // タレットの親クラス
private TowerParent towerParent; // タワーの親クラス
private EnemySpawnerParent enemySpawnerParent; // スポナーの親クラス
```

---

#### 3.2.1.3 メソッド

##### 3.2.1.3.1 receiveIClickable(IClickable iClickable)

Controller が、クリックされた IClickable のオブジェクトを Model に通知するためのメソッド。受け取るオブジェクトのクラスによって処理が以下のように変わる。

- null である場合
  - 選択中のタレットを null にする
- BaseTurret のインスタンスである場合

- 選択中のタレットをクリックされたタレットに更新する
- TurretSocket のインスタンスである場合
  - 選択中のタレットが null でなければ、その TurretSocket に対しタレットの設置を試みる

#### **3.2.1.3.2 setCameraMoveDirection(Vector2 cameraMoveDirection)**

Controller が、キー入力を受け取りカメラ移動ベクトルを計算し、そのベクトルを Model に渡すためのメソッド。

#### **3.2.1.3.3 setCameraZoomOperation(int cameraZoomOperation)**

Controller が、キー入力を受け取りカメラの拡縮操作を範囲 [-1,1] の整数に変換し、Model に渡すためのメソッド。

#### **3.2.1.3.4 actionPerformed(ActionEvent e)**

タイマーから呼び出されることで、ゲームの状態遷移を行うメソッド。ActionEvent として受け取るタイマーの種類により処理が以下のように変わる。

- nextWaveTimer の場合
  - カウントダウン値を減らしながら wave 開始までのカウントダウンを行う。
  - カウントダウンが終了したら currentState を OnWave とし、waveEndTimer を起動する。
- waveEndTimer の場合
  - waveNum が maxWaveNum に一致しない、すなわち凌ぎ切ればクリアとなるウェーブ数に満たない場合は、currentState を WaitingForNextWave とし、カウントダウン値を初期化し、nextWaveTimer を起動する。
  - waveNum が maxWaveNum に一致する場合はゲームクリアとなる。リザルト画面遷移までのタイマーである gameClearTimer を起動させる。
- gameOverTimer および gameClearTimer の場合
  - スコアを計算し、Intent としてスコアとクリアの成否の情報を ResultScene に渡す。ResultScene に遷移する。

#### **3.2.1.3.5 calc(float deltaTime)**

フレーム毎に必要となる処理を記述するメソッド。本プログラムではカメラの移動と拡縮を行う。カメラ操作の入力は Controller が受け取り、移動方向ベクトルと拡縮操作を範囲 [-1,1] の整数で表したものを受け取る。これらはそれぞれ Model のフィールドである cameraMoveDirection と cameraZoomOperation に格納される。よって、calc メソッドでこれらの値を元にカメラをマイフレーム毎に操作することで、カメラの移動と拡縮を実装している。

#### **3.2.1.3.6 onTowerFall(Tower tower)**

襲撃目標が破壊されたときに呼ばれるメソッド。全目標が破壊された場合、currentState が GameClear でなければゲームオーバーのリザルトシーンに遷移するための処理を行う。currentState を GameOver にし、gameOverTimer を起動させる。クリア条件は「maxWaveNum のウェーブを凌ぎ切った時点でクリア」であるため、クリア後にゲームオーバーの判定を行わないようにした。

### 3.2.1.3.7 pauseKeyPressed()

ポーズキーの入力時に Controller から呼ばれるメソッド。ポーズ中のシーンに遷移する。

## 3.2.2 EnemySpawner

### 3.2.2.1 仕様

- ゲーム内オブジェクトクラス GameObject を継承する
- マップに設置されると、敵を自分の場所に一定時間間隔で出現させる
- 一定の敵数が出現したら出現を止める
- ゲームのポーズ時にタイマーを止め、再開時に戻すことができる

### 3.2.2.2 フィールド

---

```
private int spawnCount; // 敵出現数。
private int initialDelay; // Wave 開始から敵出現開始までの遅延(ms)
private int spawnDelay; // 敵出現間隔(ms)
private GameTimer timer; // 出現待ちに用いるゲーム内タイマー
private BaseEnemy spawnEnemy; // 出現敵種
private Vertex initialVertex; // 敵出現地点
private EnemyParent enemyParent; // 出現した敵が属するゲーム内オブジェクトのヒエラルキーの親
private EnemyDieListener enemyDieListener; // エネミー死亡リスナー
```

---

### 3.2.2.3 メソッド

#### 3.2.2.3.1 actionPerformed(ActionEvent e)

spawnCount が 0 より大きいならば、敵を自身の場所に出現させ、spawnCount を 1 減らす。タイマーが spawnDelay のミリ秒分の周期で actionPerformed を呼び出す。

#### 3.2.2.3.2 public void pause(), public void unpause()

ゲームをポーズする際に EnemySpawner が停止しなければ、ポーズ画面の裏で敵が出現し続けることになる。よって、EnemySpawner がフィールドに持つタイマーを停止および再開することが出来るメソッドとしてこれらを定義した。

## 3.2.3 TurretSocket

### 3.2.3.1 仕様

- タレットの土台の役割を持つ。よって、タレットを持つことができる。
- 設置処理は Model が行うため、TurretSocket では実装しない。その代わり、Model からタレット配置処理を呼び出せるようパブリックメソッド tryBuildTurret を用意する。
- IClickable を implements する。Controller はクリックされたオブジェクト (IClickable) を検索して Model に渡すことができ、これを用いるため。

### 3.2.3.2 フィールド

---

```
private TurretParent turretParent; // 配置されたタレットが属するゲーム内オブジェクトのヒエラルキーの親
private BaseTurret turret; // タレットを持つ。配置されていない間はnullとする
```

---

### 3.2.3.3 メソッド

#### 3.2.3.3.1 tryBuildTurret(BaseTurret turret)

Model は、TurretSocket がクリックされると、その TurretSocket に対し tryBuildTurret を呼び出す。この際、引数には選択中のタレットが入る。こうすることで、タレットが選択された状態で TurretSocket をクリックすると、その TurretSocket に選択中のタレットを建設できる。

「タレットの設置を試みる」とした理由は二つある。まず一つ目に、タレットが選択中でない場合は、選択中のタレットが null となるためである。二つ目は、TurretSocket にすでにタレットが設置されている場合は設置処理をしないためである。

## 3.2.4 TurretSocketArranger

### 3.2.4.1 仕様

TurretSocket を配置する機能を持ったクラスであり、フィールド変数はない。

### 3.2.4.2 メソッド

#### 3.2.4.2.1 arrangeTurrets(String turretSocketsPath, GameObject root, TurretSocketParent turretSocketParent, TurretParent turretParent)

Scanner を用いて、TurretSocket を配置する座標を記述したファイルを読み込み、マップ上に配置する。ファイルのフォーマットは一行目が入力行数、それ以降が TurretSocket を配置するマップ上の x 座標と y 座標となっている。

(文責: 上田)

## 3.3 青木担当分主要クラス

### 3.3.1 GameSceneController

#### 仕様

マウスやキーの操作を受け取り、その際の動作を GameSceneModel に渡す

- 画面の移動 (WASD キー)
- 画面の拡縮 (Shift Space キー)
- タレットの配置と売却とアップデート (マウスの左クリック)

#### フィールド

---

```
private boolean isWKeyPressed;
private boolean isAKeyPressed;
private boolean isSKeyPressed;
private boolean isDKeyPressed;
private boolean isSHIFTKeyPressed;
private boolean isSPACEKeyPressed;
// WASD 及び Shift, Space キーが押されているかを判定するフィールド (押されていれば true、押されていなければ false となる)

private GameSceneModel gameModel;
private GameSceneView gameView;
// GameSceneModel と GameSceneView を表すフィールド

private GameSceneView.AddTurretPanel addTurretPanel;
private GameSceneView.UpgradeTurretPanel upgradeTurretPanel;
```

```

// タレット購入ボタンがあるパネルとアップグレードボタンがあるパネルを表すフィールド
private JButton rangeButton, powerButton, rofButton, sellButton;
// タレットのアップグレードと売却をするボタンを表すフィールド
private ObservableComponent<ArrayList<TurretDisplayButton>>
    turretButtons;
// タレットの購入ボタンを要素とする配列を表すフィールド

```

---

## メソッド

**public void actionPerformed(ActionEvent e)**

各ボタンからのイベントを受け取った際に、GameSceneModelでタレットの購入や売却、アップデートを行うメソッド。

**public void addActionListenerToTurretButtons()**

タレットボタンに ActionListener を付与するフィールド。

**public void moveCamera()**

画面の移動を行うメソッド。WASD キーが押されているかどうかによって方向ベクトルを求め、それを標準化して GameSceneModel に渡している。

**public void updateScaleOperation()**

画面の拡縮を行うメソッド。Shift, Space キーが押されているかどうかによって拡縮を判断し、それを GameSceneModel に渡している。

**public void mouseClicked(MouseEvent e)**

マウスで左クリックした座標を GameSceneModel に渡すメソッド。クリックした座標を受け取ったら、その座標をワールド座標に変換してから、クリック可能なゲームオブジェクトをクリックしているかどうかを確かめる。もしクリックしていたら model にそれを通知。クリックしていないかったら NULL を返す。クリックしているオブジェクトは、findClickedObject にワールド座標を渡して判断する。

**public void keyPressed(KeyEvent e)**

キーを押した時、それが特定のキー (WASD, Shift, Space キー) ならば、それぞれの判定フィールドを true にする。また、WASD キーを押した時には moveCamera メソッドを、Shift または Space キーを押した時には updateScaleOperation メソッドをそれぞれ呼び出す。

**public void keyReleased(KeyEvent e)**

キーを離した時、それが特定のキー (WASD, Shift, Space キー) ならば、それぞれの判定フィールドを false にする。また、WASD キーを押した時には moveCamera メソッドを、Shift または Space キーを押した時には updateScaleOperation メソッドをそれぞれ呼び出す。

**public void update(Observable o, Object arg)**

turretButtons の変更を検知すると、その変更を addActionListenerToTurretButtons フィールドに伝えるフィールド。

### 3.3.2 BaseTurret

#### 仕様

全タレットの親クラスであり、タレットの持つ情報を外部へと渡したり、パラメータのレベルアップを行うクラス。タレットの持つ情報は以下のようなものがある。

- 名前
- 画像
- 攻撃力とそのレベル
- 攻撃範囲とそのレベル
- 攻撃速度とそのレベル
- 最大レベル
- 購入金額
- 売却金額

## フィールド

---

```

protected String name; //名前のフィールド
protected double turretCollisionRadius; //攻撃範囲のフィールド
protected double attackPower; //攻撃力のフィールド
protected int rof; //攻撃速度のフィールド
protected int attackLevel; //攻撃力のレベルのフィールド
protected int rangeLevel; //攻撃範囲のレベルのフィールド
protected int rofLevel; //攻撃速度のレベルのフィールド
protected int maxLevel; //各パラメータの最大レベル(共通)のフィールド
protected int purchasePrice; //購入金額のフィールド
protected Image[] images; //画像のフィールド
protected IMoneyTransfer iMoneyTransfer; //所持金を追加するフィールド
protected GameObject enemyParentObject; //エネミーの親クラスのフィールド

```

---

## メソッド

### **protected void onSummoned()**

タレットを配置した時に処理を行うメソッド。ここでは、エネミーの親クラスを機構構の中から探している。

### **public void sell()**

タレットを売却する処理を行うメソッド。implement した iMoneyTransfer を利用して、タレット売却時にタレットの売却値分だけ所持金が増えるようにしている。

### **public double getTurretCollisionRadius()**

攻撃範囲 (turretCollisionRadius) を外部に渡すメソッド。

### **public double getAttackPower()**

攻撃力 (attackPower) を外部に渡すメソッド。

### **public int getRof()**

攻撃速度 (rof) を外部に渡すメソッド。

### **public Image getImage()**

タレットの画像 (image) を外部に渡すメソッド。

### **public String getName()**

タレットの名前 (name) を外部に渡すメソッド。

### **public int getAttackLevel()**

攻撃力のレベル (attackLevel) を外部に渡すメソッド。

```
public int getRangeLevel()
    攻撃範囲のレベル (rangeLevel) を外部に渡すメソッド。

public int getRofLevel()
    攻撃速度のレベル (rofLevel) を外部に渡すメソッド。

public int getMaxLevel()
    各パラメータの最大レベル (maxLevel) を外部に渡すメソッド。

public abstract int getSalePrice()
    売却金額を外部に渡すメソッド。売却金額の計算は他のクラスで行うため、抽象クラスとなっている。

public int getPurchasePrice()
    購入金額 (purchasePrice) を外部に渡すメソッド。

public abstract int getAttackPowerLevelupPrice()
    攻撃力をレベルアップした時に上がった売却金額を外部に渡すメソッド。売却金額の計算は他のクラスで行うため、抽象クラスとなっている。

public abstract int getTurretCollisionRadiusLevelupPrice()
    攻撃範囲をレベルアップした時に上がった売却金額を外部に渡すメソッド。売却金額の計算は他のクラスで行うため、抽象クラスとなっている。

public abstract int getRofLevelupPrice()
    攻撃速度をレベルアップした時に上がった売却金額を外部に渡すメソッド。売却金額の計算は他のクラスで行うため、抽象クラスとなっている。

public void onSelected()
    対象を選択している時に処理を行うメソッド。

public void onUnSelected()
    対象を選択していない時に処理を行うメソッド。

public void upAttackLevel()
    攻撃力をレベルアップの処理を行うメソッド。

public void upRangeLevel()
    攻撃範囲のレベルアップの処理を行うメソッド。

public void upRofLevel()
    攻撃速度のレベルアップの処理を行うメソッド。

public BaseTurret clone()
    タレットの複製を行うメソッド。これにより、タレットを新しく配置する度に BaseTurret を作る必要がない。
```

### 3.3.3 LaserTurret

#### 仕様

レーザー塔の処理を行うクラスであり、エネミーにレーザーで攻撃を行う処理や、レベルに応じた各パラメータの計算を行う。

#### フィールド

---

```
private static final double TURRET_RADIUS = 50; // 見た目の大きさを表す定数フィールド。
private int attackPowerLevelupPrice; //攻撃力をレベルアップした時に上がった売却金額を表すフィールド。
private int turretCollisionRadiusLevelupPrice; //攻撃範囲をレベルアップした時に上がった売却金額を表すフィールド。
private int rofLevelupPrice; //攻撃速度をレベルアップした時に上がった売却金額を表すフィールド。
private int totalPurchasePrice = 0; //タレットの購入とレベルアップで払った金額の合計を表すフィールド。売却金額の計算に用いる。
ArrayList<GameObject> nearEnemies; //レーザー塔の攻撃範囲内にいるすべてのエネミーを表す配列フィールド。
private final static double LEVELUP_ATTACK_POWER = 0.2; //攻撃力のレベルアップの計算式に用いる比例定数のフィールド。
private final static double LEVELUP_COLLISION_RADIUS = 10; //攻撃範囲のレベルアップの計算式に用いる比例定数のフィールド。
private final static double INTERCEPT_COLLISION_RADIUS = 150; //初期レベル(レベル1)時の攻撃範囲を表す比例フィールド。
private final static int LEVELUP_LANDING_TIME = 50; //攻撃速度のレベルアップの計算式に用いる比例定数のフィールド。
private final static int LEVELUP_PRICE = 10; //レベルアップした際の売却金額の計算式に用いる比例定数のフィールド。
private static final double INTERCEPT_ATTACK_POWER = 1; //初期レベル(レベル1)時の攻撃力を表す比例フィールド。
private SearchEnemyField searchEnemyField; //一定範囲内のエネミーを探す円形のオブジェクトを表すフィールド。
private GameTimer timer; //ゲームタイマーを表すフィールド。
```

---

## メソッド

### **public void onDestroyed()**

レーザー塔が削除(売却)された時に処理を行うメソッド。ゲームタイマーを停止する。

### **protected void onSummoned()**

レーザー塔を配置した時に処理を行うメソッド。索敵範囲をセットしたsearchEnemyFieldを木構造における自身の子に加えて、ゲームタイマーをスタートする。

### **public void upAttackLevel()**

レベルアップにより上昇した攻撃力の計算を行うメソッド。計算式は「初期レベルの攻撃力 + 比例定数 \* 攻撃力のレベル」である。また、Laserクラスに攻撃力の値を渡している。

### **public void upRangeLevel()**

レベルアップにより上昇した攻撃範囲の計算を行うメソッド。計算式は「初期レベルの攻撃範囲 + 比例定数 \* 攻撃範囲のレベル」である。また、Laserクラスに攻撃範囲の値を渡している。

### **public void upRofLevel()**

レベルアップにより上昇した攻撃速度の計算を行うメソッド。計算式は「初期レベルの攻撃速度 - 比例定数 \* 攻撃速度のレベル」である。ここでは、ゲームタイマーを一度停止し、新たに計算した攻撃速度(rof)をゲームタイマーに渡して、再度タイマーをスタートすることで、攻撃速度の変化を実現している。

### **public int getAttackPowerLevelupPrice()**

攻撃力をレベルアップした時に上がった売却金額を外部に渡すメソッド。計算式は「初期レベルの売却値 + 比例定数 \* 攻撃力のレベル」である。

### **public int getTurretCollisionRadiusLevelupPrice()**

攻撃範囲をレベルアップした時に上がった売却金額を外部に渡すメソッド。計算式は「初期レベルの売却値 + 比例定数 \* 攻撃範囲のレベル」である。

**public int getRofLevelupPrice()**

攻撃速度をレベルアップした時に上がった売却金額を外部に渡すメソッド。計算式は「初期レベルの売却値 + 比例定数 \* 攻撃速度のレベル」である。

**public int getSalePrice()**

売却金額を計算して外部に渡すメソッド。計算式は「(購入金額 + レーザー塔に使った金額の合計) \* 0.8」である。

**public void hitLaser()**

レーザーを発射するメソッド。攻撃範囲内にいる攻撃済みのエネミーを集合 enemySetInAttack に加え、攻撃範囲内のエネミーが enemySetInAttack に含まれていなければ、Laser クラスを作って攻撃を行う。これにより、一度攻撃したエネミーに対して Laser を生成し続けてゲームの処理が重くなってしまうのを防いだ。また、Laser クラスを生成した際に、Laser クラスを自身の子に加えている。

**public void actionPerformed(ActionEvent e)**

ゲームタイマーが動いている時に、hitLaser メソッドを呼び出すメソッド。

**public void pause()**

ポーズした時の処理を行うメソッド。タイマーを一時停止する。

**public void unpause()**

ポーズを解除した時の処理を行うメソッド。タイマーを再開する。

**public void onSelected()**

レーザー塔を選択している時に処理を行うメソッド。searchEnemyField を表示する為にフラグを true にする。

**public void onUnSelected()**

レーザー塔を選択していない時に処理を行うメソッド。searchEnemyField を表示しない為にフラグを false にする。

### 3.3.4 Laser

#### 仕様

レーザー塔から発射されるレーザーの処理を行うクラスであり、レーザーの表示やエネミーにダメージを与える処理を行う。

#### フィールド

ソースコード 1: uectd.game.gameScene.gameMain

---

```
protected double attackPower; //攻撃力を表すフィールド
protected double radius; //円形の攻撃範囲の半径を表すフィールド
private IAttackable attacker; //攻撃するタレット(レーザー塔)を表すフィールド
private GameObject enemyParentObject; //エネミーの基底クラスを表すフィールド
protected BaseEnemy nearEnemy; //攻撃範囲内にいるエネミーの内の
                               1体を表すフィールド
private GameTimer timer; //ゲームタイマーを表すフィールド
private Image image; //レーザーの画像を表すフィールド
```

---

#### メソッド

```

protected void onSummoned()
    レーザーを配置した時に処理を行うメソッド。ゲームタイマーをスタートする。

public void pause()
    ポーズした時の処理を行うメソッド。タイマーを一時停止する。

public void unpause()
    ポーズを解除した時の処理を行うメソッド。タイマーを再開する。

public void calc(float deltaTime)
    レーザーの更新処理を行うメソッド。nearEnemy が攻撃範囲内にいる時にはレーザーを表示し、それ以外の時はレーザーを表示しない(表示していたレーザーを消去する)。

public void draw(Graphics g, Vector2 screenPosition, float ratio)
    レーザーを適切な形に成形して「表示する」メソッド。タレットと攻撃するエネミーの位置関係は常に変化する為、位置関係に応じてレーザーの形を成形する必要があった。レーザーの成形にはアフィン変換行列を利用して、Enemy の位置に対する Laser の長さや傾きを常に計算している。なお、グラフィックス (g) はすべての draw で使いまわしている為、表示後にアフィン変換行列を元の状態に戻す必要がある。

public void onEnemyDead(BaseEnemy enemy)
    攻撃したエネミーが死んだ時の処理を行うメソッド。表示しているレーザーを削除する。

public void actionPerformed(ActionEvent e)
    ゲームタイマーが動いている時に、エネミーにダメージを与えるメソッド。

public void onDestroyed()
    レーザーを削除した時の処理を行うメソッド。ゲームタイマーを停止する。

```

### 3.3.5 SearchEnemyField

#### 仕様

タレットの周りに円形の攻撃範囲(索敵フィールド)を表示し、その範囲内のエネミーをサーチする。

#### フィールド

---

ソースコード 2: uectd.game.gameScene.gameMain

---

private EnemyParent enemyParent; //エネミーの基底クラスを表すフィールド。

---

#### メソッド

```

public void calc(float deltaTime)
    索敵フィールドの更新処理を行うメソッド。

protected void onSummoned()
    索敵フィールドを配置した時に処理を行うメソッド。ここでは、エネミーの親クラスを木構造の中から探している。

public void setRadius(double radius)
    砲撃範囲(半径)をセットし、表示する円の大きさを計算するメソッド。

```

```

public void setDisplayFlag(boolean isDisplaying)
    索敵フィールドを表示するかどうかのフラグを管理するメソッド。
public ArrayList<GameObject> searchNearEnemies()
    索敵フィールド内にいるすべてのエネミーの配列を返すメソッド。
public BaseEnemy searchNearestEnemy()
    索敵フィールド内にいるエネミーの内、円の中心のタレットから最も近い位置にいるエネミーを返すメソッド。索敵フィールド内にいるすべてのエネミーで、タレット(円の中心)からエネミーまでの距離の2乗を計算し(Vector2)、その値が最も小さいエネミーを返り値としている。

```

### 3.3.6 GameSound

#### 仕様

ゲームのBGMやSEを管理する。

#### フィールド

---

ソースコード 3: uectd.gameSystem.util

---

```

private String filePath; //流すBGM(SE)のリソースパスを表すフィールド。
private Clip clip; //オーディオ・クリップを表すフィールド。

```

---

#### メソッド

```

public void init()
    オーディオ・クリップを初期化するメソッド。
private static Clip createClip(File path)
    オーディオ・クリップを作成するメソッド。指定されたURLのオーディオ入力ストリームとファイルの形式を取得し、単一のオーディオ形式を含む指定した情報からデータラインの情報オブジェクトを構築する。最後に、指定されたLine.Infoオブジェクトの記述に一致するラインを取得することで、再生準備が整う。
public void start()
    オーディオ・クリップを再生するメソッド。
public void loop()
    オーディオ・クリップをループ再生するメソッド。
public void pause()
    オーディオ・クリップの再生を一時停止するメソッド。
public void stop()
    オーディオ・クリップの再生を停止するメソッド。
public boolean isPlaying()
    オーディオ・クリップを再生しているかどうかの判定をするメソッド。

```

### **public void volume(int soundVolume)**

オーディオ・クリップの音量調整をするメソッド。デシベル単位では音量を設定しにくくなってしまうので、対数を使った比率による調整を行っている。(例えば、音量を2倍にしたい時には、引数のsoundVolumeに2を入れればよい。)

(文責: 青木)

以下、全てのクラスの簡潔な説明を書く。文責は括弧内の各担当者にある。

**Direction(遠藤)** 敵や大砲など多くのオブジェクトは8方向の画像を持つため、角度（実数）と方向（8つ）の対応付けを行うためのユーティリティクラス

**Arrow(青木)** バリストが発射する矢を表すオブジェクト

**Background(遠藤)** 背景画像（ここでは衛星写真）を表すオブジェクト

**BallistaTurret(青木)** バリストのゲームオブジェクト

**BaseEnemy(遠藤)** 敵の基底クラス

**BaseEnemyActStrategy(遠藤)** 敵の挙動基底クラス（Strategy パターン）

**BaseTurret(青木)** タレット基底クラス

**Cannonball(青木)** 大砲から発射される砲弾を表すオブジェクト

**CannonTurret(青木)** 大砲のゲームオブジェクト

**Damage(青木)** ダメージの情報を表すクラス。ダメージの大きさと攻撃者の情報を保持する。後者は現在は未使用だが特定の攻撃に耐性を持つ特殊な敵の実装を想定している

**Cyclopes(遠藤)** 敵の一種のサイクロプスのオブジェクト

**EnemyAnimation(遠藤)** 敵のアニメーション処理を担当するクラス

**Goblin(遠藤)** 敵の一種のゴブリン

**Hobgoblin(遠藤)** 敵の一種のホブゴブリン

**SearchAndDestroyStrategy(遠藤)** 敵の挙動の一つ。選んだ防衛対象まで最短経路で移動し、攻撃する戦略

**Zombie(遠藤)** 敵の一種のゾンビ

**EnemyDieEffect(遠藤)** 敵死亡時のエフェクトのオブジェクト

**EnemyDieListener(上田)** 敵死亡に反応するイベントを取得するためのインターフェース

**EnemyParent(上田)** 敵親オブジェクト

**EnemySpawner(上田)** 与えられた情報をもとに敵を一定数出現させるオブジェクト

**EnemySpawnerArrangementStrategy(上田)** 敵出現アルゴリズムの基底クラス（Strategy パターンを使用）

**EnemySpawnerEffect(遠藤)** 的出現時のエフェクト Spawn にする？

**EnemySpawnerManager(上田)** wave 数をもとに EnemySpawner を配置する。配置アルゴリズムは EnemySpawnerArrangementStrategy に従う。

**EnemySpawnerMark(遠藤)** 敵出現の候補地を示すオブジェクト

**EnemySpawnerParent(上田)** EnemySpawner の親オブジェクト

**FileReadSpawnStrategy(上田)** 敵出現アルゴリズムの実装。将来的に異なる実装も可能なように Strategy パターンを使用している

**GameLevel(遠藤)** 敵が経路探索するためのデータ構造（グラフ）を持つオブジェクト

**Graph(遠藤)** グラフのデータ構造。頂点と辺の情報を持つ

**GraphBuilder(遠藤)** ファイルからデータを読み取り Graph クラスのインスタンスを生成するためのクラス

**IAttackable(上田)** ゲームオブジェクトに攻撃が可能な特性を示させるためのインターフェース。現在は未使用だが特定の攻撃に耐性を持つ特殊な敵の実装を想定している

**IDamageApplicable(青木)** ゲームオブジェクトにダメージを与えることができるという特性を示せるためのインターフェース。実装すると applyDamage メソッドも実装することになる

**IMoneyTransfer(上田)** 所持金のやり取りを行うことができる性質を表すインターフェース。実際には GameSceneModel が実装

**Laser(青木)** レーザー塔が発射するレーザーを表すオブジェクト

**LaserTurret(青木)** レーザー塔のゲームオブジェクト

**SearchEnemyField(青木)** タレットが索敵できる領域を表すオブジェクト

**Tower(遠藤)** タワー（防衛対象）を表すオブジェクト

**TowerArranger(上田)** タワーを設置するクラス

**TowerFallEffect(遠藤)** タワーが陥落したときのエフェクト

**TowerFallEffectSub(遠藤)** タワーが陥落したときの爆発エフェクトのうち、1つだけを表すオブジェクト

**TowerFallListener(遠藤)** タワーの陥落を感知するためのリスナーインターフェース

**TowerParent(上田)** タワーの親オブジェクト

**TurretAttackToEnemyEffect(青木)** タレットが的に攻撃したときのオブジェクト

**TurretParent(上田)** タレットの親オブジェクト

**TurretSocket(上田)** タレットを設置するための土台を表すオブジェクト

**TurretSocketArranger(上田)** タレットの土台を設置するためのクラス

**TurretSocketParent(上田)** タレットの土台の親オブジェクト

**GameScene(遠藤)** ゲームシーン

**GameSceneController(青木)** ゲームシーンのコントローラー

**GameSceneModel(上田)** ゲームシーンのモデル

**GameSceneView(遠藤)** ゲームシーンのビュー

**PauseScene(遠藤)** ポーズ画面を表すクラス

**PauseSceneController(遠藤)** ポーズ画面のコントローラー

**PauseSceneModel(遠藤)** ポーズ画面のモデル

**PauseSceneView(遠藤)** ポーズ画面のビュー

**ResourcePathDefines(遠藤)** ゲーム内で使うパスを定義しておくクラス

**ResultScene(遠藤)** リザルト画面を表すクラス

**ResultSceneController(遠藤)** リザルト画面のコントローラー

**ResultSceneModel(遠藤)** リザルト画面のモデル

**ResultSceneView(遠藤)** リザルト画面のビュー

**TitleScene(遠藤)** タイトル画面を表すクラス

**TitleSceneController(遠藤)** タイトル画面のコントローラー

**TitleSceneModel(遠藤)** タイトル画面のモデル

**TitleSceneView(遠藤)** タイトル画面のビュー

**Define(遠藤)** ウィンドウの大きさなど基本的な情報を定義しておくクラス

**GameObject(遠藤)** ゲームオブジェクトの基底クラス。Composite パターンを利用して再帰的な木構造を作る

**MainFrame(遠藤)** JFrame を継承した、ゲームの表示の基本となるフレーム。main メソッドはここにある。

**Scene(遠藤)** 1つのシーンを作るための基底クラス

**SceneController(遠藤)** 1つのシーンのコントローラー部分を作るための基底クラス

**SceneModel(遠藤)** 1つのシーンのモデル部分を作るための基底クラス

**SceneView(遠藤)** 1つのシーンのビュー部分を作るための基底クラス

**AnimationSprite(遠藤)** 敵のようにアニメーションするゲームオブジェクトの基底クラス。Sprite を継承している

**Camera(遠藤)** 描画する範囲を指定するカメラの役割を果たす特殊なゲームオブジェクト。

**CircleCollider(遠藤)** Collider を継承している、円の形をした当たり判定。

**Collider(遠藤)** ゲームオブジェクト同士の当たり判定を担当するクラス

**Effect(遠藤)** エフェクトの基底オブジェクト。Sprite を継承している

**GameSound(青木)** ゲーム内の音を流すのに使う、Clip クラスのラッパークラス

**GameTimer(遠藤)** javax.swing.Timer を継承してゲーム内で使えるようポーズ機能を追加したタイマークラス

**IClickable(遠藤)** クリック可能なオブジェクトであることを示すマーカーインターフェース

**ImageManager(遠藤)** 画像データを管理するクラス。Singleton パターンと Flyweight パターンを利用している。

**Intent(遠藤)** シーン間での情報の受け渡しに用いるクラス。ハッシュマップが実装に使われている。

**ObservableComponent(遠藤)** ジェネリクスを使ってクラスを Observable にするクラス

**SoundManager(遠藤)** 音声データを管理するクラス。ImageManager と同様 Singleton パターンと Flyweight パターンを利用している。

**Sprite(遠藤)** 画像を描画に使うゲームオブジェクトの派生オブジェクト。

**Vector2(遠藤)** 2 次元ベクトルを表すクラス。加算、内積、スカラー倍等基本的な演算ができるようになっている。

## 4 実行例

ゲームを実行すると、図 3 のタイトル画面が表示される。Start ボタンでゲーム開始、Exit でゲームを終了することができる。



図 3: タイトル画面。

ゲームを開始すると、図 4 の画面が表示される。画面上に所持金、防衛対象、ウェーブ開始までの残り時間が表示される。



図 4: ゲームスタート直後の画面。

カメラ操作はシフトキーとスペースキーで拡縮を、W,A,S,D キーで上下左右に移動することが出来る。図 5 に拡大したゲーム画面を示す。



図 5: シフトキーでゲームを画面ズームした状態。

星形のアイコンがタレットを配置できる場所である。タレットを配置する際は、画面下のタレットのアイコンをクリックして選択し、マップ上の星形のアイコンをクリックすることで設置できる。図 6 に、マップ上にキャノンを配置した状態を示す。



図 6: キャノンを配置した状態。

マップ上に配置したタレットを選択することで、敵を攻撃可能な範囲が赤い円で表示される。さらに、画面右にアップグレードと売却のメニューが表示される。タレットはキャノン、レーザー、バリスタの3種類あり、それぞれ特徴が異なる。一つ目のキャノンは攻撃範囲、威力、攻撃速度ともに平均的なタレットである。



図 7: キャノンを選択した状態。

二つ目のレーザーは、攻撃範囲、威力ともに低いが、攻撃可能範囲内の全ての敵に対し継続してダメージを与えることができる。



図 8: レーザーを配置した状態。範囲内の複数体の敵に同時に攻撃が可能。

三つ目のバリスタは、攻撃範囲、威力ともに高いが、攻撃速度が遅いタレットである。



図 9: バリスタを配置した状態。アップグレードをしていない状態でも攻撃範囲が広い。

アップグレードの種類は攻撃範囲、威力、攻撃速度の3種類あり、アップグレードに対応するボタンを押すと一定の金額を消費して性能を強化することができる。アップグレードのレベルは10段階あり、タレット設置時は3種類ともレベル1である。画面右のアップグレードボタンには「アップグレードの種類:現在のレベル:消費する金額」が表示される。攻撃範囲のレベルを上げると、そのタレットが攻撃可能な半径が広がるため、赤い円の表示もそれに従って大きくなる。



図 10: 攻撃範囲をレベル7まで強化した状態。図7と比べると範囲が広くなっている。

敵はマップ端の魔法陣から出現し、防衛対象に向かってくる。敵の種類はゾンビ、ゴブリン、ホブゴブリン、サイクロプスの4種類あり、以下の特徴を持つ。

- ゾンビ

- 体力、攻撃力、移動速度がいずれも平均的。
  - ゴブリン
    - 体力と攻撃力は低いが、移動速度が速い。
  - ホブゴブリン
    - 移動速度はゴブリンに劣るが、体力と攻撃力はゴブリンより強い。
  - サイクロプス
    - 移動速度はかなり遅いが、体力と攻撃力が非常に高い。
- 敵を倒すと、敵ごとに一定の金額が手に入る。ウェーブを進めると、複数の魔法陣から敵が同時に出現したり、時間差で出現したりし難易度が上がる。複数種類の敵が同時に出現することもある。



図 11: ウェーブが進むと、複数の魔法陣から敵が出現する。

防衛対象はゲーム開始時に候補地からランダムに2つ抽選で決定され、ゲーム開始直後は緑色の輪で表示される。この輪は、防衛対象の残りヒットポイントの表示としても機能しており、敵に攻撃されヒットポイントが減ると色が黄色、赤と変化する。



図 12: 防衛対象が攻撃を受けている状態。防衛対象のヒットポイントが減ると、輪の表示が変化する。

防衛対象は、残りヒットポイントがゼロになると陥落する。



図 13: 防衛対象が陥落した状態。

マップ上の全て、すなわち 2 つの防衛対象が陥落するとゲームオーバーとなる。



図 14: ゲームオーバーのリザルト画面。

一定のウェーブ数を凌ぎ切ると、ゲームクリアとなる。



図 15: ゲームクリアのリザルト画面。

敵を倒すと、敵ごとに一定のスコアが得られる。リザルトに表示されるスコアは、ゲームオーバーの場合は敵の討伐時のスコアのみとなるが、ゲームクリアの場合は陥落していないタワーの数だけ一定のスコアがさらに加算される。

(文責: 上田)

## 5 考察

### 5.1 プログラム制作の過程について

制作するものを決定した直後の段階で細かい仕様を決定せず、初めにゲームシステムを作成し、次に必要となるクラスや機能を話し合っては実装を繰り返し、タレットや敵クラス、ゲームの進行の処理などを順に作成して完成に至った。

制作の過程では、自分が担当する機能やクラスを決定しても、その箇所をどう実装したら良いかわからず作業が頻繁に滞ることが大きな問題となった。具体的には、ゲームシステム部で定義されているクラスの役割がわからず適切な処理を記述できなかったり、細かい仕様が決定されておらず自分の担当箇所についての相談を頻繁に挟むということが生じた。原因は、ゲームシステムに関する知識の不足と、実装段階で仕様決定が不十分だったことにあると分析する。ゲームシステムでどのような機能が実装されているかが周知されていれば、自分でライブラリを探して実装するべきなのか否かが各自で判断できよりスムーズに開発ができたかもしれない。また、細かい仕様や分担を最初に決定したうえで実装をできていれば、毎回話し合う必要は無くなるだろう。しかし、プロジェクト自体が膨大であることや知識の差があったことを考慮すると、最初に隅々まで仕様を決めたうえで各自がコーディングする形で開発をするのは困難であったと推測する。

### 5.2 完成したプログラムについて

- MVC パターンを利用して各担当者で分業してそれぞれのコーディングを行う
- Model の規模が大きいため、ゲームに登場する各オブジェクトを、Model のことや他の担当者の担当範囲のことをほぼ知らなくてもよい粒度で分割して実装する

という当初の目的は達成できたと考える。しかし、各オブジェクトを独立して動作させるため情報の流れが複雑になってしまった結果、各オブジェクトが持っていないなければならない情報を設計段階で見落としていると修正箇所がいくつかの場所にわたるといったこともあった。また、オブジェクトの生成が冗長になってしまった箇所がいくつか生じた。例えば、EnemySpawner のコンストラクタは次のようになっている。

```
public EnemySpawner(GameObject root, GameObject parent, int initialDelay, int spawnDelay
    , int spawnCount, BaseEnemy spawnEnemy, Vertex initialVertex, EnemyParent
    enemyParent, EnemyDieListener enemyDieListener) {
    super(root, parent);
    this.initialDelay = initialDelay;
    this.spawnDelay = spawnDelay;
    this.spawnCount = spawnCount;
    this.spawnEnemy = spawnEnemy;
    this.initialVertex = initialVertex;
    this.enemyParent = enemyParent;
    this.enemyDieListener = enemyDieListener;
}
```

このような箇所に関しては、生成に関するデザインパターンの適用を考えるべきかもしれない。ゲームオブジェクトをマップ上に配置するためには addChild() を実行し、配置するゲームオブジェクトをゲームオブジェクトの木構造のヒエラルキーに入れる必要がある。addChild メソッドで、親にあたるゲームオブジェクトを自動的にオブジェクトに渡すことができれば、コンストラクタで親と根のゲームオブジェクトを渡さずに済み、冗長さをなくすことで情報の流れの見通しを改善できると考える。さらに、この設計では、オブジェクトの生成ごとに根と親を渡すため、誤った参照を渡すと木構造が破綻してしまう危険がある。システム部に関連する処理のバグをシステム部に関係ない場所で発生させてしまうという特定が困難なバグにつながりかねないため、その部分をシステム部以外には隠すような設計にするべきだったと考える。

一方で、システム部と各シーンの部分を分離したことでのシーンなどに関係なく、各シーンに固有のことのみ考えてコーディングを行うことができた。これによって、シーン内の記述の簡潔さを保つとともに、シーンごとの独立性も高くなるように設計することができた。また、エネミーの出現の実装では、Strategy パターンを用いることで、柔軟に出現アルゴリズムを変えるよう設計できた。

(文責: 上田)

## 6 各自の反省と感想

### 6.1 青木

グループでのゲーム作成を通して、プログラミングを分業して行うことの難しさを学んだ。自身の担当クラスを書く為に、他のメンバーが書いたクラスを読み解くのが非常に大変だった。また、メンバー間での java に関する知識の差が大きく、自分の担当クラスを書く時も他のメンバーから教わりながら作業を進めた為に、一部完全に分業が出来なかったクラスもあった。一方で、プログラムの統合に用いた GitHub を活用することで、クラスやフィールド、メソッドの名前の共有が非常にやりやすかった。

自分が担当した Controller クラスでは、キーボードとマウスによる操作を前提とした実装にしたが、タッチパネルによる操作が可能になれば操作性が非常に良くなるだろう。また、タレットを 3 種類作成したが、ゲームの設計段階では、エネミーの進行を阻む壁やエネミーの経路に設置できる地雷、さらには空を飛ぶエネミーに対して有効な対空砲などのタレットも案として挙がっていた。しかし、これらのタレットは設計難易度が高く、プログラムがうまく書けなかつたので実装を断念した。いずれはこれらの機能を実装してゲーム性をさらに広げてみたいと思う。

Java に関しては、オブジェクト指向で言語が構築されている分、応用性が非常に高い言語であると感じた。一方で、ゲーム制作などの特定の用途で用いる際には、ゲーム制作に特化している Unity と比べて活用できる機能が少なく、不便に感じることが多々あった。また、MVC モデルによるプログラミングは、分業がしやすくなる分、MVC の分類を意識しながらコードを書くことが煩わしくなることが多かった。メンバーの持つ知識の差が少なく、完全に分業が出来た時に MVC モデルのありがたみを実感できるかもしれない。

プログラミング演習の授業に関して、初めて触れた高級言語である java を短期間で効率良く学習することが出来たと思う。一方で、ゲーム作成の時に生かせる知識(各デザインパターンや木構造・グラフに関するライブラリの使い方など)を授業で触れることが出来れば、ゲーム制作がよりスムーズに進みそうだと感じた。

初めての高級言語によるプログラミングやゲーム制作を通して学ぶことが多くあった授業だった。

### 6.2 上田

ゲーム制作をするのは初めてで、シーン遷移やゲームシステムについて全く知識が無い状態であった。序盤は担当の箇所についてどう処理すれば良いかわからず、知識共有をしながらの作業がほとんどになってしまったが、最終的にはゲームシステムをある程度理解したうえでコーディングが出来るようになった。オブジェクト指向のプログラミングに慣れることができ、デザインパターンを使うメリットやクラス管理について理解を深めることができたと感じる。本講義を通して、Java やオブジェクト指向、デザインパターン、MVC モデルを学んだうえで実践でき、実際にアプリケーションを作成する力が身に付けることができた。プレゼンテーションでは、優秀賞を狙う意欲のあるグループが多かったように感じる。授業の内容だけでなく、各グループが制作したいものについて知識が深まったと考える。

完成したプログラムについては、タワーディフェンスとしては十分な機能を実装できたので、おおむね当初予定していた通りのものができたと考える。しかし、ゲーム内容としては一定のウェーブを凌ぎ切ればクリアというだけであり、ゲームとしてのやりこみ要素はまだ乏しい。ゲームの進行によりアイテムやポイントをためることで次回プレイで有利に進められるような要素を追加することで改善をできたらと思う。また、プレゼンテーションの際に受けた指摘もあるが、画面外の敵の存在がわかるような表示ができるとより快適にプレイできると考える。

### 6.3 遠藤

グループメンバーでの情報の共有や作業の分担はかなり大変だった。システム部などでメンバーに知つてもらわなければならないことを説明した上で、各自のできる範囲内で作業を分担し、完成させるという一連の流れは一人で作業するだけでは得られない経験だった。

ゲームとしては最低限の機能を持つものは作れたが、これ以降改善できる点として、

- View を拡張して、画面外の敵の位置などが分かるようにマーカーを表示したりする。
- 敵の行動アルゴリズムにバリエーションを持たせる。例えば最短経路を通るだけでなく、砲撃をされにくいルートを探索する

というような改善ができたらと思う。

Java は今までに触れたことのある C# とは思想が異なり、扱うのに苦労した。比較的プログラマーにできることを多くする C# や C++ とは異なり、言語の文法をシンプルに保とうとする傾向がある Java の使用感は新鮮で勉強になった。

また、全体のプログラムを MVC の 3 つに分割することで、それぞれがお互いの細かい実装を知らなくてもプログラムを記述することができるというメリットを享受して開発ができたと感じる。

## 付録1：操作法マニュアル（ユーザーズマニュアル）



## 付録2：プログラムリスト

ソースコード 4: uectd.game.gameScene.Direction

```
1 package uectd.game.gameScene;
2
3 import uectd.gameSystem.util.Vector2;
4
5 public class Direction {
6     public static final int NORTH = 0;
7     public static final int NORTH_WEST = 1;
8     public static final int WEST = 2;
9     public static final int NORTH_EAST = 3;
10    public static final int EAST = 4;
11    public static final int SOUTH_WEST = 5;
12    public static final int SOUTH = 6;
13    public static final int SOUTH_EAST = 7;
14
15    private static final double PI_1_8 = Math.PI * 1 / 8;
16    private static final double PI_3_8 = Math.PI * 3 / 8;
17    private static final double PI_5_8 = Math.PI * 5 / 8;
18    private static final double PI_7_8 = Math.PI * 7 / 8;
19
20    public static int convertVectorToConstant(Vector2 vector) {
21        return convertAngleToConstant(Math.atan2(vector.y, vector.x));
22    }
23
24    public static int convertAngleToConstant(double angle) {
25        if (0 <= angle) {
26            if (angle <= PI_5_8) {
27                if (angle <= PI_3_8) {
28                    if (angle <= PI_1_8) {
29                        return EAST;
30                    }
31                    return NORTH_EAST;
32                }
33                return NORTH;
34            }
35            if (angle <= PI_7_8) {
36                return NORTH_WEST;
37            }
38        } else {
39            if (-PI_5_8 <= angle) {
40                if (-PI_3_8 <= angle) {
41                    if (-PI_1_8 <= angle) {
42                        return EAST;
43                    }
44                    return SOUTH_EAST;
45                }
46                return SOUTH;
47            }
48            if (-PI_7_8 <= angle) {
49                return SOUTH_WEST;
50            }
51        }
52        return WEST;
53    }
54 }
```

ソースコード 5: uectd.game.gameScene.gameMain.Arrow

```
1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.game.gameScene.Direction;
5 import uectd.gameSystem.GameObject;
6 import uectd.gameSystem.util.CircleCollider;
7 import uectd.gameSystem.util.ImageManager;
8 import uectd.gameSystem.util.Sprite;
9 import uectd.gameSystem.util.Vector2;
10
11 import java.awt.*;
```

```

13 public class Arrow extends Sprite implements IAttackable {
14     private double attackPower;
15     private double radius;
16     private double cannonballSpeed;
17     private IAttackable attacker;
18     private GameObject enemyParentObject;
19     private BaseEnemy nearestEnemy;
20     private Image[] images;
21
22     public Arrow(GameObject root, GameObject parent, double attackPower, int landingTime,
23                 BallistaTurret BallistaTurret,
24                 BaseEnemy nearestEnemy) {
25         super(root, parent);
26         this.attackPower = attackPower;
27         this.radius = 20;
28         this.cannonballSpeed = 400;
29         this.collider = new CircleCollider(this, radius);
30         this.attacker = BallistaTurret;
31         this.nearestEnemy = nearestEnemy;
32         this.images = ImageManager.getInstance().getDivImage(ResourcePathDefines.
33                                         ARROW_IMAGES, 8, 1, 64, 64);
34         setImage(this.images[0]);
35         this.depth = 2;
36     }
37
38     @Override
39     protected void onSummoned() {
40         super.onSummoned();
41     }
42
43     @Override
44     public void calc(float deltaTime) {
45         super.calc(deltaTime);
46         if (nearestEnemy.getEnabled()) {
47             Vector2 diff = Vector2.diff(nearestEnemy.position, this.position);
48             setImage(images[Direction.convertAngleToConstant(
49                         Math.atan2(nearestEnemy.position.y - position.y, nearestEnemy.position.x
50                         - position.x))]);
51             if (diff.norm() < radius * radius) {
52                 nearestEnemy.applyDamage(new Damage(attackPower, this));
53                 // TODO 被弾エフェクト
54                 this.destroy();
55                 return;
56             }
57             Vector2 direction = diff.normalized();
58             position.add(Vector2.scale(direction, cannonballSpeed * deltaTime));
59         } else {
60             destroy();
61         }
62     }
63 }

```

---

ソースコード 6: uectd.game.gameScene.gameMain.BackGround

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.gameSystem.GameObject;
5 import uectd.gameSystem.util.ImageManager;
6 import uectd.gameSystem.util.Sprite;
7
8 public class BackGround extends Sprite {
9
10    public BackGround(GameObject root, GameObject parent) {
11        super(root, parent);
12        setImage(ImageManager.getInstance().getImage(ResourcePathDefines.BACKGROUND_IMAGE
13                  ));
14        this.depth = 10;
15    }
16 }

```

---

ソースコード 7: uectd.game.gameScene.gameMain.BallistaTurret

---

```
1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
4 import uectd.gameSystem.util.CircleCollider;
5 import uectd.gameSystem.util.GameTimer;
6 import uectd.gameSystem.util.ImageManager;
7 import uectd.game.ResourcePathDefines;
8 import uectd.game.gameScene.Direction;
9
10 import java.awt.event.*;
11
12 public class BallistaTurret extends BaseTurret implements ActionListener {
13     private static final double TURRET_RADIUS = 50; // 見た目の大さ
14     private int attackPowerLevelupPrice;
15     private int turretCollisionRadiusLevelupPrice;
16     private int rofLevelupPrice;
17     private int totalPurchasePrice = 0;
18     private final static double LEVELUP_ATTACK_POWER = 0.5;
19     private final static double LEVELUP_COLLISION_RADIUS = 20;
20     private final static double INTERCEPT_ATTACK_POWER = 20;
21     private final static double INTERCEPT_COLLISION_RADIUS = 400;
22     private final static int LEVELUP_LANDING_TIME = 50;
23     private final static int LEVELUP_PRICE = 10;
24     private final static int SALE_PRICE = 10;
25     private SearchEnemyField searchEnemyField;
26     private GameTimer timer;
27
28     public BallistaTurret(GameObject root, GameObject parent, IMoneyTransfer
29         iMoneyTransfer) {
30         super(root, parent, "Ballista", INTERCEPT_COLLISION_RADIUS +
31             LEVELUP_COLLISION_RADIUS,
32             INTERCEPT_ATTACK_POWER + LEVELUP_ATTACK_POWER, 2000, 10, 80,
33             ImageManager.getInstance().getDivImage(ResourcePathDefines.
34                 BALLISTA_TURRET_IMAGES, 8, 1, 64, 64),
35             iMoneyTransfer);
36         depth = 1;
37     }
38
39     @Override
40     public void onDestroyed() {
41         super.onDestroyed();
42         timer.stop();
43     }
44
45     @Override
46     protected void onSummoned() {
47         super.onSummoned();
48         this.collider = new CircleCollider(this, TURRET_RADIUS);
49         searchEnemyField = new SearchEnemyField(root, this);
50         searchEnemyField.setRadius(getTurretCollisionRadius());
51         this.addChild(searchEnemyField);
52         searchEnemyField.position = this.position;
53         this.timer = new GameTimer(getRof(), this);
54         this.timer.start();
55     }
56
57     @Override
58     public void upAttackLevel() {
59         super.upAttackLevel();
60         attackPower = INTERCEPT_ATTACK_POWER + LEVELUP_ATTACK_POWER * getAttackLevel();
61     }
62
63     @Override
64     public void upRangeLevel() {
65         super.upRangeLevel();
66         System.out.print(turretCollisionRadius + "□");
67         turretCollisionRadius = INTERCEPT_COLLISION_RADIUS + LEVELUP_COLLISION_RADIUS *
68             getRangeLevel();
69         System.out.println(turretCollisionRadius);
70         searchEnemyField.setRadius(turretCollisionRadius);
71     }
72
73     @Override
74     public void upRofLevel() {
```

```

71     int prevRofLevel = getRofLevel();
72     super.upRofLevel();
73     if (prevRofLevel != getRofLevel()) {
74         rof = (int) (2000 - LEVELUP_LANDING_TIME * getRofLevel());
75         timer.pause();
76         int nextInitialDelay = timer.getInitialDelay();
77         timer.stop();
78         timer = new GameTimer(rof, this);
79         timer.setInitialDelay(nextInitialDelay);
80         timer.start();
81     }
82 }
83
84 @Override
85 public int getAttackPowerLevelupPrice() {
86     attackPowerLevelupPrice = 20 + LEVELUP_PRICE * getAttackLevel();
87     totalPurchasePrice += attackPowerLevelupPrice;
88     return attackPowerLevelupPrice;
89 }
90
91 @Override
92 public int getTurretCollisionRadiusLevelupPrice() {
93     turretCollisionRadiusLevelupPrice = 20 + LEVELUP_PRICE * getRangeLevel();
94     totalPurchasePrice += turretCollisionRadiusLevelupPrice;
95     return turretCollisionRadiusLevelupPrice;
96 }
97
98 @Override
99 public int getRofLevelupPrice() {
100    rofLevelupPrice = 20 + LEVELUP_PRICE * getRofLevel();
101    totalPurchasePrice += rofLevelupPrice;
102    return rofLevelupPrice;
103 }
104
105 @Override
106 public int getSalePrice() {
107     return (purchasePrice + totalPurchasePrice) * 8 / 10;
108 }
109
110 public void fireCannonball() {
111     BaseEnemy nearestEnemy = searchEnemyField.searchNearestEnemy();
112
113     // enemy の座標に Cannonball を置く
114     if (nearestEnemy != null) {
115         Arrow arrow = new Arrow(root, root, attackPower, rof, this, nearestEnemy);
116         arrow.position = this.position.clone();
117         root.addChild(arrow);
118         setImage(images[Direction.convertAngleToConstant(
119             Math.atan2(nearestEnemy.position.y - position.y, nearestEnemy.position.x
120             - position.x))]);
121     }
122 }
123
124 @Override
125 public void actionPerformed(ActionEvent e) {
126     if (e.getSource() == this.timer)
127         this.fireCannonball();
128 }
129
130 @Override
131 public void pause() {
132     super.pause();
133     timer.pause();
134 }
135
136 @Override
137 public void unpause() {
138     super.unpause();
139     timer.unpause();
140 }
141
142 @Override
143 public void onSelected() {
144     super.onSelected();
145     searchEnemyField.setDisplayFlag(true);

```

```

145     }
146
147     @Override
148     public void onUnSelected() {
149         super.onUnSelected();
150         searchEnemyField.setDisplayFlag(false);
151     }
152 }
153 }
```

---

ソースコード 8: uectd.game.gameScene.gameMain.BaseEnemy

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.util.ArrayList;
4
5 import uectd.gameSystem.GameObject;
6 import uectd.gameSystem.util.AnimationSprite;
7
8 public abstract class BaseEnemy extends AnimationSprite implements IDamageApplicable,
9     IAttackable {
10
11     // 耐久、攻撃、スピード、ドロップ金額、撃破時スコア
12     protected double hp, attackPower, speed;
13     protected int dropValue, dropScore;
14
15     protected BaseEnemyActStrategy mover; // Strategy パターンを使用して実装、敵の行動パターンを表すクラス
16     protected GameLevel level; // ゲームのマップデータのオブジェクト。経路探索に使う
17     protected ArrayList<Tower> targets; // 襲撃対象のリスト
18     protected Graph.Vertex initialVertex; // 最初に出現したマップ上の地点
19     protected ArrayList<EnemyDieListener> enemyDieListeners; // この敵が撃破されたときの通
20     知を受け取るリスナーの一覧
21
22     @Override
23     public BaseEnemy clone() {
24         BaseEnemy enemy = null;
25         enemy = (BaseEnemy) super.clone();
26         enemy.mover = mover.clone();
27         enemy.mover.enemyGameObject = enemy;
28         enemy.enemyDieListeners = new ArrayList<>(enemy.enemyDieListeners);
29         return enemy;
30     }
31
32     public BaseEnemy(GameObject root, GameObject parent, GameLevel level, ArrayList<Tower
33         > targets,
34         Graph.Vertex initialVertex) {
35         super(root, parent);
36         this.level = level;
37         this.targets = targets;
38         this.enemyDieListeners = new ArrayList<>();
39         this.initialVertex = initialVertex;
40     }
41
42     @Override
43     public void pause() {
44         super.pause();
45         mover.pause();
46     }
47
48     @Override
49     public void unpause() {
50         super.unpause();
51         mover.unpause();
52     }
53
54     public double getHp() {
55         return hp;
56     }
57
58     public double getAttackPower() {
59         return attackPower;
60     }
```

```

58     public double getSpeed() {
59         return speed;
60     }
61
62     public int getDropValue() {
63         return dropValue;
64     }
65
66     public int getDropScore() {
67         return dropScore;
68     }
69
70     @Override
71     public void calc(float deltaTime) {
72         mover.calc(deltaTime);
73     }
74
75     @Override
76     public void applyDamage(Damage d) {
77         if (this.getEnabled()) {
78             hp -= d.attack;
79             if (hp < 0) {
80                 hp = 0;
81                 var effect = new EnemyDieEffect(root, root);
82                 effect.position = this.position.clone();
83                 root.addChild(effect);
84                 for (var enemyDieListener : enemyDieListeners) {
85                     enemyDieListener.onEnemyDead(this);
86                 }
87                 destroy();
88             }
89         }
90     }
91
92     public void addEnemyDieListener(EnemyDieListener enemyDieListener) {
93         enemyDieListeners.add(enemyDieListener);
94     }
95
96     public void removeEnemyDieListener(EnemyDieListener enemyDieListener) {
97         enemyDieListeners.remove(enemyDieListener);
98     }
99
100    public Graph.Vertex getInitialVertex() {
101        return initialVertex;
102    }
103
104    @Override
105    protected void onSummoned() {
106        super.onSummoned();
107        mover.onSummoned();
108    }
109
110    @Override
111    public void onDestroyed() {
112        super.onDestroyed();
113        mover.onDestroyed();
114    }
115 }
116 }
```

---

ソースコード 9: uectd.game.gameScene.gameMain.BaseEnemyActStrategy

---

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.util.ArrayList;
4
5 import uectd.gameSystem.FatalError;
6 import uectd.gameSystem.GameObject;
7
8 public abstract class BaseEnemyActStrategy implements Cloneable {
9     protected BaseEnemy enemyGameObject;
10    protected GameLevel level;
11    protected ArrayList<Tower> targets;
12
13    abstract public void calc(float deltaTime);
```

```

14     public BaseEnemyActStrategy(GameObject enemyGameObject, GameLevel level, ArrayList<
15         Tower> targets) {
16         this.enemyGameObject = (BaseEnemy) enemyGameObject;
17         this.level = level;
18         this.targets = targets;
19     }
20
21     public void pause() {
22     }
23
24     public void unpause() {
25     }
26
27     @Override
28     public BaseEnemyActStrategy clone() {
29         BaseEnemyActStrategy baseEnemyActStrategy = null;
30
31         try {
32             baseEnemyActStrategy = (BaseEnemyActStrategy) super.clone();
33         } catch (CloneNotSupportedException ce) {
34             ce.printStackTrace();
35             FatalError.quit("オブジェクトのクローンに失敗しました");
36         }
37         return baseEnemyActStrategy;
38     }
39
40     public abstract double getAngle();
41
42     public void onSummoned() {
43     }
44
45     public void onDestroyed() {
46     }
47 }
```

---

ソースコード 10: uectd.game.gameScene.gameMain.BaseTurret

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
4 import uectd.gameSystem.util.IClickable;
5 import uectd.gameSystem.util.Sprite;
6
7 import java.awt.*;
8
9 public abstract class BaseTurret extends Sprite implements IAttackable, IClickable {
10     protected String name;
11     protected double turretCollisionRadius;
12     protected double attackPower;
13     protected int rof;
14     protected int attackLevel;
15     protected int rangeLevel;
16     protected int rofLevel;
17     protected int maxLevel;
18     protected int purchasePrice;
19     protected int levelupPrice;
20     protected final static int LEVELUP_PRICE = 10;
21     protected Image[] images;
22     protected IMoneyTransfer iMoneyTransfer;
23     protected GameObject enemyParentObject;
24
25     public BaseTurret(GameObject root, GameObject parent, String name, double
26         turretCollisionRadius, double attackPower,
27         int rof, int maxLevel, int purchasePrice, Image[] images, IMoneyTransfer
28         iMoneyTransfer) {
29         super(root, parent);
30         this.name = name;
31         this.turretCollisionRadius = turretCollisionRadius;
32         this.attackPower = attackPower;
33         this.rof = rof;
34         this.attackLevel = 1;
35         this.rangeLevel = 1;
36         this.rofLevel = 1;
```

```

35         this.maxLevel = maxLevel;
36         this.purchasePrice = purchasePrice;
37         this.images = images;
38         this.iMoneyTransfer = iMoneyTransfer;
39         this.setImage(images[0]);
40     }
41
42     @Override
43     protected void onSummoned() {
44         super.onSummoned();
45         this.enemyParentObject = root.findChild(EnemyParent.class);
46     }
47
48     public void sell() {
49         iMoneyTransfer.moneyAdd(this.getSalePrice());
50         destroy();
51     }
52
53     public double getTurretCollisionRadius() {
54         return turretCollisionRadius;
55     }
56
57     public double getAttackPower() {
58         return attackPower;
59     }
60
61     public int getRof() {
62         return rof;
63     }
64
65     public Image getImage() {
66         return images[0];
67     }
68
69     public String getName() {
70         return this.name;
71     }
72
73     public int getAttackLevel() {
74         return this.attackLevel;
75     }
76
77     public int getRangeLevel() {
78         return this.rangeLevel;
79     }
80
81     public int getRofLevel() {
82         return this.rofLevel;
83     }
84
85     public int getMaxLevel() {
86         return this.maxLevel;
87     }
88
89     public abstract int getSalePrice();
90
91     public int getPurchasePrice() {
92         return this.purchasePrice;
93     }
94
95     public abstract int getAttackPowerLevelupPrice();
96
97     public abstract int getTurretCollisionRadiusLevelupPrice();
98
99     public abstract int getRofLevelupPrice();
100
101    public void onSelected() {
102    }
103
104    public void onUnSelected() {
105    }
106
107    public void upAttackLevel() {
108        if (attackLevel < maxLevel && iMoneyTransfer.tryMoneyPay(
109            getAttackPowerLevelupPrice())) {

```

```

109         ++attackLevel;
110     }
111 }
112
113 public void upRangeLevel() {
114     if (rangeLevel < maxLevel && iMoneyTransfer.tryMoneyPay(
115         getTurretCollisionRadiusLevelupPrice())) {
116         ++rangeLevel;
117     }
118 }
119
120 public void upRofLevel() {
121     if (rofLevel < maxLevel && iMoneyTransfer.tryMoneyPay(getRofLevelupPrice())) {
122         ++rofLevel;
123     }
124 }
125
126 @Override
127 public BaseTurret clone() {
128     BaseTurret baseTurret = null;
129     baseTurret = (BaseTurret) super.clone();
130     baseTurret.name = new String(this.name);
131     baseTurret.images = this.images.clone();
132     return baseTurret;
133 }
```

---

#### ソースコード 11: uectd.game.gameScene.gameMain.Cannonball

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.gameSystem.GameObject;
5 import uectd.gameSystem.util.CircleCollider;
6 import uectd.gameSystem.util.ImageManager;
7 import uectd.gameSystem.util.Sprite;
8 import uectd.gameSystem.util.Vector2;
9
10 public class Cannonball extends Sprite implements IAttackable {
11     private double attackPower;
12     private double radius;
13     private double cannonballSpeed;
14     private IAttackable attacker;
15     private GameObject enemyParentObject;
16     private BaseEnemy nearestEnemy;
17
18     public Cannonball(GameObject root, GameObject parent, double attackPower, int
19                     landingTime,
20                     CannonTurret CannonTurret, BaseEnemy nearestEnemy) {
21         super(root, parent);
22         this.attackPower = attackPower;
23         this.radius = 10;
24         this.cannonballSpeed = 200;
25         this.collider = new CircleCollider(this, radius);
26         this.attacker = CannonTurret;
27         this.nearestEnemy = nearestEnemy;
28         setImage(ImageManager.getInstance().getImage(ResourcePathDefines.
29                   CANNONBALL_IMAGE_PATH));
30         this.depth = 2;
31     }
32
33     @Override
34     protected void onSummoned() {
35         super.onSummoned();
36     }
37
38     @Override
39     public void calc(float deltaTime) {
40         super.calc(deltaTime);
41         if (nearestEnemy.getEnabled()) {
42             Vector2 diff = Vector2.diff(nearestEnemy.position, this.position);
43             if (diff.norm() < radius * radius) {
44                 nearestEnemy.applyDamage(new Damage(attackPower, this));
45                 var effect = new TurretAttackToEnemyEffect(root, root);
46                 effect.position = nearestEnemy.position.clone();
47             }
48         }
49     }
50 }
```

```

45         root.addChild(effect);
46         this.destroy();
47         return;
48     }
49     Vector2 direction = diff.normalized();
50     position.add(Vector2.scale(direction, cannonballSpeed * deltaTime));
51 } else {
52     destroy();
53 }
54 }
55 }
56 }

```

---

ソースコード 12: uectd.game.gameScene.gameMain.CannonTurret

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
4 import uectd.gameSystem.util.CircleCollider;
5 import uectd.gameSystem.util.GameTimer;
6 import uectd.gameSystem.util.ImageManager;
7 import uectd.game.ResourcePathDefines;
8 import uectd.game.gameScene.Direction;
9
10 import java.awt.event.*;
11
12 public class CannonTurret extends BaseTurret implements ActionListener {
13     private static final double TURRET_RADIUS = 50; // 見た目の大きさ
14     private int attackPowerLevelupPrice;
15     private int turretCollisionRadiusLevelupPrice;
16     private int rofLevelupPrice;
17     private int totalPurchasePrice = 0;
18     private final static double INTERCEPT_ATTACK_POWER = 10;
19     private final static double LEVELUP_ATTACK_POWER = 1;
20     private final static double INTERCEPT_COLLISION_RADIUS = 200;
21     private final static double LEVELUP_COLLISION_RADIUS = 12;
22     private final static int LEVELUP_LANDING_TIME = 50;
23     private final static int LEVELUP_PRICE = 10;
24     private final static int SALE_PRICE = 10;
25     private SearchEnemyField searchEnemyField;
26     private GameTimer timer;
27
28     public CannonTurret(GameObject root, GameObject parent, IMoneyTransfer
29                         iMoneyTransfer) {
30         super(root, parent, "Cannon", INTERCEPT_COLLISION_RADIUS +
31               LEVELUP_COLLISION_RADIUS,
32               INTERCEPT_ATTACK_POWER + LEVELUP_ATTACK_POWER, 1000, 10, 100,
33               ImageManager.getInstance().getDivImage(ResourcePathDefines.
34                           CANNON_TURRET_IMAGES, 8, 1, 64, 64),
35               iMoneyTransfer);
36         depth = 1;
37     }
38
39     @Override
40     public void onDestroyed() {
41         super.onDestroyed();
42         timer.stop();
43     }
44
45     @Override
46     protected void onSummoned() {
47         super.onSummoned();
48         this.collider = new CircleCollider(this, TURRET_RADIUS);
49         searchEnemyField = new SearchEnemyField(root, this);
50         searchEnemyField.setRadius(getTurretCollisionRadius());
51         this.addChild(searchEnemyField);
52         searchEnemyField.position = this.position;
53         this.timer = new GameTimer(getRof(), this);
54         this.timer.start();
55     }
56
57     @Override
58     public void upAttackLevel() {
59         super.upAttackLevel();

```

```

57     attackPower = INTERCEPT_ATTACK_POWER + LEVELUP_ATTACK_POWER * getAttackLevel();
58 }
59
60 @Override
61 public void upRangeLevel() {
62     super.upRangeLevel();
63     System.out.print(turretCollisionRadius + " ");
64     turretCollisionRadius = INTERCEPT_COLLISION_RADIUS + LEVELUP_COLLISION_RADIUS *
65         getRangeLevel();
66     System.out.println(turretCollisionRadius);
67     searchEnemyField.setRadius(turretCollisionRadius);
68 }
69
70 @Override
71 public void upRofLevel() {
72     int prevRofLevel = getRofLevel();
73     super.upRofLevel();
74     if (prevRofLevel != getRofLevel()) {
75         rof = (int) (1000 - LEVELUP_LANDING_TIME * getRofLevel());
76         timer.pause();
77         int nextInitialDelay = timer.getInitialDelay();
78         timer.stop();
79         timer = new GameTimer(rof, this);
80         timer.setInitialDelay(nextInitialDelay);
81         timer.start();
82     }
83 }
84
85 @Override
86 public int getAttackPowerLevelupPrice() {
87     attackPowerLevelupPrice = 20 + LEVELUP_PRICE * getAttackLevel();
88     totalPurchasePrice += attackPowerLevelupPrice;
89     return attackPowerLevelupPrice;
90 }
91
92 @Override
93 public int getTurretCollisionRadiusLevelupPrice() {
94     turretCollisionRadiusLevelupPrice = 20 + LEVELUP_PRICE * getRangeLevel();
95     totalPurchasePrice += turretCollisionRadiusLevelupPrice;
96     return turretCollisionRadiusLevelupPrice;
97 }
98
99 @Override
100 public int getRofLevelupPrice() {
101     rofLevelupPrice = 20 + LEVELUP_PRICE * getRofLevel();
102     totalPurchasePrice += rofLevelupPrice;
103     return rofLevelupPrice;
104 }
105
106 @Override
107 public int getSalePrice() {
108     return (purchasePrice + totalPurchasePrice) * 8 / 10;
109 }
110
111 public void fireCannonball() {
112     BaseEnemy nearestEnemy = searchEnemyField.searchNearestEnemy();
113
114     // enemy の座標に Cannonball を置く
115     if (nearestEnemy != null) {
116         Cannonball cannonball = new Cannonball(root, root, attackPower, rof, this,
117             nearestEnemy);
118         cannonball.position = this.position.clone();
119         root.addChild(cannonball);
120         setImage(images[Direction.convertAngleToConstant(
121             Math.atan2(nearestEnemy.position.y - position.y, nearestEnemy.position.x -
122             position.x))]);
123     }
124 }
125
126 @Override
127 public void actionPerformed(ActionEvent e) {
128     if (e.getSource() == this.timer)
129         this.fireCannonball();

```

```

129     @Override
130     public void pause() {
131         super.pause();
132         timer.pause();
133     }
134
135     @Override
136     public void unpause() {
137         super.unpause();
138         timer.unpause();
139     }
140
141     @Override
142     public void onSelected() {
143         super.onSelected();
144         searchEnemyField.setDisplayFlag(true);
145     }
146
147     @Override
148     public void onUnSelected() {
149         super.onUnSelected();
150         searchEnemyField.setDisplayFlag(false);
151     }
152 }

```

---

ソースコード 13: uectd.game.gameScene.gameMain.Damage

```

1 package uectd.game.gameScene.gameMain;
2
3 public class Damage {
4
5     public double attack;
6     public IAttackable attacker;
7
8     public Damage(double attack, IAttackable attacker) {
9         this.attack = attack;
10        this.attacker = attacker;
11    }
12 }

```

---

ソースコード 14: uectd.game.gameScene.gameMain.enemy.Cyclopes

```

1 package uectd.game.gameScene.gameMain.enemy;
2
3 import java.util.ArrayList;
4
5 import uectd.game.ResourcePathDefines;
6 import uectd.game.gameScene.Direction;
7 import uectd.game.gameScene.gameMain.BaseEnemy;
8 import uectd.game.gameScene.gameMain.Tower;
9 import uectd.game.gameScene.gameMain.GameLevel;
10 import uectd.game.gameScene.gameMain.Graph;
11 import uectd.gameSystem.GameObject;
12 import uectd.gameSystem.util.CircleCollider;
13 import uectd.gameSystem.util.ImageManager;
14
15 public class Cyclopes extends BaseEnemy {
16
17     private final static float MOVE_SPEED = 45;
18     private final static int TIME_PER_FRAME = 500;
19     private static final double ATTACK_POWER = 75;
20     private static final double DEFAULT_HP = 175;
21     private final static int[][] IMAGE_INDEXES = { { 0, 1, 2, 1 }, { 3, 4, 5, 4 }, { 6, 7, 8,
22             7 }, { 9, 10, 11, 10 },
23             { 12, 13, 14, 13 }, { 15, 16, 17, 16 }, { 18, 19, 20, 19 }, { 21, 22, 23, 22 } };
24
25     private int currentAngle;
26
27     public Cyclopes(GameObject root, GameObject parent, GameLevel level, ArrayList<Tower>
28             targets,
29             Graph.Vertex initialVertex) {
30         super(root, parent, level, targets, initialVertex);

```

```

29     this.mover = new SearchAndDestroyStrategy(this, level, targets, MOVE_SPEED, 1000);
30     this.setImages(ImageManager.getInstance().getDivImage(ResourcePathDefines.
31         ENEMY3_IMAGES, 6, 4, 48, 48),
32         TIME_PER_FRAME);
33     this.hp = DEFAULT_HP;
34     this.attackPower = ATTACK_POWER;
35     this.setImageIndexes(IMAGE_INDEXES[0]);
36     this.speed = MOVE_SPEED;
37     this.startAnimation();
38     this.collider = new CircleCollider(this, 10);
39     this.depth = 3;
40     this.dropValue = 120;
41     this.dropScore = 300;
42   }
43
44   @Override
45   public void start() {
46     super.start();
47     this.startAnimation();
48   }
49
50   @Override
51   public void calc(float deltaTime) {
52     super.calc(deltaTime);
53     int angle = Direction.convertAngleToConstant(mover.getAngle());
54     if (angle != currentAngle) {
55       currentAngle = angle;
56       this.setImageIndexes(IMAGE_INDEXES[angle]);
57     }
58   }
59 }
```

---

#### ソースコード 15: uectd.game.gameScene.gameMain.enemy.EnemyAnimation

```

1 package uectd.game.gameScene.gameMain.enemy;
2
3 import uectd.gameSystem.util.Vector2;
4
5 public class EnemyAnimation {
6   private long startTime;
7   private int period, moveStartTime;
8   private double moveLength;
9
10  public EnemyAnimation(int period, int moveStartTime, double moveLength) {
11    this.period = period;
12    this.moveStartTime = moveStartTime;
13    this.moveLength = moveLength;
14  }
15
16  public void start() {
17    startTime = System.currentTimeMillis();
18  }
19
20  public Vector2 getOffset(double angle) {
21    int time = (int) ((System.currentTimeMillis() - startTime) % period);
22    if (time < moveStartTime) {
23      return new Vector2();
24    }
25    double r = moveLength * Math.sin((time - moveStartTime) * Math.PI / (period -
26      moveStartTime));
27    return new Vector2(r * Math.cos(angle), r * Math.sin(angle));
28  }
}
```

---

#### ソースコード 16: uectd.game.gameScene.gameMain.enemy.Goblin

```

1 package uectd.game.gameScene.gameMain.enemy;
2
3 import java.util.ArrayList;
4
5 import uectd.game.ResourcePathDefines;
6 import uectd.game.gameScene.Direction;
```

```

7 import uectd.game.gameScene.gameMain.BaseEnemy;
8 import uectd.game.gameScene.gameMain.Tower;
9 import uectd.game.gameScene.gameMain.GameLevel;
10 import uectd.game.gameScene.gameMain.Graph;
11 import uectd.gameSystem.GameObject;
12 import uectd.gameSystem.util.CircleCollider;
13 import uectd.gameSystem.util.ImageManager;
14
15 public class Goblin extends BaseEnemy {
16
17     private final static float MOVE_SPEED = 80;
18     private final static int TIME_PER_FRAME = 500;
19     private static final double ATTACK_POWER = 15;
20     private static final double DEFAULT_HP = 35;
21
22     private final static int[][] IMAGE_INDEXES = { { 0, 1, 2, 1 }, { 3, 4, 5, 4 }, { 6, 7, 8,
23         7 }, { 9, 10, 11, 10 },
24         { 12, 13, 14, 13 }, { 15, 16, 17, 16 }, { 18, 19, 20, 19 }, { 21, 22, 23, 22 } };
25
26     private int currentAngle;
27
28     public Goblin(GameObject root, GameObject parent, GameLevel level, ArrayList<Tower>
29         targets,
30         Graph.Vertex initialVertex) {
31         super(root, parent, level, targets, initialVertex);
32         this.mover = new SearchAndDestroyStrategy(this, level, targets, MOVE_SPEED, 1000);
33         this.setImages(ImageManager.getInstance().getDivImage(ResourcePathDefines.
34             GOBLIN_IMAGES, 6, 4, 48, 48),
35             TIME_PER_FRAME);
36         this.hp = DEFAULT_HP;
37         this.attackPower = ATTACK_POWER;
38         this.setImageIndexes(IMAGE_INDEXES[0]);
39         this.speed = MOVE_SPEED;
40         this.startAnimation();
41         this.collider = new CircleCollider(this, 10);
42         this.depth = 3;
43         this.dropValue = 30;
44         this.dropScore = 200;
45     }
46
47     @Override
48     public void start() {
49         super.start();
50         this.startAnimation();
51     }
52
53     @Override
54     public void calc(float deltaTime) {
55         super.calc(deltaTime);
56         int angle = Direction.convertAngleToConstant(mover.getAngle());
57         if (angle != currentAngle) {
58             currentAngle = angle;
59             this.setImageIndexes(IMAGE_INDEXES[angle]);
60         }
61     }
62 }

```

ソースコード 17: uectd.game.gameScene.gameMain.enemy.Hobgoblin

---

```

1 package uectd.game.gameScene.gameMain.enemy;
2
3 import java.util.ArrayList;
4
5 import uectd.game.ResourcePathDefines;
6 import uectd.game.gameScene.Direction;
7 import uectd.game.gameScene.gameMain.BaseEnemy;
8 import uectd.game.gameScene.gameMain.Tower;
9 import uectd.game.gameScene.gameMain.GameLevel;
10 import uectd.game.gameScene.gameMain.Graph;
11 import uectd.gameSystem.GameObject;
12 import uectd.gameSystem.util.CircleCollider;
13 import uectd.gameSystem.util.ImageManager;
14
15 public class Hobgoblin extends BaseEnemy {

```

```

16     private final static float MOVE_SPEED = 75;
17     private final static int TIME_PER_FRAME = 500;
18     private static final double ATTACK_POWER = 30;
19     private static final double DEFAULT_HP = 65;
20
21     private final static int[][] IMAGE_INDEXES = { { 0, 1, 2, 1 }, { 3, 4, 5, 4 }, { 6, 7, 8,
22         7 }, { 9, 10, 11, 10 },
23         { 12, 13, 14, 13 }, { 15, 16, 17, 16 }, { 18, 19, 20, 19 }, { 21, 22, 23, 22 } };
24
25     private int currentAngle;
26
27     public Hobgoblin(GameObject root, GameObject parent, GameLevel level, ArrayList<Tower
28         > targets,
29         Graph.Vertex initialVertex) {
30         super(root, parent, level, targets, initialVertex);
31         this.mover = new SearchAndDestroyStrategy(this, level, targets, MOVE_SPEED, 1000);
32         this.setImages(ImageManager.getInstance().getDivImage(ResourcePathDefines.
33             ENEMY2_IMAGES, 6, 4, 48, 48),
34             TIME_PER_FRAME);
35         this.hp = DEFAULT_HP;
36         this.attackPower = ATTACK_POWER;
37         this.setImageIndexes(IMAGE_INDEXES[0]);
38         this.speed = MOVE_SPEED;
39         this.startAnimation();
40         this.collider = new CircleCollider(this, 10);
41         this.depth = 3;
42         this.dropValue = 70;
43         this.dropScore = 200;
44     }
45
46     @Override
47     public void start() {
48         super.start();
49         this.startAnimation();
50     }
51
52     @Override
53     public void calc(float deltaTime) {
54         super.calc(deltaTime);
55         int angle = Direction.convertAngleToConstant(mover.getAngle());
56         if (angle != currentAngle) {
57             currentAngle = angle;
58             this.setImageIndexes(IMAGE_INDEXES[angle]);
59         }
60     }

```

---

ソースコード 18: uectd.game.gameScene.gameMain.enemy.SearchAndDestroyStrategy

```

1 package uectd.game.gameScene.gameMain.enemy;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.Stack;
6 import java.awt.event.*;
7
8 import uectd.game.gameScene.gameMain.BaseEnemyActStrategy;
9 import uectd.game.gameScene.gameMain.Tower;
10 import uectd.game.gameScene.gameMain.Damage;
11 import uectd.game.gameScene.gameMain.GameLevel;
12 import uectd.game.gameScene.gameMain.Graph;
13 import uectd.game.gameScene.gameMain.TowerFallListener;
14 import uectd.gameSystem.GameObject;
15 import uectd.gameSystem.util.GameTimer;
16 import uectd.gameSystem.util.Vector2;
17
18 public class SearchAndDestroyStrategy extends BaseEnemyActStrategy implements
19     TowerFallListener, ActionListener {
20
21     private enum Mode {
22         search, destroy,
23     };

```

```

24     private static final double THRESHOLD_RADIUS = 10;
25     private Mode currentMode;
26     private Stack<Graph.Vertex> route;
27     private int attackInterval;
28     private ArrayList<Tower> targets;
29     private Tower target;
30     private GameTimer attackTimer;
31     private EnemyAnimation animation;
32     private double angle;
33
34     @Override
35     public SearchAndDestroyStrategy clone() {
36         SearchAndDestroyStrategy sdStrategy = null;
37         sdStrategy = (SearchAndDestroyStrategy) super.clone();
38         sdStrategy.attackTimer = new GameTimer(attackInterval, sdStrategy);
39         sdStrategy.targets = new ArrayList<>(targets);
40         sdStrategy.animation = new EnemyAnimation(attackInterval, attackInterval * 7 / 10,
41             30);
42         return sdStrategy;
43     }
44
45     @Override
46     public void onSummoned() {
47         Collections.shuffle(targets);
48         if (!targets.isEmpty())
49             target = targets.get(0);
50         animation.start();
51
52         for (var tower : targets) {
53             tower.addTowerFallListener(this);
54         }
55         enemyGameObject.start();
56     }
57
58     @Override
59     public void pause() {
60         super.pause();
61         attackTimer.pause();
62     }
63
64     @Override
65     public void unpause() {
66         super.unpause();
67         attackTimer.unpause();
68     }
69
70     public SearchAndDestroyStrategy(GameObject enemyGameObject, GameLevel level,
71         ArrayList<Tower> targets, float speed,
72         int attackInterval) {
73         super(enemyGameObject, level, targets);
74         this.currentMode = Mode.search;
75         this.route = new Stack<>();
76         this.targets = new ArrayList<>(targets);
77         this.attackInterval = attackInterval;
78         if (!this.targets.isEmpty())
79             this.target = this.targets.get(0);
80         this.attackTimer = new GameTimer(attackInterval, this);
81         this.animation = new EnemyAnimation(attackInterval, attackInterval * 7 / 10, 30);
82     }
83
84     @Override
85     public void calc(float deltaTime) {
86         if (target == null)
87             return;
88         if (currentMode == Mode.search) {
89             if (route.isEmpty()) {
90                 route = level.graph.shortestPath(level.graph.nearestVertex(enemyGameObject.
91                     position),
92                     target.getVertex());
93             }
94             var nextDst = route.peek();
95             var vector = Vector2.diff(nextDst.position, enemyGameObject.position);
96             if (vector.magnitude() < THRESHOLD_RADIUS) {
97                 route.pop();
98                 if (route.isEmpty()) {

```

```

96         currentMode = Mode.destroy;
97         attackTimer.start();
98         animation.start();
99         vector = Vector2.diff(nextDst.position, enemyGameObject.position);
100        return;
101    }
102    nextDst = route.peek();
103    vector = Vector2.diff(enemyGameObject.position, nextDst.position);
104 }
105 angle = Math.atan2(vector.y, vector.x);
106 enemyGameObject.position.add(Vector2.scale(vector.normalized(), deltaTime *
107     enemyGameObject.getSpeed()));
108 } else if (currentMode == Mode.destroy) {
109     enemyGameObject.offset = animation.getOffset(angle);
110 }
111
112 @Override
113 public void onTowerFall(Tower tower) {
114     attackTimer.stop();
115     route.clear();
116     currentMode = Mode.search;
117     target.removeTowerFallListener(this);
118     targets.remove(target);
119     if (tower == target) {
120         if (!targets.isEmpty())
121             target = targets.get(0);
122     }
123 }
124
125 @Override
126 public void actionPerformed(ActionEvent e) {
127     animation.start();
128     target.applyDamage(new Damage(enemyGameObject.getAttackPower(), enemyGameObject
129         ));
130 }
131 public double getAngle() {
132     return angle;
133 }
134
135 @Override
136 public void onDestroyed() {
137     this.attackTimer.stop();
138     for (var target : targets) {
139         target.removeTowerFallListener(this);
140     }
141 }
142 }

```

---

ソースコード 19: uectd.game.gameScene.gameMain.enemy.Zombie

```

1 package uectd.game.gameScene.gameMain.enemy;
2
3 import java.util.ArrayList;
4
5 import uectd.game.ResourcePathDefines;
6 import uectd.game.gameScene.Direction;
7 import uectd.game.gameScene.gameMain.BaseEnemy;
8 import uectd.game.gameScene.gameMain.Tower;
9 import uectd.game.gameScene.gameMain.GameLevel;
10 import uectd.game.gameScene.gameMain.Graph;
11 import uectd.gameSystem.GameObject;
12 import uectd.gameSystem.util.CircleCollider;
13 import uectd.gameSystem.util.ImageManager;
14
15 public class Zombie extends BaseEnemy {
16
17     private final static float MOVE_SPEED = 60;
18     private final static int TIME_PER_FRAME = 500;
19     private static final double ATTACK_POWER = 20;
20     private static final double DEFAULT_HP = 50;
21     private final static int[][] IMAGE_INDEXES = { { 0, 1, 2, 1 }, { 3, 4, 5, 4 }, { 6, 7, 8,
22         7 }, { 9, 10, 11, 10 },
23         { 12, 13, 14, 13 }, { 15, 16, 17, 16 }, { 18, 19, 20, 19 }, { 21, 22, 23, 22 } };

```

```

23     private int currentAngle;
24
25     public Zombie(GameObject root, GameObject parent, GameLevel level, ArrayList<Tower>
26         targets,
27         Graph.Vertex initialVertex) {
28         super(root, parent, level, targets, initialVertex);
29         this.mover = new SearchAndDestroyStrategy(this, level, targets, MOVE_SPEED, 1000);
30         this.setImages(ImageManager.getInstance().getDivImage(ResourcePathDefines.
31             ZOMBIE_IMAGES, 6, 4, 48, 48),
32             TIME_PER_FRAME);
33         this.hp = DEFAULT_HP;
34         this.attackPower = ATTACK_POWER;
35         this.setImageIndexes(IMAGE_INDEXES[0]);
36         this.speed = MOVE_SPEED;
37         this.startAnimation();
38         this.collider = new CircleCollider(this, 10);
39         this.depth = 3;
40         this.dropValue = 20;
41         this.dropScore = 100;
42     }
43
44     @Override
45     public void start() {
46         super.start();
47         this.startAnimation();
48     }
49
50     @Override
51     public void calc(float deltaTime) {
52         super.calc(deltaTime);
53         int angle = Direction.convertAngleToConstant(mover.getAngle());
54         if (angle != currentAngle) {
55             currentAngle = angle;
56             this.setImageIndexes(IMAGE_INDEXES[angle]);
57         }
58     }
59 }
```

---

ソースコード 20: uectd.game.gameScene.gameMain.EnemyDieEffect

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.gameSystem.GameObject;
5 import uectd.gameSystem.util.Effect;
6 import uectd.gameSystem.util.ImageManager;
7 import uectd.gameSystem.util.SoundManager;
8 import uectd.gameSystem.util.Vector2;
9
10 public class EnemyDieEffect extends Effect {
11
12     private final static int X_NUM = 5;
13     private final static int Y_NUM = 2;
14     private final static int X_SIZE = 192;
15     private final static int Y_SIZE = 192;
16
17     public EnemyDieEffect(GameObject root, GameObject parent) {
18         super(root, parent, ImageManager.getInstance().getDivImage(ResourcePathDefines.
19             ENEMY_DIE_EFFECT_IMAGES, X_NUM,
20             Y_NUM, X_SIZE, Y_SIZE), false, 0.03f, 0);
21         setDrawSize(new Vector2(80, 80));
22         SoundManager.getInstance().play(ResourcePathDefines.ENEMY_DIE_SOUND_PATH);
23     }
24 }
```

---

ソースコード 21: uectd.game.gameScene.gameMain.EnemyDieListener

```

1 package uectd.game.gameScene.gameMain;
2
3 public interface EnemyDieListener {
```

```
4     void onEnemyDead(BaseEnemy enemy);
5 }
```

---

#### ソースコード 22: uectd.game.gameScene.gameMain.EnemyParent

```
1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
4
5 public class EnemyParent extends GameObject {
6
7     public EnemyParent(GameObject root, GameObject parent) {
8         super(root, parent);
9     }
10
11 }
```

---

#### ソースコード 23: uectd.game.gameScene.gameMain.EnemySpawner

```
1 package uectd.game.gameScene.gameMain;
2
3 import java.awt.event.*;
4
5 import uectd.game.gameScene.gameMain.Graph.Vertex;
6 import uectd.gameSystem.GameObject;
7 import uectd.gameSystem.util.*;
8
9 public class EnemySpawner extends GameObject implements ActionListener {
10     private int spawnCount; // 敵出現数。
11     private int initialDelay; // Wave 開始から敵出現開始までの遅延(ms)
12     private int spawnDelay; // 敵出現間隔(ms)
13     private GameTimer timer; // 出現待ちに用いるゲーム内タイマー
14     private BaseEnemy spawnEnemy; // 出現敵種
15     private Vertex initialVertex; // 敵出現地点
16     private EnemyParent enemyParent; // 出現した敵が属するゲーム内オブジェクトのヒエラルキー
17     の親
18     private EnemyDieListener enemyDieListener; // エネミー死亡リスナー
19
20     public EnemySpawner(GameObject root, GameObject parent, int initialDelay, int
21         spawnDelay, int spawnCount,
22         BaseEnemy spawnEnemy, Vertex initialVertex, EnemyParent enemyParent,
23         EnemyDieListener enemyDieListener) {
24         super(root, parent);
25         this.initialDelay = initialDelay;
26         this.spawnDelay = spawnDelay;
27         this.spawnCount = spawnCount;
28         this.spawnEnemy = spawnEnemy;
29         this.initialVertex = initialVertex;
30         this.enemyParent = enemyParent;
31         this.enemyDieListener = enemyDieListener;
32     }
33
34     public void setSpawnCount(int spawnCount) {
35         this.spawnCount = spawnCount;
36     }
37
38     public void setInitialDelay(int initialDelay) {
39         this.initialDelay = initialDelay;
40     }
41
42     @Override
43     public void onEnabled() {
44         // スポナーがenableされた場合の処理
45         this.timer = new GameTimer(this.spawnDelay, this);
46         this.timer.setInitialDelay(this.initialDelay);
47         this.timer.start();
48     }
49
50 }
```

```

51     @Override
52     public void actionPerformed(ActionEvent e) {
53         if (spawnCount > 0) {
54             // 一體ごと出現
55             BaseEnemy enemy = spawnEnemy.clone();
56             enemy.position = this.position.clone();
57             enemy.initialVertex = this.initialVertex;
58             enemy.addEnemyDieListener(enemyDieListener);
59             enemyParent.addChild(enemy);
60             var effect = new EnemySpawnerEffect(root, root);
61             effect.position = this.position.clone();
62             root.addChild(effect);
63             spawnCount--;
64         } else {
65             this.setEnabled(false);
66             timer.stop();
67         }
68     }
69
70     @Override
71     public void pause() {
72         super.pause();
73         timer.pause();
74     }
75
76     @Override
77     public void unpause() {
78         super.unpause();
79         timer.unpause();
80     }
81
82     @Override
83     public void onDestroyed() {
84         super.onDestroyed();
85         timer.stop();
86     }
87 }
```

---

ソースコード 24: uectd.game.gameScene.gameMain.EnemySpawnerArrangementStrategy

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.util.ArrayList;
4
5 import uectd.game.gameScene.gameMain.Graph.Vertex;
6 import uectd.gameSystem.GameObject;
7
8 public abstract class EnemySpawnerArrangementStrategy {
9     protected ArrayList<Vertex> initialVertexList;
10    protected GameObject rootGameObject;
11    protected GameObject parentGameObject;
12    protected EnemySpawnerParent enemySpawnerParent;
13    protected EnemyParent enemyParent;
14    protected GameLevel level;
15    protected ArrayList<Tower> targets;
16
17    public abstract void arrange(int wave);
18 }
```

---

ソースコード 25: uectd.game.gameScene.gameMain.EnemySpawnerEffect

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.gameSystem.GameObject;
5 import uectd.gameSystem.util.Effect;
6 import uectd.gameSystem.util.ImageManager;
7 import uectd.gameSystem.util.SoundManager;
8 import uectd.gameSystem.util.Vector2;
9
10 public class EnemySpawnerEffect extends Effect {
11     private final static int X_NUM = 10;
12     private final static int Y_NUM = 1;
```

```

13     private final static int X_SIZE = 240;
14     private final static int Y_SIZE = 240;
15
16     public EnemySpawnerEffect(GameObject root, GameObject parent) {
17         super(root, parent, ImageManager.getInstance().getDivImage(ResourcePathDefines.
18             ENEMY_SUMMON_IMAGES, X_NUM,
19             Y_NUM, X_SIZE, Y_SIZE), false, 0.03f, 0);
20         setDrawSize(new Vector2(80, 80));
21         this.offset.y = -25;
22         SoundManager.getInstance().play(ResourcePathDefines.SUMMON_SOUND_PATH);
23         this.depth = 1;
24     }
25 }

```

---

ソースコード 26: uectd.game.gameScene.gameMain.EnemySpawnerManager

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.util.ArrayList;
6 import java.util.Collections;
7 import java.util.Scanner;
8
9 import uectd.game.gameScene.gameMain.Graph.Vertex;
10 import uectd.gameSystem.FatalError;
11 import uectd.gameSystem.GameObject;
12
13 // GameObject ではない
14 public class EnemySpawnerManager {
15     private EnemySpawnerArrangementStrategy spawnerStrategy;
16     private static final int MAX_VERTEX_NUM = 3;
17
18     public EnemySpawnerManager(Graph graph, EnemySpawnerArrangementStrategy
19         spawnerStrategy,
20         String candidateVertexFilePath, GameObject rootGameObject, GameObject
21         parentGameObject,
22         EnemySpawnerParent enemySpawnerParent, EnemyParent enemyParent, GameLevel
23         level, ArrayList<Tower> targets) {
24         this.spawnerStrategy = spawnerStrategy;
25         spawnerStrategy.rootGameObject = rootGameObject;
26         spawnerStrategy.parentGameObject = parentGameObject;
27         spawnerStrategy.enemySpawnerParent = enemySpawnerParent;
28         spawnerStrategy.enemyParent = enemyParent;
29         spawnerStrategy.level = level;
30         spawnerStrategy.targets = targets;
31
32         // 3つくらい頂点を抽出
33         // 頂点の候補 (どの頂点番号が敵出現位置になり得るか)は外部ファイル
34
35         // 敵出現場所の候補が入ったファイル -> candidateVertexList に最大 3つまで頂点を入れる
36         // ファイルに書いてある頂点の個数が 3つより少ない場合は、1 or 2個しか入らない
37         try {
38             File candidateVertexFile = new File(candidateVertexFilePath);
39             if (candidateVertexFile.exists() && candidateVertexFile.isFile() &&
40                 candidateVertexFile.canRead()) {
41                 Scanner sc = new Scanner(candidateVertexFile);
42                 int n = sc.nextInt(); // 入力行数。
43                 var candidateVertexList = new ArrayList<Vertex>();
44                 for (int i = 0; i < n; i++) {
45                     int idx = sc.nextInt();
46                     candidateVertexList.add(graph.vertices.get(idx));
47                 }
48                 sc.close();
49                 // ファイル閉じてここから頂点を決定
50                 Collections.shuffle(candidateVertexList);
51                 var vertexList = new ArrayList<>(
52                     candidateVertexList.subList(0, Math.min(MAX_VERTEX_NUM,
53                         candidateVertexList.size())));
54                 // 先頭 3要素のリストを作成 (もし
55                 // candidate が少なければ 3つより少なくなる)
56                 spawnerStrategy.initialVertexList = vertexList; //
57                     // strategy に初期スポ位置を渡す
58                 for (Vertex vertex : vertexList) {

```

```

51         var spawnerMark = new EnemySpawnerMark(rootGameObject, rootGameObject);
52         spawnerMark.position = vertex.position;
53         rootGameObject.addChild(spawnerMark);
54     }
55 } else {
56     FatalError.quit("敵情報ファイルが見つからないか開けません");
57 }
58 } catch (FileNotFoundException e) {
59     FatalError.quit("敵情報ファイルが存在しません");
60 }
61 }
62 // wave終了時にスポナーは全消し → spawnerParent の child をまとめて殺せばok
63 // wav開始時に、Model側で、スポナー生成関数を呼び出す。
64
65 public void arrange(int wave) {
66     spawnerStrategy.arrange(wave);
67 }
68
69 }
70 }
```

---

ソースコード 27: uectd.game.gameScene.gameMain.EnemySpawnerMark

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.awt.Graphics;
4 import java.awt.geom.AffineTransform;
5
6 import uectd.game.ResourcePathDefines;
7 import uectd.gameSystem.GameObject;
8 import uectd.gameSystem.util.Drawable;
9 import uectd.gameSystem.util.ImageManager;
10 import uectd.gameSystem.util.Vector2;
11
12 import java.awt.*;
13
14 public class EnemySpawnerMark extends Drawable {
15     Image image;
16     int width, height;
17
18     public EnemySpawnerMark(GameObject root, GameObject parent) {
19         super(root, parent);
20         image = ImageManager.getInstance().getImage(ResourcePathDefines.
21             ENEMY_SPAWNER_MARK_IMAGE);
22         width = image.getWidth(null);
23         height = image.getHeight(null);
24         this.depth = 9;
25     }
26
27     @Override
28     public void draw(Graphics g, Vector2 screenPosition, float ratio) {
29         Graphics2D g2 = (Graphics2D) g;
30         AffineTransform originalAffineTransform = g2.getTransform();
31         AffineTransform affineTransform = (AffineTransform) originalAffineTransform.clone();
32         affineTransform.rotate(System.currentTimeMillis() / 2000.0, screenPosition.x,
33             screenPosition.y);
34         g2.setTransform(affineTransform);
35         g2.drawImage(this.image, (int) (screenPosition.x - width * ratio * 0.5),
36             (int) (screenPosition.y - height * ratio * 0.5), (int) (width * ratio),
37             (int) (height * ratio), null);
38         g2.setTransform(originalAffineTransform); // アフィン変換行列を元に戻す
39     }
40 }
```

---

ソースコード 28: uectd.game.gameScene.gameMain.EnemySpawnerParent

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
4
5 public class EnemySpawnerParent extends GameObject {
```

```

6     public EnemySpawnerParent(GameObject root, GameObject parent) {
7         super(root, parent);
8     }
9 }
10
11 }

```

---

ソースコード 29: uectd.game.gameScene.gameMain.FileReadSpawnStrategy

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.util.NoSuchElementException;
6 import java.util.Scanner;
7
8 import uectd.game.ResourcePathDefines;
9 import uectd.game.gameScene.gameMain.enemy.Zombie;
10 import uectd.gameSystem.FatalError;
11 import uectd.game.gameScene.gameMain.enemy.Goblin;
12 import uectd.game.gameScene.gameMain.enemy.Hobgoblin;
13 import uectd.game.gameScene.gameMain.enemy.Cyclopes;
14
15 public class FileReadSpawnStrategy extends EnemySpawnerArrangementStrategy {
16     private EnemyDieListener enemyDieListener;
17
18     public FileReadSpawnStrategy(EnemyDieListener enemyDieListener) {
19         this.enemyDieListener = enemyDieListener;
20     }
21
22     @Override
23     public void arrange(int wave) { // W数に対応する spawner の設置を行う。
24
25         try {
26             File spawnerDetailFile = new File(ResourcePathDefines.
27                 ENEMY_SPAWNER_DEFINES_PATH);
28             if (spawnerDetailFile.exists() && spawnerDetailFile.isFile() &&
29                 spawnerDetailFile.canRead()) {
30                 Scanner sc = new Scanner(spawnerDetailFile);
31                 try {
32                     int loadedWave = sc.nextInt(), n = sc.nextInt(); // W数とその Wでの入力行数
33                     while (wave != loadedWave) {
34                         for (int i = 0; i < n; i++) {
35                             for (int j = 0; j < 5; j++)
36                                 sc.nextInt();
37                             loadedWave = sc.nextInt();
38                             n = sc.nextInt();
39                         }
40                         // この時点で、loadedWave==wave、nはその Wでの入力行数 になっている
41                         for (int i = 0; i < n; i++) {
42                             EnemySpawner spawner = generateSpawner(sc.nextInt(), sc.nextInt(),
43                                 sc.nextInt(),
44                                 sc.nextInt());
45                             enemySpawnerParent.addChild(spawner);
46                         }
47
48                     } catch (NoSuchElementException e) {
49                         e.printStackTrace();
50                         FatalError.quit("該当するウェーブデータが存在しません");
51                     }
52                     sc.close();
53                     // ファイル閉じてここから頂点を決定
54                 } else {
55                     FatalError.quit("ウェーブデータファイルが見つからないか開けません");
56                 }
57             } catch (FileNotFoundException e) {
58                 e.printStackTrace();
59                 FatalError.quit("ウェーブデータファイルが存在しません");
60             }
61         } // [csv のフォーマット]
62     }

```

```

61 // Wave数 今回Waveの入力行数
62 // initialVertexNum initialDelay spawnDelay spawnCount spawnEnemy
63
64 // Wごとの spawner の定義を記述したファイル(EnemySpawnerDefines.dat)の、連続した 5つの値を
65 // 読み込んで、spawner を生成する
66 private EnemySpawner generateSpawner(int initialVertexNum, int initialDelay, int
67     spawnDelay, int spawnCount,
68     int spawnEnemyNum) {
69     BaseEnemy spawnEnemy;
70     switch (spawnEnemyNum) { // ここで、
71         enemy の種類を番号から識別する。enemy 追加等で書き換えが必要となる箇所なので注意！
72         case 0:
73             spawnEnemy = new Zombie(rootGameObject, enemyParent, level, targets,
74                 initialVertexList.get(Math.min(initialVertexNum, initialVertexList.
75                     size())));
76             break;
77         case 1:
78             spawnEnemy = new Goblin(rootGameObject, enemyParent, level, targets,
79                 initialVertexList.get(Math.min(initialVertexNum, initialVertexList.
80                     size())));
81             break;
82         case 2:
83             spawnEnemy = new Hobgoblin(rootGameObject, enemyParent, level, targets,
84                 initialVertexList.get(Math.min(initialVertexNum, initialVertexList.
85                     size())));
86             break;
87         case 3:
88             spawnEnemy = new Cyclopes(rootGameObject, enemyParent, level, targets,
89                 initialVertexList.get(Math.min(initialVertexNum, initialVertexList.
90                     size())));
91             break;
92         default:
93             spawnEnemy = new Zombie(rootGameObject, enemyParent, level, targets,
94                 initialVertexList.get(Math.min(initialVertexNum, initialVertexList.
95                     size())));
96             break;
97     }
98
99     EnemySpawner spawner = new EnemySpawner(rootGameObject, enemySpawnerParent,
100         initialDelay, spawnDelay,
101         spawnCount, spawnEnemy, initialVertexList.get(initialVertexNum),
102         enemyParent, enemyDieListener);
103     spawner.position = initialVertexList.get(initialVertexNum).position.clone();
104     spawner.setEnabled(true);
105     return spawner;
106 }

```

---

### ソースコード 30: uectd.game.gameScene.gameMain.GameLevel

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.awt.*;
4
5 import uectd.gameSystem.GameObject;
6 import uectd.gameSystem.util.Drawable;
7 import uectd.gameSystem.util.Vector2;
8
9 public class GameLevel extends Drawable {
10
11     public Graph graph;
12
13     public GameLevel(GameObject root, GameObject parent, Graph graph) {
14         super(root, parent);
15         this.graph = graph;
16         this.depth = 9;
17     }
18
19     @Override
20     public void draw(Graphics g, Vector2 screenPosition, float ratio) {
21         Graphics2D g2 = (Graphics2D) g;
22         BasicStroke bs = new BasicStroke(6 * ratio);
23         Color color = new Color(0, 255, 255, 20);
24         g2.setStroke(bs);

```

```

25     g2.setColor(color);
26     for (var edges : graph.data) {
27         for (var edge : edges) {
28             int x1 = (int) ((screenPosition.x + graph.vertices.get(edge.from).position.x
29                             * ratio));
29             int y1 = (int) ((screenPosition.y + graph.vertices.get(edge.from).position.y
30                             * ratio));
30             int x2 = (int) ((screenPosition.x + graph.vertices.get(edge.to).position.x *
31                             ratio));
31             int y2 = (int) ((screenPosition.y + graph.vertices.get(edge.to).position.y *
32                             ratio));
32             g2.drawLine(x1, y1, x2, y2);
33         }
34     }
35     g2.setColor(new Color(240, 240, 40, 80));
36     for (var vertex : graph.vertices) {
37         int x = (int) (screenPosition.x + (vertex.position.x - 5) * ratio);
38         int y = (int) (screenPosition.y + (vertex.position.y - 5) * ratio);
39         int l = (int) (10 * ratio);
40         g2.drawOval(x, y, l, l);
41     }
42 }
43 }
44 }
```

---

ソースコード 31: uectd.game.gameScene.gameMain.Graph

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.util.*;
4
5 import uectd.gameSystem.util.Vector2;
6
7 public class Graph {
8     public class Vertex {
9         public Vector2 position;
10        public int idx;
11        public boolean enabled;
12
13        public Vertex(Vector2 position, int idx) {
14            this.position = position;
15            this.idx = idx;
16            this.enabled = true;
17        }
18    }
19
20    public class Edge {
21        public int from, to;
22        public double dist;
23
24        public Edge(int from, int to, double dist) {
25            this.from = from;
26            this.to = to;
27            this.dist = dist;
28        }
29    }
30
31    public ArrayList<Vertex> vertexes; // 頂点の情報のリスト
32    public ArrayList<ArrayList<Edge>> data; // 隣接リスト表現のマップデータ
33
34    public Graph(int vNum) {
35        vertexes = new ArrayList<>();
36        data = new ArrayList<>();
37        for (int i = 0; i < vNum; i++) {
38            data.add(new ArrayList<>());
39        }
40    }
41
42    public void addVertex(Vector2 position, int idx) {
43        vertexes.add(new Vertex(position, idx));
44    }
45
46    public void addEdge(int from, int to) {
47        double dist = Vector2.diff(vertexes.get(from).position, vertexes.get(to).position
```

```

        ).magnitude();
48    data.get(from).add(new Edge(from, to, dist));
49    data.get(to).add(new Edge(to, from, dist));
50 }
51
52 public static class DistVertexPair implements Comparable<DistVertexPair> {
53     double dist;
54     int vertexIdx;
55
56     public DistVertexPair(double dist, int vertex) {
57         this.dist = dist;
58         this.vertexIdx = vertex;
59     }
60
61     @Override
62     public int compareTo(DistVertexPair o) {
63         return this.dist < o.dist ? -1 : 1;
64     }
65 }
66
67
68 public Vertex nearestVertex(Vector2 position) {
69     if (vertexes.isEmpty())
70         return null;
71     Vertex res = vertexes.get(0);
72     double minDistSquare = Double.POSITIVE_INFINITY;
73     for (var vertex : vertexes) {
74         double distSquare = Vector2.diff(position, vertex.position).norm();
75         if (Vector2.diff(position, vertex.position).norm() < minDistSquare) {
76             res = vertex;
77             minDistSquare = distSquare;
78         }
79     }
80     return res;
81 }
82
83 public Stack<Vertex> shortestPath(Vertex s, Vertex t) {
84     var dist = new double[this.vertexes.size()];
85     var prev = new int[this.vertexes.size()];
86
87     PriorityQueue<DistVertexPair> pq = new PriorityQueue<>();
88     pq.add(new DistVertexPair(0, s.idx));
89     prev[s.idx] = -1;
90     Arrays.fill(dist, Double.POSITIVE_INFINITY);
91     dist[s.idx] = 0.0;
92     while (!pq.isEmpty()) {
93         DistVertexPair current = pq.poll();
94         int current_pos = current.vertexIdx;
95         double current_dist = current.dist;
96         for (Edge edge : data.get(current_pos)) {
97             if (dist[edge.to] > current_dist + edge.dist) {
98                 dist[edge.to] = current_dist + edge.dist;
99                 pq.add(new DistVertexPair(dist[edge.to], edge.to));
100                prev[edge.to] = current_pos;
101            }
102        }
103    }
104    var res = new Stack<Vertex>();
105    int current_pos = t.idx;
106    while (current_pos != -1) {
107        res.push(vertexes.get(current_pos));
108        current_pos = prev[current_pos];
109    }
110    return res;
111 }
112
113 }

```

---

ソースコード 32: uectd.game.gameScene.gameMain.GraphBuilder

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.io.*;
4 import java.util.*;
5

```

```

6 import uectd.game.ResourcePathDefines;
7 import uectd.gameSystem.FatalError;
8 import uectd.gameSystem.util.ImageManager;
9 import uectd.gameSystem.util.Vector2;
10
11 public class GraphBuilder {
12     public static Graph build(String graphPath) {
13         try {
14             File graphFile = new File(graphPath);
15             Graph graph = null;
16             if (graphFile.exists() && graphFile.isFile() && graphFile.canRead()) {
17                 Scanner sc = new Scanner(graphFile);
18                 int v = sc.nextInt();
19                 graph = new Graph(v);
20                 int halfWidth = ImageManager.getInstance().getImage(ResourcePathDefines.
21                             BACKGROUND_IMAGE).getWidth(null)
22                             / 2;
23                 int halfHeight = ImageManager.getInstance().getImage(ResourcePathDefines.
24                             BACKGROUND_IMAGE)
25                             .getHeight(null) / 2;
26                 for (int i = 0; i < v; i++) {
27                     int x = sc.nextInt() - halfWidth, y = sc.nextInt() - halfHeight;
28                     int idx = sc.nextInt();
29                     graph.addVertex(new Vector2(x, y), idx);
30                 }
31                 int e = sc.nextInt();
32                 for (int i = 0; i < e; i++) {
33                     int s = sc.nextInt(), t = sc.nextInt();
34                     graph.addEdge(s, t);
35                 }
36                 sc.close();
37                 return graph;
38             } else {
39                 FatalError.quit("マップデータファイルが見つからないか開けません");
40                 return null;
41             }
42         } catch (FileNotFoundException e) {
43             e.printStackTrace();
44             FatalError.quit("マップデータファイルが存在しません");
45         }
46         return null;
47     }
48 }

```

---

ソースコード 33: uectd.game.gameScene.gameMain.IAttackable

```

1 package uectd.game.gameScene.gameMain;
2
3 public interface IAttackable {
4 }

```

---

ソースコード 34: uectd.game.gameScene.gameMain.IDamageApplicable

```

1 package uectd.game.gameScene.gameMain;
2
3 public interface IDamageApplicable {
4     void applyDamage(Damage d);
5 }

```

---

ソースコード 35: uectd.game.gameScene.gameMain.IMoneyTransfer

```

1 package uectd.game.gameScene.gameMain;
2
3 public interface IMoneyTransfer {
4     void moneyAdd(int amount);
5
6     boolean tryMoneyPay(int amount);
7
8     int getBalance();
9
10 }

```

---

ソースコード 36: uectd.game.gameScene.gameMain.Laser

---

```
1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.gameSystem.GameObject;
5 import uectd.gameSystem.util.CircleCollider;
6 import uectd.gameSystem.util.Drawable;
7 import uectd.gameSystem.util.GameTimer;
8 import uectd.gameSystem.util.ImageManager;
9 import uectd.gameSystem.util.Vector2;
10 import java.awt.Graphics;
11 import java.awt.Graphics2D;
12 import java.awt.geom.AffineTransform;
13
14 import java.awt.*;
15 import java.awt.event.*;
16
17 public class Laser extends Drawable implements IAttackable, EnemyDieListener,
18     ActionListener { // Spriteだと回転が面倒なので Drawable をそのまま使う
19     protected double attackPower;
20     protected double radius;
21     private IAttackable attacker;
22     private GameObject enemyParentObject;
23     protected BaseEnemy nearEnemy;
24     private GameTimer timer;
25
26     private Image image;
27
28     public Laser(GameObject root, GameObject parent, double attackPower, double radius,
29                 LaserTurret laserTurret,
30                 BaseEnemy nearEnemy) {
31         super(root, parent);
32         this.attackPower = attackPower;
33         this.radius = radius;
34         this.collider = new CircleCollider(this, radius);
35         this.attacker = laserTurret;
36         this.nearEnemy = nearEnemy;
37         this.image = ImageManager.getInstance().getImage(ResourcePathDefines.
38             LASER_IMAGE_PATH);
39         this.depth = 2;
40     }
41
42     @Override
43     protected void onSummoned() {
44         super.onSummoned();
45         timer = new GameTimer(300, this);
46         timer.start();
47     }
48
49     @Override
50     public void pause() {
51         super.pause();
52         timer.pause();
53     }
54
55     @Override
56     public void unpause() {
57         super.unpause();
58         timer.unpause();
59     }
60
61     @Override
62     public void calc(float deltaTime) {
63         super.calc(deltaTime);
64         if (nearEnemy.getEnabled()) {
65             Vector2 diff = Vector2.diff(nearEnemy.position, this.position);
66             if (diff.norm() > radius * radius) {
67                 this.destroy();
68             }
69         } else {
70             this.destroy();
71         }
72     }
73
74     @Override
```

```

72 public void draw(Graphics g, Vector2 screenPosition, float ratio) {
73     Graphics2D g2 = (Graphics2D) g;
74     AffineTransform originalAffineTransform = g2.getTransform();
75     AffineTransform affineTransform = (AffineTransform) originalAffineTransform.clone
76         ();
77     double dist = Math.sqrt((nearEnemy.position.y - this.position.y) * (nearEnemy.
78         position.y - this.position.y)
79             + (nearEnemy.position.x - this.position.x) * (nearEnemy.position.x - this.
80                 position.x));
81     affineTransform
82         .rotate(Math.atan2(nearEnemy.position.y - this.position.y, nearEnemy.
83             position.x - this.position.x)
84                 - Math.toRadians(90), screenPosition.x, screenPosition.y);
85     g2.setTransform(affineTransform);
86     g2.drawImage(this.image, (int) (screenPosition.x - 200 * ratio * 0.5), (int) (
87         screenPosition.y),
88             (int) (200 * ratio), (int) (dist * ratio), null);
89     g2.setTransform(originalAffineTransform); // アフィン変換行列を元に戻す
90 }
91
92 @Override
93 public void onEnemyDead(BaseEnemy enemy) {
94     this.destroy();
95 }
96
97 @Override
98 public void actionPerformed(ActionEvent e) {
99     if (e.getSource() == timer) {
100         nearEnemy.applyDamage(new Damage(attackPower, this));
101     }
102 }
103
104 }

```

---

ソースコード 37: uectd.game.gameScene.gameMain.LaserTurret

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
4 import uectd.gameSystem.util.CircleCollider;
5 import uectd.gameSystem.util.GameTimer;
6 import uectd.gameSystem.util.ImageManager;
7 import uectd.gameSystem.util.SoundManager;
8 import uectd.game.ResourcePathDefines;
9
10 import java.util.ArrayList;
11 import java.util.HashSet;
12
13 import java.awt.event.*;
14
15 public class LaserTurret extends BaseTurret implements ActionListener {
16     private static final double TURRET_RADIUS = 50; // 見た目の大さ
17     private int attackPowerLevelupPrice;
18     private int turretCollisionRadiusLevelupPrice;
19     private int rofLevelupPrice;
20     private int totalPurchasePrice = 0;
21     ArrayList<GameObject> nearEnemies;
22     private final static double LEVELUP_ATTACK_POWER = 0.2;
23     private final static double LEVELUP_COLLISION_RADIUS = 10;
24     private final static double INTERCEPT_COLLISION_RADIUS = 150;
25     private final static int LEVELUP_LANDING_TIME = 50;
26     private final static int LEVELUP_PRICE = 10;
27     private final static int SALE_PRICE = 10;
28     private static final double INTERCEPT_ATTACK_POWER = 1;
29     private SearchEnemyField searchEnemyField;
30     private GameTimer timer;
31
32     public LaserTurret(GameObject root, GameObject parent, IMoneyTransfer iMoneyTransfer
33     ) {

```

```

33     super(root, parent, "Laser", INTERCEPT_COLLISION_RADIUS +
34         LEVELUP_COLLISION_RADIUS,
35         INTERCEPT_ATTACK_POWER + LEVELUP_ATTACK_POWER, 1000, 10, 100,
36         ImageManager.getInstance().getDivImage(ResourcePathDefines.
37             LASER_TURRET_IMAGE, 1, 1, 64, 64),
38         iMoneyTransfer);
39     depth = 1;
40 }
41
42 @Override
43 public void onDestroyed() {
44     super.onDestroyed();
45     timer.stop();
46 }
47
48 @Override
49 protected void onSummoned() {
50     super.onSummoned();
51     this.collider = new CircleCollider(this, TURRET_RADIUS);
52     searchEnemyField = new SearchEnemyField(root, this);
53     searchEnemyField.setRadius(getTurretCollisionRadius());
54     this.addChild(searchEnemyField);
55     searchEnemyField.position = this.position;
56     this.timer = new GameTimer(getRof(), this);
57     this.timer.start();
58 }
59
60 @Override
61 public void upAttackLevel() {
62     super.upAttackLevel();
63     attackPower = INTERCEPT_ATTACK_POWER + LEVELUP_ATTACK_POWER * getAttackLevel();
64     for (GameObject child : children) {
65         if (child instanceof Laser) {
66             ((Laser) child).attackPower = attackPower;
67         }
68     }
69 }
70
71 @Override
72 public void upRangeLevel() {
73     super.upRangeLevel();
74     turretCollisionRadius = INTERCEPT_COLLISION_RADIUS + LEVELUP_COLLISION_RADIUS *
75         getRangeLevel();
76     searchEnemyField.setRadius(turretCollisionRadius);
77     for (GameObject child : children) {
78         if (child instanceof Laser) {
79             ((Laser) child).radius = turretCollisionRadius;
80         }
81     }
82 }
83
84 @Override
85 public void upRofLevel() {
86     int prevRofLevel = getRofLevel();
87     super.upRofLevel();
88     if (prevRofLevel != getRofLevel()) {
89         rof = (int) (1000 - LEVELUP_LANDING_TIME * getRofLevel());
90         timer.pause();
91         int nextInitialDelay = timer.getInitialDelay();
92         timer.stop();
93         timer = new GameTimer(rof, this);
94         timer.setInitialDelay(nextInitialDelay);
95         timer.start();
96     }
97 }
98
99 @Override
100 public int getAttackPowerLevelupPrice() {
101     attackPowerLevelupPrice = 20 + LEVELUP_PRICE * getAttackLevel();
102     totalPurchasePrice += attackPowerLevelupPrice;
103     return attackPowerLevelupPrice;
104 }

```

```

105     turretCollisionRadiusLevelupPrice = 20 + LEVELUP_PRICE * getRangeLevel();
106     totalPurchasePrice += turretCollisionRadiusLevelupPrice;
107     return turretCollisionRadiusLevelupPrice;
108 }
109
110 @Override
111 public int getRofLevelupPrice() {
112     rofLevelupPrice = 20 + LEVELUP_PRICE * getRofLevel();
113     totalPurchasePrice += rofLevelupPrice;
114     return rofLevelupPrice;
115 }
116
117 @Override
118 public int getSalePrice() {
119     return (purchasePrice + totalPurchasePrice) * 8 / 10;
120 }
121
122 public void hitLaser() {
123     nearEnemies = searchEnemyField.searchNearEnemies();
124     var enemySetInAttack = new HashSet<BaseEnemy>();
125     for (var child : children) {
126         if (child instanceof Laser && child.getEnabled()) {
127             enemySetInAttack.add((Laser) child).nearEnemy);
128         }
129     }
130     for (GameObject nearEnemy : nearEnemies) {
131         if (nearEnemy != null && !enemySetInAttack.contains(nearEnemy)) {
132             Laser laser = new Laser(root, parent, attackPower, getTurretCollisionRadius
133             (), this,
134             (BaseEnemy) nearEnemy);
135             laser.position = this.position.clone();
136             SoundManager.getInstance().play(ResourcePathDefines.LASER_SOUND_PATH);
137             this.addChild(laser);
138         }
139     }
140 }
141
142 @Override
143 public void actionPerformed(ActionEvent e) {
144     if (e.getSource() == this.timer)
145         this.hitLaser();
146 }
147
148 @Override
149 public void pause() {
150     super.pause();
151     timer.pause();
152 }
153
154 @Override
155 public void unpause() {
156     super.unpause();
157     timer.unpause();
158 }
159
160 @Override
161 public void onSelected() {
162     super.onSelected();
163     searchEnemyField.setDisplayFlag(true);
164 }
165
166 @Override
167 public void onUnSelected() {
168     super.onUnSelected();
169     searchEnemyField.setDisplayFlag(false);
170 }

```

---

ソースコード 38: uectd.game.gameScene.gameMain.SearchEnemyField

---

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.util.ArrayList;
4 import java.awt.*;
5

```

```

6 import uectd.game.ResourcePathDefines;
7 import uectd.gameSystem.GameObject;
8 import uectd.gameSystem.util.CircleCollider;
9 import uectd.gameSystem.util.Sprite;
10 import uectd.gameSystem.util.Vector2;
11
12 public class SearchEnemyField extends Sprite {
13     private boolean isDisplaying;
14     private EnemyParent enemyParent;
15
16     public SearchEnemyField(GameObject root, GameObject parent) {
17         super(root, parent);
18         collider = new CircleCollider(this, 0);
19         depth = 4;
20     }
21
22     @Override
23     public void calc(float deltaTime) {
24         super.calc(deltaTime);
25     }
26
27     @Override
28     protected void onSummoned() {
29         super.onSummoned();
30         this.enemyParent = (EnemyParent) root.findChild(EnemyParent.class);
31     }
32
33     public void setRadius(double radius) {
34         ((CircleCollider) collider).radius = radius;
35         this.setDrawSize(new Vector2(radius * 2, radius * 2));
36     }
37
38     public void setDisplayFlag(boolean isDisplaying) {
39         this.isDisplaying = isDisplaying;
40         if (isDisplaying) {
41             setImage(ResourcePathDefines.ENEMY_SEARCH_FIELD_IMAGE, false);
42         } else {
43             setImage((Image) null, false);
44         }
45     }
46
47     public ArrayList<GameObject> searchNearEnemies() {
48         return this.collider.findCollidedChildren(this.enemyParent);
49     }
50
51     public BaseEnemy searchNearestEnemy() {
52         BaseEnemy nearestEnemy = null;
53         double nearestDistance = Double.POSITIVE_INFINITY;
54         ArrayList<GameObject> collisionEnemies = searchNearEnemies();
55         for (GameObject gameObject : collisionEnemies) {
56             if (gameObject instanceof BaseEnemy) {
57                 // range 内で Turret から最も近い enemy を選択
58                 double distance = Vector2.diff(this.position, gameObject.position).norm();
59                 if (distance < nearestDistance) {
60                     nearestDistance = distance;
61                     nearestEnemy = (BaseEnemy) gameObject;
62                 }
63             }
64         }
65         return nearestEnemy;
66     }
67 }

```

---

ソースコード 39: uectd.game.gameScene.gameMain.Tower

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import java.util.ArrayList;
6
7 import uectd.gameSystem.GameObject;
8 import uectd.gameSystem.util.Sprite;
9 import uectd.gameSystem.util.Vector2;

```

```

10 import java.awt.*;
11
12 public class Tower extends Sprite implements IDamageApplicable {
13
14     private static final double DEFAULT_HP = 1000;
15     protected double hp;
16     protected String name;
17     protected ArrayList<TowerFallListener> towerFallListeners, addList, removeList;
18     protected Graph.Vertex vertex;
19
20     public Tower(GameObject root, GameObject parent, Graph.Vertex vertex) {
21         super(root, parent);
22         this.vertex = vertex;
23         this.towerFallListeners = new ArrayList<>();
24         this.addList = new ArrayList<>();
25         this.removeList = new ArrayList<>();
26         this.position = vertex.position;
27         this.hp = DEFAULT_HP;
28     }
29
30     public String getName() {
31         return name;
32     }
33
34     public void setName(String name) {
35         this.name = name;
36     }
37
38     public double getHp() {
39         return this.hp;
40     }
41
42     public double getMaxHp() {
43         return DEFAULT_HP;
44     }
45
46     public Graph.Vertex getVertex() {
47         return vertex;
48     }
49
50
51     @Override
52     public void calc(float deltaTime) {
53         super.calc(deltaTime);
54         if (!addList.isEmpty()) {
55             for (var towerFallListener : addList) {
56                 towerFallListeners.add(towerFallListener);
57             }
58             addList.clear();
59         }
60         if (!removeList.isEmpty()) {
61             for (var towerFallListener : removeList) {
62                 towerFallListeners.remove(towerFallListener);
63             }
64             removeList.clear();
65         }
66     }
67
68     @Override
69     public void applyDamage(Damage d) {
70         if (getEnabled()) {
71             hp -= d.attack;
72             if (hp <= 0) {
73                 hp = 0;
74                 for (TowerFallListener towerFallListener : towerFallListeners) {
75                     towerFallListener.onTowerFall(this);
76                 }
77                 addList.clear();
78                 removeList.clear();
79                 towerFallListeners.clear();
80                 this.destroy();
81
82                 var effect = new TowerFallEffect(root, root);
83                 effect.position = this.position.clone();
84                 root.addChild(effect);

```

```

85         }
86     }
87 }
88 public void addTowerFallListener(TowerFallListener towerFallListener) {
89     addList.add(towerFallListener);
90 }
91 public void removeTowerFallListener(TowerFallListener towerFallListener) {
92     removeList.add(towerFallListener);
93 }
94 }
95 }
96 @Override
97 public void draw(Graphics g, Vector2 screenPosition, float ratio) {
98     super.draw(g, screenPosition, ratio);
99     g.setColor(new Color(50, 50, 240, 120));
100    Graphics2D g2d = (Graphics2D) g;
101    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
102        VALUE_ANTIALIAS_ON);
103    g2d.setRenderingHint(RenderingHints.KEY_RENDERING, RenderingHints.
104        VALUE_RENDER_QUALITY);
105    g2d.setStroke(new BasicStroke((int) (20 * ratio), BasicStroke.CAP_BUTT,
106        BasicStroke.JOIN_MITER));
107    g2d.setColor(Color.GRAY);
108    g2d.drawOval((int) screenPosition.x, (int) screenPosition.y - (int) (80 * ratio), (
109        int) (90 * ratio),
110        (int) (90 * ratio));
111    g2d.setStroke(new BasicStroke((int) (13 * ratio), BasicStroke.CAP_BUTT,
112        BasicStroke.JOIN_MITER));
113    if (hp > DEFAULT_HP * 0.5) {
114        g.setColor(Color.GREEN);
115    } else if (hp > DEFAULT_HP * 0.3) {
116        g.setColor(Color.YELLOW);
117    } else {
118        g.setColor(Color.RED);
119    }
}

```

ソースコード 40: uectd.game.gameScene.gameMain.TowerArranger

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.io.*;
4 import java.util.*;
5
6 import uectd.gameSystem.FatalError;
7 import uectd.gameSystem.GameObject;
8
9 public class TowerArranger {
10     private static final int TOWER_NUM = 2;
11
12     public static ArrayList<Tower> arrange(String towerCandidateFilePath, GameObject
13         rootGameObject, TowerParent parent,
14         Graph graph) {
15         try {
16             File candidateVertexFile = new File(towerCandidateFilePath);
17             if (candidateVertexFile.exists() && candidateVertexFile.isFile() &&
18                 candidateVertexFile.canRead()) {
19                 Scanner sc = new Scanner(candidateVertexFile, "UTF-8");
20                 int n = sc.nextInt(); // 入力行数。
21                 var candidateIndexes = new ArrayList<Integer>();
22                 for (int i = 0; i < n; i++) {
23                     candidateIndexes.add(i);
24                 }
25                 Collections.shuffle(candidateIndexes);
26                 var arrangeIndexes = candidateIndexes.subList(0, Math.min(TOWER_NUM, n));
27                 var res = new ArrayList<Tower>();
28                 for (int i = 0; i < n; i++) {
29                     int idx = sc.nextInt();
30                     String name = sc.next();
31                     Tower tower = new Tower(name, idx);
32                     res.add(tower);
33                 }
34             }
35         }
36     }
37 }

```

```

29         if (arrangeIndexes.contains(i)) {
30             Tower tower = new Tower(rootGameObject, (GameObject) parent, graph.
31                 vertexes.get(idx));
32             tower.setName(name);
33             parent.addChild(tower);
34             res.add(tower);
35         }
36     sc.close();
37     return res;
38 } else {
39     FatalError.quit("タワー位置情報ファイルが見つからないか開けません");
40 }
41 } catch (FileNotFoundException e) {
42     e.printStackTrace();
43     FatalError.quit("タワー位置情報ファイルが存在しません");
44 }
45 return null;
46 }
47 }

```

---

ソースコード 41: uectd.game.gameScene.gameMain.TowerFallEffect

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.awt.event.*;
4
5 import uectd.gameSystem.GameObject;
6 import uectd.gameSystem.util.Effect;
7 import uectd.gameSystem.util.GameTimer;
8 import uectd.gameSystem.util.Vector2;
9
10 public class TowerFallEffect extends Effect implements ActionListener {
11
12     private final static int DELAY = 250;
13
14     private int bombCount;
15     private GameTimer timer;
16
17     public TowerFallEffect(GameObject root, GameObject parent) {
18         super(root, parent, null, true, 0.0f, 0);
19         timer = new GameTimer(DELAY, this);
20         timer.start();
21     }
22
23     @Override
24     protected void onSummoned() {
25         super.onSummoned();
26         bombCount = 0;
27     }
28
29     @Override
30     public void actionPerformed(ActionEvent e) {
31         bombCount++;
32         if (bombCount == 10) {
33             timer.stop();
34             destroy();
35         } else {
36             var sub = new TowerFallEffectSub(root, parent);
37             sub.position = Vector2.sum(position, new Vector2(Math.random() * 200 - 100, Math
38             .random() * 200 - 100));
39             this.addChild(sub);
40         }
41     }
42 }

```

---

ソースコード 42: uectd.game.gameScene.gameMain.TowerFallEffectSub

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.gameSystem.GameObject;

```

```
5 import uectd.gameSystem.util.Effect;
6 import uectd.gameSystem.util.ImageManager;
7 import uectd.gameSystem.util.SoundManager;
8
9 public class TowerFallEffectSub extends Effect {
10     private final static int X_NUM = 7;
11     private final static int Y_NUM = 1;
12     private final static int X_SIZE = 120;
13     private final static int Y_SIZE = 120;
14
15     public TowerFallEffectSub(GameObject root, GameObject parent) {
16         super(root, parent, ImageManager.getInstance().getDivImage(ResourcePathDefines.
17             EFFECT_BOMB_IMAGES, X_NUM, Y_NUM,
18             X_SIZE, Y_SIZE), false, 0.07f, 0);
19     }
20 }
```

---

ソースコード 43: uectd.game.gameScene.gameMain.TowerFallListener

```
1 package uectd.game.gameScene.gameMain;
2
3 public interface TowerFallListener {
4     void onTowerFall(Tower tower);
5 }
```

---

ソースコード 44: uectd.game.gameScene.gameMain.TowerParent

```
1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
4
5 public class TowerParent extends GameObject {
6
7     public TowerParent(GameObject root, GameObject parent) {
8         super(root, parent);
9     }
10
11 }
```

---

ソースコード 45: uectd.game.gameScene.gameMain.TurretAttackToEnemyEffect

```
1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.gameSystem.GameObject;
5 import uectd.gameSystem.util.Effect;
6 import uectd.gameSystem.util.ImageManager;
7 import uectd.gameSystem.util.SoundManager;
8 import uectd.gameSystem.util.Vector2;
9
10 public class TurretAttackToEnemyEffect extends Effect {
11     private final static int X_NUM = 5;
12     private final static int Y_NUM = 1;
13     private final static int X_SIZE = 240;
14     private final static int Y_SIZE = 240;
15
16     public TurretAttackToEnemyEffect(GameObject root, GameObject parent) {
17         super(root, parent, ImageManager.getInstance().getDivImage(ResourcePathDefines.
18             TURRET_ATTACK_TO_ENEMY_IMAGES,
19             X_NUM, Y_NUM, X_SIZE, Y_SIZE), false, 0.03f, 0);
20         setDrawSize(new Vector2(80, 80));
21     }
22 }
```

---

ソースコード 46: uectd.game.gameScene.gameMain.TurretParent

```
1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
```

```

4
5 public class TurretParent extends GameObject {
6
7     public TurretParent(GameObject root, GameObject parent) {
8         super(root, parent);
9     }
10}
11}

```

---

ソースコード 47: uectd.game.gameScene.gameMain.TurretSocket

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.game.ResourcePathDefines;
4 import uectd.gameSystem.GameObject;
5 import uectd.gameSystem.util.CircleCollider;
6 import uectd.gameSystem.util.IClickable;
7 import uectd.gameSystem.util.ImageManager;
8 import uectd.gameSystem.util.Sprite;
9
10 public class TurretSocket extends Sprite implements IClickable {
11
12     private TurretParent turretParent; // 配置されたタレットが属するゲーム内オブジェクトのヒ
13     // エラルキーの親
14     private BaseTurret turret; // タレットを持つ。配置されていない間はnullとする
15     // GameSceneModel.
16     // selectingTurret を private で持っておく。ソケットクリック時に selectingTurret
17     // GO のコンストラクタ GO(root, parent)なので
18     // root から findchild してなんとか TP を発見してこのコンストラクタに投げてあげればおk
19     public TurretSocket(GameObject root, GameObject parent, TurretParent turretParent) {
20         super(root, parent);
21         this.turret = null;
22         this.setImage(ImageManager.getInstance().getImage(ResourcePathDefines.
23             TURRET_SOCKET_IMAGE));
24         this.turretParent = turretParent;
25         this.depth = 5;
26         this.collider = new CircleCollider(this, this.image.getHeight(null) / 2);
27     }
28
29     // 建築。もしソケットにタレットがあれば建築はできない。
30     public boolean tryBuildTurret(BaseTurret turret) {
31         if (this.turret == null || !(this.turret.getEnabled())) { // タレットがない or 売却
32             // 済みにより enabled==false の場合、タレットは建設可能
33             this.turret = turret;
34             turret.position = this.position.clone();
35             turret.parent = turretParent;
36             turretParent.addChild(turret);
37             return true;
38         } else { // この条件に当てはまらないと建設はできない
39             return false;
40         }
41     }
42 }

```

---

ソースコード 48: uectd.game.gameScene.gameMain.TurretSocketArranger

```

1 package uectd.game.gameScene.gameMain;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.util.Scanner;
6
7 import uectd.game.ResourcePathDefines;
8 import uectd.gameSystem.FatalError;
9 import uectd.gameSystem.GameObject;
10 import uectd.gameSystem.util.ImageManager;
11 import uectd.gameSystem.util.Vector2;
12
13 public class TurretSocketArranger {

```

```

14     public static void arrangeTurrets(String turretSocketsPath, GameObject root,
15             TurretSocketParent turretSocketParent,
16             TurretParent turretParent) {
17         try {
18             File turretsFile = new File(turretSocketsPath);
19             if (turretsFile.exists() && turretsFile.isFile() && turretsFile.canRead()) {
20                 Scanner sc = new Scanner(turretsFile);
21                 int n = sc.nextInt(); // 入力行数。
22                 int halfWidth = ImageManager.getInstance().getImage(ResourcePathDefines.
23                                     BACKGROUND_IMAGE).getWidth(null)
24                                     / 2;
25                 int halfHeight = ImageManager.getInstance().getImage(ResourcePathDefines.
26                                     BACKGROUND_IMAGE)
27                                     .getHeight(null) / 2;
28                 for (int i = 0; i < n; i++) {
29                     int x = sc.nextInt() - halfWidth, y = sc.nextInt() - halfHeight;
30                     var turretSocket = new TurretSocket(root, turretSocketParent,
31                                         turretParent);
32                     turretSocket.position = new Vector2(x, y);
33                     turretSocketParent.addChild(turretSocket);
34                 }
35                 sc.close();
36             } else {
37                 FatalError.quit("タレット配置情報ファイルが見つからないか開けません");
38             }
39         } catch (FileNotFoundException e) {
40             e.printStackTrace();
41             FatalError.quit("タレット配置情報ファイルが存在しません");
42         }
43     }

```

---

ソースコード 49: uectd.game.gameScene.gameMain.TurretSocketParent

```

1 package uectd.game.gameScene.gameMain;
2
3 import uectd.gameSystem.GameObject;
4
5 public class TurretSocketParent extends GameObject {
6
7     public TurretSocketParent(GameObject root, GameObject parent) {
8         super(root, parent);
9     }
10 }
11

```

---

ソースコード 50: uectd.game.gameScene.GameScene

```

1 package uectd.game.gameScene;
2
3 import uectd.gameSystem.*;
4
5 public class GameScene extends Scene {
6
7     public GameScene(SceneChangeListener sceneChangeListener) {
8         super(sceneChangeListener);
9         this.model = new GameSceneModel(sceneChangeListener);
10        this.view = new GameSceneView(model);
11        this.controller = new GameSceneController(model, view);
12    }
13 }
14

```

---

ソースコード 51: uectd.game.gameScene.GameSceneController

```

1 package uectd.game.gameScene;
2
3 import java.awt.Point;
4 import java.awt.event.*;
5 import java.util.ArrayList;
6 import java.util.Observable;

```

```

7 import java.util.Observer;
8
9 import javax.swing.JButton;
10
11 import uectd.game.gameScene.GameSceneView.AddTurretPanel.TurretDisplayButton;
12 import uectd.gameSystem.SceneController;
13 import uectd.gameSystem.SceneModel;
14 import uectd.gameSystem.SceneView;
15 import uectd.gameSystem.util.IClickable;
16 import uectd.gameSystem.util.ObservableComponent;
17 import uectd.gameSystem.util.Vector2;
18
19 public class GameSceneController extends SceneController implements Observer {
20     private boolean isWKeyPressed;
21     private boolean isAKeyPressed;
22     private boolean isSKeyPressed;
23     private boolean isDKeyPressed;
24     private boolean isSHIFTKeyPressed;
25     private boolean isSPACEKeyPressed;
26     private GameSceneModel gameModel;
27     private GameSceneView gameView;
28     private GameSceneView.AddTurretPanel addTurretPanel;
29     private GameSceneView.UpgradeTurretPanel upgradeTurretPanel;
30     private JButton rangeButton, powerButton, rofButton, sellButton;
31     private ObservableComponent<ArrayList<TurretDisplayButton>> turretButtons;
32
33     public GameSceneController(SceneModel model, SceneView view) {
34         super(model, view);
35         gameModel = (GameSceneModel) model;
36         gameView = (GameSceneView) view;
37         addTurretPanel = gameView.addTurretPanel;
38         turretButtons = gameView.addTurretPanel.turretButtons;
39         turretButtons.addObserver(this);
40         addActionListenerToTurretButtons();
41         upgradeTurretPanel = gameView.upgradeTurretPanel;
42         rangeButton = upgradeTurretPanel.rangeButton;
43         powerButton = upgradeTurretPanel.powerButton;
44         rofButton = upgradeTurretPanel.rofButton;
45         sellButton = upgradeTurretPanel.sellButton;
46         rangeButton.addActionListener(this);
47         powerButton.addActionListener(this);
48         rofButton.addActionListener(this);
49         sellButton.addActionListener(this);
50     }
51
52     public void addActionListenerToTurretButtons() {
53         for (TurretDisplayButton turretDisplayButton : turretButtons.getValue()) {
54             turretDisplayButton.addActionListener(this);
55         }
56     }
57
58     @Override
59     public void actionPerformed(ActionEvent e) {
60         super.actionPerformed(e);
61         var src = e.getSource();
62         if (src == rangeButton) {
63             gameModel.upTurretRangeLevel();
64         } else if (src == powerButton) {
65             gameModel.upTurretAttackLevel();
66         } else if (src == rofButton) {
67             gameModel.upTurretRofLevel();
68         } else if (src == sellButton) {
69             gameModel.sellTurret();
70         }
71         for (TurretDisplayButton button : turretButtons.getValue()) {
72             if (button == src) {
73                 gameModel.setPurchasingTurret(button.turret);
74             }
75         }
76     }
77
78     public void moveCamera() {
79         Vector2 cameraVector = new Vector2();
80         cameraVector.x = (isAKeyPressed ? -1 : 0) + (isDKeyPressed ? 1 : 0);
81

```

```

82         cameraVector.y = (isSKeyPressed ? 1 : 0) + (isWKeyPressed ? -1 : 0);
83         gameModel.setCameraMoveDirection(cameraVector.normalized());
84     }
85
86     public void updateScaleOperation() {
87         int scale = 0;
88         scale += (isSHIFTKeyPressed ? 1 : 0);
89         scale += (isSPACEKeyPressed ? -1 : 0);
90         // scale = 1 --> 拡大; scale = 0 -->何もしない; scale = -1 --> 縮小;
91         gameModel.setCameraZoomOperation(scale); // model に scale を渡す。
92     }
93
94     @Override
95     public void mouseClicked(MouseEvent e) {
96         // クリックした座標を受け取ったら、その座標をワールド座標に変換してから、クリック可能なゲー
97         // ムオブジェクトをクリックしているかどうかを確かめて、もしクリックしていたら
98         // model にそれを通知。クリックしていないなら NULL を返す。
99         // クリックしているかどうかの処理はfindClickedObject
100        // 例はClickSample の GameSceneController の mousePressed
101        Point point = e.getPoint();
102        Vector2 worldPos = view.convertScreenPosToWorldPos(new Vector2(point.x, point.y));
103        IClickable clicked = findClickedObject(worldPos);
104        gameModel.receiveIClickable(clicked);
105    }
106
107    @Override
108    public void mousePressed(MouseEvent e) {
109    }
110
111    @Override
112    public void mouseReleased(MouseEvent e) {
113    }
114
115    @Override
116    public void mouseEntered(MouseEvent e) {
117    }
118
119    @Override
120    public void mouseExited(MouseEvent e) {
121    }
122
123    @Override
124    public void mouseDragged(MouseEvent e) {
125    }
126
127    @Override
128    public void mouseMoved(MouseEvent e) {
129    }
130
131    @Override
132    public void keyTyped(KeyEvent e) {
133    }
134
135    @Override
136    public void keyPressed(KeyEvent e) {
137        int keyCode = e.getKeyCode();
138        if (keyCode == KeyEvent.VK_ESCAPE) {
139            ((GameSceneModel) model).pauseKeyPressed();
140        }
141        if (keyCode == KeyEvent.VK_W) {
142            isWKeyPressed = true;
143        }
144        if (keyCode == KeyEvent.VK_A) {
145            isAKKeyPressed = true;
146        }
147        if (keyCode == KeyEvent.VK_S) {
148            isSKKeyPressed = true;
149        }
150        if (keyCode == KeyEvent.VK_D) {
151            isDKKeyPressed = true;
152        }
153        if (keyCode == KeyEvent.VK_SHIFT) {

```

```

154         isSHIFTKeyPressed = true;
155     }
156     if (keyCode == KeyEvent.VK_SPACE) {
157         isSPACEKeyPressed = true;
158     }
159
160     if (isWKeyPressed || isAKeyPressed || isSKeyPressed || isDKeyPressed) {
161         moveCamera();
162     }
163
164     if (isSHIFTKeyPressed || isSPACEKeyPressed) {
165         updateScaleOperation();
166     }
167 }
168
169 @Override
170 public void keyReleased(KeyEvent e) {
171     int keyCode = e.getKeyCode();
172     if (keyCode == KeyEvent.VK_W) {
173         isWKeyPressed = false;
174     }
175     if (keyCode == KeyEvent.VK_A) {
176         isAKeyPressed = false;
177     }
178     if (keyCode == KeyEvent.VK_S) {
179         isSKeyPressed = false;
180     }
181     if (keyCode == KeyEvent.VK_D) {
182         isDKeyPressed = false;
183     }
184
185     if (keyCode == KeyEvent.VK_SHIFT) {
186         isSHIFTKeyPressed = false;
187     }
188     if (keyCode == KeyEvent.VK_SPACE) {
189         isSPACEKeyPressed = false;
190     }
191
192     if (!(isWKeyPressed && isAKeyPressed && isSKeyPressed && isDKeyPressed)) {
193         moveCamera();
194     }
195
196     if (!(isSHIFTKeyPressed && isSPACEKeyPressed)) {
197         updateScaleOperation();
198     }
199 }
200
201 @Override
202 public void update(Observable o, Object arg) {
203     if (o == turretButtons) {
204         addActionListenerToTurretButtons();
205     }
206 }
207 }
```

ソースコード 52: uectd.game.gameScene.GameSceneModel

---

```

1 package uectd.game.gameScene;
2
3 import uectd.gameSystem.util.*;
4
5 import java.awt.event.*;
6 import java.util.ArrayList;
7
8 import uectd.game.ResourcePathDefines;
9 import uectd.game.gameScene.gameMain.BackGround;
10 import uectd.game.gameScene.gameMain.BallistaTurret;
11 import uectd.game.gameScene.gameMain.BaseEnemy;
12 import uectd.game.gameScene.gameMain.Tower;
13 import uectd.game.gameScene.gameMain.BaseTurret;
14 import uectd.game.gameScene.gameMain.CannonTurret;
15 import uectd.game.gameScene.gameMain.EnemyDieListener;
16 import uectd.game.gameScene.gameMain.EnemyParent;
17 import uectd.game.gameScene.gameMain.EnemySpawnerManager;
18 import uectd.game.gameScene.gameMain.EnemySpawnerParent;
```

```

19 import uectd.game.gameScene.gameMain.GameLevel;
20 import uectd.game.gameScene.gameMain.GraphBuilder;
21 import uectd.game.gameScene.gameMain.IMoneyTransfer;
22 import uectd.game.gameScene.gameMain.LaserTurret;
23 import uectd.game.gameScene.gameMain.FileReadSpawnStrategy;
24 import uectd.game.gameScene.gameMain.TowerArranger;
25 import uectd.game.gameScene.gameMain.TowerFallListener;
26 import uectd.game.gameScene.gameMain.TowerParent;
27 import uectd.game.gameScene.gameMain.TurretParent;
28 import uectd.game.gameScene.gameMain.TurretSocket;
29 import uectd.game.gameScene.gameMain.TurretSocketArranger;
30 import uectd.game.gameScene.gameMain.TurretSocketParent;
31 import uectd.game.gameScene.gameMain.enemy.Zombie;
32 import uectd.game.gameScene.gameMain.enemy.Goblin;
33 import uectd.game.gameScene.gameMain.enemy.Hobgoblin;
34 import uectd.game.gameScene.gameMain.enemy.Cyclopes;
35 import uectd.game.pauseScene.PauseScene;
36 import uectd.game.resultScene.ResultScene;
37 import uectd.gameSystem.Define;
38 import uectd.gameSystem.SceneChangeListener;
39 import uectd.gameSystem.SceneModel;
40
41 public class GameSceneModel extends SceneModel
42     implements ActionListener, IMoneyTransfer, EnemyDieListener, TowerFallListener {
43
44     public enum GameState {
45         WaitingForNextWave, OnWave, GameOver, GameClear
46     } // 内部クラスで状態を表すクラスを定義。状態遷移に用いる。
47
48     private final float cameraZoomSpeed = 0.3f; // カメラの拡縮スピード
49     private final float cameraZoomMax = 2.0f; // カメラ拡大の最大値
50     private final float cameraZoomMin = 0.2f; // カメラ縮小の最小値
51
52     private BackGround backGround; // 背景
53
54     private Camera camera;
55     private Vector2 cameraMoveDirection; // 正規化された、カメラが動くベクトルが入る変数
56     private double cameraSpeed; // カメラの移動速度に緩急をつけるための変数
57     private int cameraZoomOperation; // カメラの拡縮操作を
58         Controllerから受け取るための変数。範囲 [-1, 1] の整数が格納される。
59
60     private GameTimer nextWaveTimer, waveEndTimer, gameOverTimer, gameClearTimer;
61     // 次
62         wave を待機するためのタイマー、wave 開始から終了までのタイマー、ゲームオーバーとゲームクリア遷移
63
64     public ObservableComponent<GameState> currentState; // 現在のGameState
65
66     public ObservableComponent<Integer> countdownNum, waveNum, numOfEnemies, balance,
67         maxWaveNum, score;
68     // wave 開始前のカウントダウン値、現在ウェーブ数、マップ上の敵数、所持金、最大ウェーブ数(このウェーブを凌げばクリア)、スコア
69
70     public ObservableComponent<ArrayList<BaseTurret>> availableTurretList; // 利用可能なタレット群
71     public ObservableComponent<BaseTurret> selectingTurret; // 画面右にタレット情報を表示させるための変数
72     public ObservableComponent<BaseTurret> purchasingTurret; // 購入対象のタレット
73
74     private EnemySpawnerManager enemySpawnerManager; //
75         wave ごとに EnemySpawner を配置するためのクラス
76
77     public ObservableComponent<ArrayList<Tower>> targets; // 襲撃目標群
78
79     // 新しく作成されるゲーム内オブジェクトが属する、ゲーム内オブジェクトのヒエラルキーの親
80     private TurretParent turretParent; // タレットの親クラス
81     private TowerParent towerParent; // タワーの親クラス
82     private EnemySpawnerParent enemySpawnerParent; // スポナーの親クラス
83
84     public GameSceneModel(SceneChangeListener sceneChangeListener) {
85         super(sceneChangeListener);

```

```

82     currentState = new ObservableComponent<GameState>(GameState.WaitingForNextWave);
83     camera = new Camera(rootGameObject, rootGameObject, 1); // 初期のズーム比率は 1
84     addGameObject(camera);
85
86     countdownNum = new ObservableComponent<>(10);
87     waveNum = new ObservableComponent<>(1);
88     maxWaveNum = new ObservableComponent<>(10);
89     numOfEnemies = new ObservableComponent<>(0);
90     balance = new ObservableComponent<>(2000); // 初期の所持金
91     nextWaveTimer = new GameTimer(1000, this);
92     cameraMoveDirection = new Vector2(0.0, 0.0);
93     cameraSpeed = 160d;
94     cameraZoomOperation = 0;
95     camera.position = new Vector2(730, 100);
96     camera.ratio = cameraZoomMin;
97
98     backGround = new BackGround(rootGameObject, rootGameObject);
99     addGameObject(backGround);
100    EnemyParent enemyParent = new EnemyParent(rootGameObject, rootGameObject);
101    addGameObject(enemyParent);
102    TurretSocketParent turretSocketParent = new TurretSocketParent(rootGameObject,
103                      rootGameObject);
104    addGameObject(turretSocketParent);
105    turretParent = new TurretParent(rootGameObject, rootGameObject);
106    addGameObject(turretParent);
107    enemySpawnerParent = new EnemySpawnerParent(rootGameObject, rootGameObject);
108    addGameObject(enemySpawnerParent);
109    towerParent = new TowerParent(rootGameObject, rootGameObject);
110    addGameObject(towerParent);
111    GameLevel gameLevel = new GameLevel(rootGameObject, rootGameObject,
112                                         GraphBuilder.build(ResourcePathDefines.UEC_MAP_GRAPH));
113    addGameObject(gameLevel);
114
115    selectingTurret = new ObservableComponent<>(null);
116    purchasingTurret = new ObservableComponent<>(null);
117    availableTurretList = new ObservableComponent<ArrayList<BaseTurret>>(new
118                           ArrayList<>());
119
120    TurretSocketArranger.arrangeTurrets(ResourcePathDefines.
121                                         TURRET_SOCKET_POSITION_PATH, rootGameObject,
122                                         turretSocketParent, turretParent);
123
124    targets = new ObservableComponent<>();
125    targets.setValue(TowerArranger.arrange(ResourcePathDefines.TOWER_CANDIDATE_PATH,
126                                         rootGameObject, towerParent,
127                                         gameLevel.graph));
128    for (Tower tower : targets.getValue()) {
129      tower.addTowerFallListener(this);
130    }
131
132    enemySpawnerManager = new EnemySpawnerManager(gameLevel.graph, new
133                                                 FileReadSpawnStrategy(this),
134                                                 ResourcePathDefines.CANDIDATE_VERTEX_LIST_PATH, rootGameObject,
135                                                 rootGameObject, enemySpawnerParent,
136                                                 enemyParent, gameLevel, targets.getValue());
137
138    score = new ObservableComponent<Integer>(0);
139
140    nextWaveTimer.start();
141
142  public void pauseKeyPressed() {
143    sceneChangeListener.scenePushed(new PauseScene(sceneChangeListener), null);
144  }
145
146  public void receiveIClickable(IClickable iClickable) {
147    // 受け取ったものによって処理を変える。
148    // nullなら selectingTurret は null, タレットならタレットそのもの、ソケットなら
149    // selectingTurret を建設することを試みる)
150    // System.out.println(iClickable);
151    if (iClickable == null) {
152      var prevTurret = selectingTurret.getValue();
153      if (prevTurret != null) {
154        prevTurret.onUnSelected();

```

```

149         }
150         selectingTurret.setValue(null);
151     } else if (iClickable instanceof BaseTurret) {
152         var prevTurret = selectingTurret.getValue();
153         if (prevTurret != null) {
154             prevTurret.onUnSelected();
155         }
156         selectingTurret.setValue((BaseTurret) iClickable);
157         ((BaseTurret) iClickable).onSelected();
158     } else if (iClickable instanceof TurretSocket) {
159         BaseTurret turret = purchasingTurret.getValue();
160         if (turret != null && tryMoneyPay(turret.getPurchasePrice())) {
161             ((TurretSocket) iClickable).tryBuildTurret(purchasingTurret.getValue().clone());
162         }
163     }
164 }
165
166 public void setPurchasingTurret(BaseTurret turret) {
167     purchasingTurret.setValue(turret);
168 }
169
170 // カメラの移動方向ベクトルを設定する
171 // setter。Controllerから移動方向ベクトルを受け取ることを想定している。
172 public void setCameraMoveDirection(Vector2 cameraMoveDirection) {
173     this.cameraMoveDirection = cameraMoveDirection;
174 }
175
176 // カメラのズーム比率を変更するため、キー入力から生成した-1,0,1の値を受け取るメソッド
177 public void setCameraZoomOperation(int cameraZoomOperation) { // 引数のintの値は
178     -1,0,1のどれか。
179     this.cameraZoomOperation = cameraZoomOperation; // Modelで持っておき Model.
180     calc メソッドで使う。
181 }
182
183 @Override
184 public void start(Intent intent) {
185     super.start(intent);
186     var list = new ArrayList<BaseTurret>();
187     if (intent.getBooleanValue("CannonTurret")) {
188         list.add(new CannonTurret(rootGameObject, turretParent, this));
189     } // 利用可能なタレットを登録していく
190     if (intent.getBooleanValue("LaserTurret")) {
191         list.add(new LaserTurret(rootGameObject, turretParent, this));
192     }
193     availableTurretList.setValue(list);
194 }
195
196 @Override
197 public void actionPerformed(ActionEvent e) {
198     var src = e.getSource();
199     if (src == nextWaveTimer) { // Waiting -> OnWave
200         int prevNum = countdownNum.getValue(); // ここでカウントダウン
201         countdownNum.setValue(prevNum - 1);
202         System.out.println("gsm.ap_" + prevNum);
203         if (prevNum <= 1) {
204             nextWaveTimer.stop(); // 一旦nextWaveTimer止める
205             currentState.setValue(GameState.OnWave); // State変更
206             waveEndTimer = new GameTimer(30000, this); // 次タイマー用意
207             enemySpawnerManager.arrange(waveNum.getValue()); //
208                 enemySpawnerManagerに対して wave 数を渡し、スポナーを設置してもらう
209             waveEndTimer.start(); // 今wave の時間切れタイマー開始
210         } else if (prevNum == 4) {
211             SoundManager.getInstance().play(ResourcePathDefines.COUNTDOWN_SOUND_PATH);
212         }
213     } else if (src == waveEndTimer) { // OnWave-> Waiting or OnWave-> GameClear
214         // spawnerParent の child をまとめて消せばok
215         if (waveNum.getValue() != maxWaveNum.getValue()) { // 次ウェーブが存在する場合
216             waveNum.setValue(waveNum.getValue() + 1);

```

```

216         waveEndTimer.stop();
217         currentState.setValue(GameState.WaitingForNextWave);
218         nextWaveTimer.stop();
219         nextWaveTimer = new GameTimer(1000, this);
220         countdownNum.setValue(10);
221         nextWaveTimer.start();
222     } else { // 次ウェーブが存在しない場合 (ゲームクリア)
223         waveEndTimer.stop();
224         if (currentState.getValue() != GameState.GameOver) {
225             currentState.setValue(GameState.GameClear);
226             gameClearTimer = new GameTimer(5000, this);
227             gameClearTimer.start();
228             System.out.println("gameclear");
229             gameClearTimer.setRepeats(false);
230         }
231     }
232 } else if (src == gameOverTimer) {
// タワー全陥落の瞬間にGame Over!! 表示-> 5sec くらい後にシーン遷移 みたいな感じ
// 遷移先はリザルト画面とか?
// タイトルに強制送還もアリかも
233     Intent intent = new Intent();
234     intent.setBooleanValue("Cleared", false);
235     intent.setIntegerValue("Score", calcScore());
236     SoundManager.getInstance().allStop();
237     sceneChangeListener.sceneChanged(new ResultScene(sceneChangeListener), intent
238 );
239 } else if (src == gameClearTimer) {
// ゲームクリア時の挙動。
240     Intent intent = new Intent();
241     intent.setBooleanValue("Cleared", true);
242     intent.setIntegerValue("Score", calcScore());
243     SoundManager.getInstance().allStop();
244     sceneChangeListener.sceneChanged(new ResultScene(sceneChangeListener), intent
245 );
246     SoundManager.getInstance().allStop();
247 }
248
249
250
251
252 @Override
253 public void calc(float deltaTime) {
// カメラの4方向移動
254     camera.position.add(Vector2.scale(cameraMoveDirection, deltaTime * cameraSpeed /
255         camera.ratio));
256     camera.position.x = Math.max(Math.min(camera.position.x, 1.3 * Define.WINDOW_WIDTH
257         ),
258         1.3 * -Define.WINDOW_WIDTH);
259     camera.position.y = Math.max(Math.min(camera.position.y, 1.3 * Define.
260         WINDOW_HEIGHT),
261         1.3 * -Define.WINDOW_HEIGHT);
// カメラの拡縮(昇降)
262     var delta = deltaTime * cameraZoomSpeed * cameraZoomOperation; // dt * (カメラの昇
263         降速度) *
// (正または負に移動、もしくは動
// かないことを示す係数)
264     camera.ratio = Math.max(Math.min(cameraZoomMax, camera.ratio + delta),
265         cameraZoomMin); // clamp処理。超過時に上限下限に合わせる。
266 }
267
268
269
270
271
272
273
274
275
276
277
278
279
280

```

```

281     public int getBalance() {
282         return balance.getValue();
283     }
284
285     @Override
286     public void onEnemyDead(BaseEnemy enemy) {
287         moneyAdd(enemy.getDropValue());
288         addScore(enemy.getDropScore());
289     }
290
291     @Override
292     public void onTowerFall(Tower tower) {
293         // タワー陥落でArrayList から削除。ArrayList が空になったら全タワー陥落
294         targets.getValue().remove(tower);
295         if (targets.getValue().isEmpty() && currentState.getValue() != GameState.GameClear)
296             {
297                 currentState.setValue(GameState.GameOver);
298                 gameOverTimer = new GameTimer(5000, this);
299                 gameOverTimer.start();
300                 gameOverTimer.setRepeats(false);
301                 System.out.println("gameover");
302             }
303     }
304
305     public void upTurretRangeLevel() {
306         BaseTurret turret = selectingTurret.getValue();
307         if (turret != null) {
308             turret.upRangeLevel();
309             selectingTurret.update();
310         }
311     }
312
313     public void upTurretAttackLevel() {
314         BaseTurret turret = selectingTurret.getValue();
315         if (turret != null) {
316             turret.upAttackLevel();
317             selectingTurret.update();
318         }
319     }
320
321     public void upTurretRofLevel() {
322         BaseTurret turret = selectingTurret.getValue();
323         if (turret != null) {
324             turret.upRofLevel();
325             selectingTurret.update();
326         }
327     }
328
329     public void sellTurret() {
330         BaseTurret turret = selectingTurret.getValue();
331         if (turret != null) {
332             turret.sell();
333             selectingTurret.setValue(null);
334         }
335     }
336
337     public void addScore(int additionalScore) {
338         score.setValue(score.getValue() + additionalScore);
339     }
340
341     @Override
342     public void pause() {
343         super.pause();
344         if (nextWaveTimer != null)
345             nextWaveTimer.pause();
346         if (waveEndTimer != null)
347             waveEndTimer.pause();
348         if (gameOverTimer != null)
349             gameOverTimer.pause();
350         if (gameClearTimer != null)
351             gameClearTimer.pause();
352     }
353
354     @Override
355     public void unpause() {

```

```

355     super.unpause();
356     if (nextWaveTimer != null)
357         nextWaveTimer.unpause();
358     if (waveEndTimer != null)
359         waveEndTimer.unpause();
360     if (gameOverTimer != null)
361         gameOverTimer.unpause();
362     if (gameClearTimer != null)
363         gameClearTimer.unpause();
364 }
365
366 @Override
367 public void stop() {
368     super.stop();
369     if (nextWaveTimer != null) {
370         nextWaveTimer.stop();
371     }
372     if (waveEndTimer != null) {
373         waveEndTimer.stop();
374     }
375     if (gameOverTimer != null) {
376         gameOverTimer.stop();
377     }
378     if (gameClearTimer != null) {
379         gameClearTimer.stop();
380     }
381 }
382
383 private int calcScore() {
384     return score.getValue() + (1000 * targets.getValue().size());
385 }
386 }

```

---

### ソースコード 53: uectd.game.gameScene.GameSceneView

```

1 package uectd.game.gameScene;
2
3
4 import java.io.File;
5 import java.io.IOException;
6 import java.util.ArrayList;
7 import java.util.Observable;
8 import java.util.Observer;
9 import java.awt.*;
10
11 import javax.swing.ImageIcon;
12 import javax.swing.JButton;
13 import javax.swing.JLabel;
14 import javax.swing.JPanel;
15
16 import uectd.game.ResourcePathDefines;
17 import uectd.game.gameScene.gameMain.Tower;
18 import uectd.game.gameScene.GameSceneModel.GameState;
19 import uectd.game.gameScene.gameMain.BaseTurret;
20 import uectd.gameSystem.Define;
21 import uectd.gameSystem.FatalError;
22 import uectd.gameSystem.SceneModel;
23 import uectd.gameSystem.SceneView;
24 import uectd.gameSystem.util.ImageManager;
25 import uectd.gameSystem.util.ObservableComponent;
26 import uectd.gameSystem.util.SoundManager;
27
28 public class GameSceneView extends SceneView implements Observer {
29     private Font font; // UI表示に用いるフォント
30
31     private InformationPanel informationPanel; // 情報表示のためのパネル。
32     public UpgradeTurretPanel upgradeTurretPanel; // 強化対象のタレットとその情報、タレット強
33     化のためのボタンを配置したパネル
34     public AddTurretPanel addTurretPanel; // 配置するタレットを選択するためのボタンを配置し
35     たパネル
36     private GameSceneModel gSceneModel; // モデルのインスタンス
37     private ObservableComponent<GameState> currentState; // モデルの現在の状態

```

```

37     public GameSceneView(SceneModel model) {
38         super(model);
39         this.setBackground(Color.WHITE);
40         try {
41             font = Font.createFont(Font.TRUETYPE_FONT, new File(ResourcePathDefines.
42                             FONT_PATH));
43         } catch (FontFormatException e) {
44             e.printStackTrace();
45             FatalError.quit("フォント形式エラー");
46         } catch (IOException e) {
47             e.printStackTrace();
48             FatalError.quit("フォント読み込みエラー");
49         }
50         font = font.deriveFont(40f);
51
52         gSceneModel = (GameSceneModel) model;
53         informationPanel = new InformationPanel();
54         upgradeTurretPanel = new UpgradeTurretPanel();
55         addTurretPanel = new AddTurretPanel();
56
57         this.setLayout(null);
58
59         informationPanel.setBounds(0, 0, 700, 128);
60         addTurretPanel.setBounds(0, Define.WINDOW_HEIGHT - 110 - 37, 700, 110);
61         upgradeTurretPanel.setBounds(700, 0, Define.WINDOW_WIDTH - 700 - 15, Define.
62                                     WINDOW_HEIGHT - 37);
63
64         this.add(informationPanel);
65         this.add(upgradeTurretPanel);
66         this.add(addTurretPanel);
67
68         currentState = gSceneModel.currentState;
69         currentState.addObserver(this);
70     }
71
72     private class InformationPanel extends JPanel implements Observer {
73         private static final String COUNTDOWN_STRING = "", WAVE_NUM_STRING = "W",
74             BALANCE_STRING = "所持金：
75             ", CURRENCY_UNIT_STRING = "万円", TARGET_TOWERS_STRING = "防衛対象：" ;
76         private JLabel countDownLatch, waveNumLabel, targetTowersLabel, balanceLabel;
77         private ObservableComponent<Integer> countdownNum, waveNum, balance;
78         private ObservableComponent<ArrayList<Tower>> targetTowers;
79         private Image backgroundImage;
80
81         public InformationPanel() {
82             super();
83             backgroundImage = ImageManager.getInstance().getImage(ResourcePathDefines.
84                             FRAME_IMAGE_PATH);
85             this.setOpaque(false);
86
87             countdownNum = gSceneModel.countdownNum;
88             countdownNum.addObserver(this);
89             waveNum = gSceneModel.waveNum;
90             waveNum.addObserver(this);
91             balance = gSceneModel.balance;
92             balance.addObserver(this);
93
94             countDownLatch = new JLabel(COUNTDOWN_STRING + countdownNum.getValue());
95             countDownLatch.setForeground(Color.WHITE);
96             countDownLatch.setFont(font);
97             waveNumLabel = new JLabel(WAVE_NUM_STRING + waveNum.getValue());
98             waveNumLabel.setForeground(Color.WHITE);
99             waveNumLabel.setFont(font);
100            targetTowersLabel = new JLabel(makeTargetTowersLabelString());
101            targetTowersLabel.setForeground(Color.WHITE);
102            targetTowersLabel.setFont(font);
103            balanceLabel = new JLabel(BALANCE_STRING + balance.getValue() +
104                                      CURRENCY_UNIT_STRING);
105            balanceLabel.setForeground(Color.WHITE);
106            balanceLabel.setFont(font);
107
108            GridBagLayout layout = new GridBagLayout();
109            this.setLayout(layout);

```

```

106     GridBagConstraints gbc = new GridBagConstraints();
107
108     gbc.gridx = 0;
109     gbc.gridy = 0;
110     gbc.gridwidth = 1;
111     gbc.gridheight = 1;
112     gbc.weightx = 1d;
113     gbc.fill = GridBagConstraints.BOTH;
114     gbc.insets = new Insets(15, 30, 15, 10);
115     layout.setConstraints(waveNumLabel, gbc);
116
117     gbc.gridx = 0;
118     gbc.gridy = 1;
119     gbc.gridwidth = 1;
120     gbc.gridheight = 1;
121     gbc.weightx = 1d;
122     gbc.fill = GridBagConstraints.BOTH;
123     gbc.insets = new Insets(15, 30, 15, 10);
124     layout.setConstraints(countDownLabel, gbc);
125
126     gbc.gridx = 1;
127     gbc.gridy = 1;
128     gbc.gridwidth = 2;
129     gbc.gridheight = 1;
130     gbc.weightx = 1d;
131     gbc.fill = GridBagConstraints.BOTH;
132     gbc.insets = new Insets(15, 10, 15, 10);
133     layout.setConstraints(targetTowersLabel, gbc);
134     targetTowers = gSceneModel.targets;
135     targetTowersLabel.setText(makeTargetTowersLabelString());
136
137
138     gbc.gridx = 1;
139     gbc.gridy = 0;
140     gbc.gridwidth = 1;
141     gbc.gridheight = 1;
142     gbc.weightx = 1d;
143     gbc.fill = GridBagConstraints.BOTH;
144     gbc.insets = new Insets(15, 10, 15, 10);
145     layout.setConstraints(balanceLabel, gbc);
146
147     this.add(countDownLabel);
148     this.add(waveNumLabel);
149     this.add(balanceLabel);
150     this.add(targetTowersLabel);
151 }
152
153 private String makeTargetTowersLabelString() {
154     String res = "";
155     if (targetTowers != null) {
156         var targetsList = targetTowers.getValue();
157         for (int i = 0; i < targetsList.size(); i++) {
158             if (i != 0)
159                 res += ", ";
160             res += targetsList.get(i).getName();
161         }
162     }
163     return TARGET_TOWERS_STRING + res;
164 }
165
166 @Override
167 protected void paintComponent(Graphics g) {
168     g.drawImage(backgroundImage, 0, 0, this.getWidth(), this.getHeight(), null);
169     super.paintComponent(g);
170 }
171
172 @Override
173 public void update(Observable o, Object arg) {
174     if (o == countdownNum) {
175         countDownLabel.setText(COUNTDOWN_STRING + countdownNum.getValue());
176     } else if (o == waveNum) {
177         waveNumLabel.setText(WAVE_NUM_STRING + waveNum.getValue());
178     } else if (o == targetTowers) {
179         targetTowersLabel.setText(makeTargetTowersLabelString());
180     } else if (o == balance) {

```

```

181         balanceLabel.setText(BALANCE_STRING + balance.getValue());
182     }
183   }
184 }
185
186 public class UpgradeTurretPanel extends JPanel implements Observer {
187   private ObservableComponent<BaseTurret> selectingTurret;
188   private Image backgroundImage;
189
190   private TurretImage turretImage;
191   public JButton rangeButton, powerButton, rofButton, sellButton;
192   private JLabel turretNameLabel;
193   private Font upgradePanelFont;
194
195   private class TurretImage extends JPanel {
196     private Image image, frameImage;
197
198     public TurretImage() {
199       this.setOpaque(false);
200       this.image = null;
201       this.frameImage = ImageManager.getInstance().getImage(ResourcePathDefines.
202           TURRET_FRAME_IMAGE_PATH);
203     }
204
205     @Override
206     protected void paintComponent(Graphics g) {
207       super.paintComponent(g);
208       if (image != null) {
209         int width = 0, height = 0;
210         if (this.getWidth() > this.getHeight()) {
211           height = this.getHeight();
212           width = (int) (image.getWidth(null) * ((float) height / image.
213               getHeight(null)));
214         } else {
215           width = this.getWidth();
216           height = (int) (image.getHeight(null) * ((float) width / image.
217               getWidth(null)));
218         }
219         g.drawImage(frameImage, this.getWidth() / 2 - width / 2, this.getHeight() /
220             2 - height / 2, width,
221             height, null);
222         g.drawImage(image, this.getWidth() / 2 - width / 2, this.getHeight() / 2 -
223             height / 2, width,
224             height, null);
225       }
226     }
227   }
228
229   public UpgradeTurretPanel() {
230     super();
231     this.setOpaque(false);
232
233     backgroundImage = ImageManager.getInstance().getImage(ResourcePathDefines.
234         UPGRADE_FRAME_IMAGE_PATH);
235     selectingTurret = gSceneModel.selectingTurret;
236     selectingTurret.addObserver(this);
237     this.upgradePanelFont = font.deriveFont(18);
238
239     turretImage = new TurretImage();
240     rangeButton = new JButton("範");
241     rangeButton.setFont(upgradePanelFont);
242     powerButton = new JButton("威");
243     powerButton.setFont(upgradePanelFont);
244     rofButton = new JButton("速");
245     rofButton.setFont(upgradePanelFont);
246     sellButton = new JButton("売");
247     sellButton.setFont(upgradePanelFont);
248     turretNameLabel = new JLabel("");
249     turretNameLabel.setHorizontalAlignment(JLabel.CENTER);
250     turretNameLabel.setFont(font.deriveFont(20));
251     turretNameLabel.setForeground(Color.WHITE);
252
253     GridBagLayout layout = new GridBagLayout();
254     this.setLayout(layout);

```

```

250    GridBagConstraints gbc = new GridBagConstraints();
251
252    gbc.gridx = 0;
253    gbc.gridy = 0;
254    gbc.gridwidth = 2;
255    gbc.gridheight = 2;
256    gbc.weightx = 1d;
257    gbc.weighty = 1d;
258    gbc.fill = GridBagConstraints.BOTH;
259    gbc.insets = new Insets(25, 35, 25, 35);
260    layout.setConstraints(turretImage, gbc);
261
262    gbc.gridx = 0;
263    gbc.gridy = 2;
264    gbc.gridwidth = 2;
265    gbc.gridheight = 1;
266    gbc.weightx = 1d;
267    gbc.weighty = 1d;
268    gbc.fill = GridBagConstraints.BOTH;
269    gbc.insets = new Insets(25, 25, 25, 25);
270    layout.setConstraints(turretNameLabel, gbc);
271
272    gbc.gridx = 0;
273    gbc.gridy = 3;
274    gbc.gridwidth = 2;
275    gbc.gridheight = 1;
276    gbc.weightx = 1d;
277    gbc.weighty = 1d;
278    gbc.insets = new Insets(5, 15, 5, 15);
279    layout.setConstraints(rangeButton, gbc);
280
281    gbc.gridx = 0;
282    gbc.gridy = 4;
283    gbc.gridwidth = 2;
284    gbc.gridheight = 1;
285    gbc.weightx = 1d;
286    gbc.weighty = 1d;
287    gbc.insets = new Insets(5, 15, 5, 15);
288    layout.setConstraints(powerButton, gbc);
289
290    gbc.gridx = 0;
291    gbc.gridy = 5;
292    gbc.gridwidth = 2;
293    gbc.gridheight = 1;
294    gbc.weightx = 1d;
295    gbc.weighty = 1d;
296    gbc.insets = new Insets(5, 15, 5, 15);
297    layout.setConstraints(rofButton, gbc);
298
299    gbc.gridx = 0;
300    gbc.gridy = 6;
301    gbc.gridwidth = 2;
302    gbc.gridheight = 1;
303    gbc.weightx = 1d;
304    gbc.weighty = 1d;
305    gbc.insets = new Insets(5, 15, 5, 15);
306    layout.setConstraints(sellButton, gbc);
307
308    this.add(turretImage);
309    this.add(turretNameLabel);
310    this.add(rangeButton);
311    this.add(powerButton);
312    this.add(rofButton);
313    this.add(sellButton);
314 }
315
316 @Override
317 protected void paintComponent(Graphics g) {
318     g.drawImage(backgroundImage, 0, 0, this.getWidth(), this.getHeight(), null);
319     super.paintComponent(g);
320 }
321
322 @Override
323 public void update(Observable o, Object arg) {
324     if (o == selectingTurret) {

```

```

325     BaseTurret turret = selectingTurret.getValue();
326     if (turret != null) {
327         turretImage.image = selectingTurret.getValue().getImage();
328         turretNameLabel.setText(selectingTurret.getValue().getName());
329         rangeButton.setText("範
330             :Lv" + selectingTurret.getValue().getRangeLevel() + ":" +
331             + selectingTurret.getValue().getTurretCollisionRadiusLevelupPrice
332             ());
332         powerButton.setText("威
333             :Lv" + selectingTurret.getValue().getAttackLevel() + ":" +
334             + selectingTurret.getValue().getAttackPowerLevelupPrice());
334         rofButton.setText("速
335             :Lv" + selectingTurret.getValue().getRofLevel() + ":" +
336             + selectingTurret.getValue().getRofLevelupPrice());
336     } else {
337         turretImage.image = null;
338         turretNameLabel.setText("");
339         rangeButton.setText("範");
340         powerButton.setText("威");
341         rofButton.setText("速");
342     }
343 }
344
345 public class AddTurretPanel extends JPanel implements Observer {
346     private Font priceFont;
347
348     public class TurretDisplayButton extends JButton {
349         Image image, frameImage;
350         int price;
351         JLabel priceLabel;
352         BaseTurret turret;
353         boolean isSelecting;
354
355         public TurretDisplayButton(BaseTurret turret) {
356             super(new ImageIcon(turret.getImage()));
357             this.turret = turret;
358             this.setContentAreaFilled(false);
359             this.image = turret.getImage();
360             this.price = turret.getPurchasePrice();
361             this.priceLabel = new JLabel("" + this.price);
362             this.priceLabel.setFont(priceFont);
363             this.priceLabel.setForeground(Color.BLACK);
364             this.priceLabel.setBackground(new Color(200, 200, 200, 100));
365             this.priceLabel.setAlignmentY(100);
366             this.priceLabel.setAlignmentX(0);
367             this.frameImage = ImageManager.getInstance().getImage(ResourcePathDefines.
368                 TURRET_BUTTON_IMAGE_PATH);
369             add(this.priceLabel);
370         }
371
372         @Override
373         protected void paintComponent(Graphics g) {
374             g.drawImage(frameImage, 0, 0, this.getWidth(), this.getHeight(), null);
375             if (image != null) {
376                 int width = 0, height = 0;
377                 if (isSelecting) {
378                     Graphics2D g2 = (Graphics2D) g;
379                     g2.setColor(Color.GRAY);
380                     g2.setStroke(new BasicStroke(13));
381                     g2.fillRect(0, 0, this.getWidth(), this.getHeight());
382                 }
383                 if (this.getWidth() > this.getHeight()) {
384                     height = this.getHeight();
385                     width = (int) (image.getWidth(null) * ((float) height / image.
386                         getHeight(null)));
386                 } else {
387                     width = this.getWidth();
388                     height = (int) (image.getHeight(null) * ((float) width / image.
389                         getWidth(null)));
389                 }
390                 g.drawImage(image, this.getWidth() / 2 - width / 2, this.getHeight() / 2
391                     - height / 2, width,
392                     height, null);
393             }
394         }
395     }
396 }

```

```

392         }
393     }
394 }
395
396 private Image backgroundImage;
397
398 private ObservableComponent<ArrayList<BaseTurret>> availableTurrets;
399 public ObservableComponent<ArrayList<TurretDisplayButton>> turretButtons;
400 public ObservableComponent<BaseTurret> purchasingTurret;
401
402 public AddTurretPanel() {
403     this.setOpaque(false);
404     priceFont = font.deriveFont(20.0f);
405     backgroundImage = ImageManager.getInstance().getImage(ResourcePathDefines.
406         ADD_TURRET_FRAME_IMAGE_PATH);
407     availableTurrets = gSceneModel.availableTurretList;
408     availableTurrets.addObserver(this);
409     turretButtons = new ObservableComponent<>(new ArrayList<TurretDisplayButton>());
410     purchasingTurret = gSceneModel.purchasingTurret;
411     purchasingTurret.addObserver(this);
412     makeTurretButtons();
413 }
414
415 private void makeTurretButtons() {
416     for (var button : turretButtons.getValue()) {
417         this.remove(button);
418     }
419
420     var buttonList = new ArrayList<TurretDisplayButton>();
421     for (var turret : availableTurrets.getValue()) {
422         var turretDisplayButton = new TurretDisplayButton(turret);
423         buttonList.add(turretDisplayButton);
424         turretDisplayButton.setPreferredSize(new Dimension(100, 100));
425         this.add(turretDisplayButton);
426     }
427     turretButtons.setValue(buttonList);
428 }
429
430 @Override
431 protected void paintComponent(Graphics g) {
432     super.paintComponent(g);
433     g.drawImage(backgroundImage, 0, 0, this.getWidth(), this.getHeight(), null);
434 }
435
436 @Override
437 public void update(Observable o, Object arg) {
438     if (o == availableTurrets) {
439         makeTurretButtons();
440     } else if (o == purchasingTurret) {
441         for (var turretButton : turretButtons.getValue()) {
442             turretButton.isSelecting = (turretButton.turret == purchasingTurret.
443                 getValue());
444         }
445     }
446 }
447
448 @Override
449 public void update(Observable o, Object arg) {
450     if (o == currentState) {
451         if (currentState.getValue() == GameState.OnWave) {
452             switch (gSceneModel.waveNum.getValue()) {
453                 case 1:
454                     SoundManager.getInstance().play(ResourcePathDefines.WAVE1_SOUND_PATH
455                         );
456                     break;
457                 case 4:
458                     SoundManager.getInstance().play(ResourcePathDefines.WAVE3_SOUND_PATH
459                         );
460                     break;
461                 default:
462                     break;
463             }
464         }
465     }
466 }

```

```
462         }
463     }
464 }
465 }
```

---

#### ソースコード 54: uectd.game.pauseScene.PauseScene

```
1 package uectd.game.pauseScene;
2
3 import uectd.gameSystem.*;
4
5 public class PauseScene extends Scene {
6
7     public PauseScene(SceneChangeListener sceneChangeListener) {
8         super(sceneChangeListener);
9         this.model = new PauseSceneModel(sceneChangeListener);
10        this.view = new PauseSceneView(model);
11        this.controller = new PauseSceneController(model, view);
12    }
13
14 }
```

---

#### ソースコード 55: uectd.game.pauseScene.PauseSceneController

```
1 package uectd.game.pauseScene;
2
3 import java.awt.event.*;
4
5 import uectd.gameSystem.SceneController;
6 import uectd.gameSystem.SceneModel;
7 import uectd.gameSystem.SceneView;
8
9 public class PauseSceneController extends SceneController {
10
11     public PauseSceneController(SceneModel sceneModel, SceneView view) {
12         super(sceneModel, view);
13         ((PauseSceneView) view).unpauseButton.addActionListener(this);
14         ((PauseSceneView) view).exitButton.addActionListener(this);
15     }
16
17     @Override
18     public void mouseClicked(MouseEvent e) {
19     }
20
21     @Override
22     public void mousePressed(MouseEvent e) {
23     }
24
25     @Override
26     public void mouseReleased(MouseEvent e) {
27     }
28
29     @Override
30     public void mouseEntered(MouseEvent e) {
31     }
32
33     @Override
34     public void mouseExited(MouseEvent e) {
35     }
36
37     @Override
38     public void mouseDragged(MouseEvent e) {
39     }
40
41     @Override
42     public void mouseMoved(MouseEvent e) {
43     }
44
45     @Override
46     public void keyTyped(KeyEvent e) {
47     }
48
49     @Override
```

```
50     public void keyPressed(KeyEvent e) {
51 }
52
53     @Override
54     public void keyReleased(KeyEvent e) {
55 }
56
57     @Override
58     public void actionPerformed(ActionEvent e) {
59         super.actionPerformed(e);
60         if (e.getSource() == ((PauseSceneView) view).unpauseButton) {
61             ((PauseSceneModel) model).unpause();
62         } else if (e.getSource() == ((PauseSceneView) view).exitButton) {
63             ((PauseSceneModel) model).exit();
64         }
65     }
66
67 }
```

---

ソースコード 56: uectd.game.pauseScene.PauseSceneModel

```
1 package uectd.game.pauseScene;
2
3 import uectd.game.titleScene.TitleScene;
4 import uectd.gameSystem.SceneChangeListener;
5 import uectd.gameSystem.SceneModel;
6
7 public class PauseSceneModel extends SceneModel {
8
9     public PauseSceneModel(SceneChangeListener sceneChangeListener) {
10         super(sceneChangeListener);
11     }
12
13     @Override
14     public void calc(float deltaTime) {
15     }
16
17     public void unpause() {
18         sceneChangeListener.scenePopped(null);
19     }
20
21     public void exit() {
22         sceneChangeListener.sceneChanged(new TitleScene(sceneChangeListener), true, null
23             );
24     }
25 }
```

---

ソースコード 57: uectd.game.pauseScene.PauseSceneView

```
1 package uectd.game.pauseScene;
2
3 import javax.swing.JButton;
4 import javax.swing.JLabel;
5
6 import uectd.game.ResourcePathDefines;
7 import uectd.gameSystem.FatalError;
8 import uectd.gameSystem.SceneModel;
9 import uectd.gameSystem.SceneView;
10 import uectd.gameSystem.util.ImageManager;
11
12 import java.awt.*;
13 import java.io.File;
14 import java.io.IOException;
15
16 public class PauseSceneView extends SceneView {
17
18     public JButton unpauseButton, exitButton;
19
20     public Image backgroundImage;
21
22     private Font font;
23 }
```

```

24     public PauseSceneView(SceneModel model) {
25         super(model);
26         try {
27             font = Font.createFont(Font.TRUETYPE_FONT, new File(ResourcePathDefines.
28                                     FONT_PATH));
29         } catch (FontFormatException e) {
30             e.printStackTrace();
31             FatalError.quit("フォント形式エラー");
32         } catch (IOException e) {
33             e.printStackTrace();
34             FatalError.quit("フォント読み込みエラー");
35         }
36         font = font.deriveFont(30f);
37
38         this.setLayout(null);
39
40         JLabel pauseLabel = new JLabel("Pause");
41         pauseLabel.setFont(font);
42         pauseLabel.setBounds(100, 40, 200, 60);
43         this.add(pauseLabel);
44
45         unpauseButton = new JButton("Continue");
46         unpauseButton.setFont(font);
47         unpauseButton.setBounds(100, 100, 200, 60);
48         unpauseButton.setContentAreaFilled(false);
49         this.add(unpauseButton);
50
51         exitButton = new JButton("Quit");
52         exitButton.setFont(font);
53         exitButton.setBounds(100, 160, 200, 60);
54         exitButton.setContentAreaFilled(false);
55         this.add(exitButton);
56     }
57
58     @Override
59     public void start() {
60         backgroundImage = ImageManager.getInstance().getImage(ResourcePathDefines.
61                                     BACKGROUND_IMAGE);
62     }
63
64     @Override
65     public void draw(Graphics g) {
66         super.draw(g);
67         g.drawImage(backgroundImage, 0, 0,
68                     backgroundImage.getWidth(null) * getHeight() / backgroundImage.getHeight(
69                                     null), getHeight(), null);
70     }
71 }
```

ソースコード 58: uectd.game.ResourcePathDefines

```

1 package uectd.game;
2
3 public class ResourcePathDefines {
4     public static final String BACKGROUND_IMAGE = "data/UECPicture.png";
5     public static final String FONT_PATH = "data/nicoca_v1.ttf";
6     public static final String FRAME_IMAGE_PATH = "data/frame.png";
7     public static final String UPGRADE_FRAME_IMAGE_PATH = "data/frame3.png";
8     public static final String FRAME_VERTICAL_IMAGE_PATH = "data/avg2_vert.png";
9     public static final String ZOMBIE_IMAGES = "data/Enemy1.png";
10    public static final String GOBLIN_IMAGES = "data/goblin.png";
11    public static final String ADD_TURRET_FRAME_IMAGE_PATH = "data/frame2.png";
12    public static final String TURRET_FRAME_IMAGE_PATH = "data/frame4.png";
13    public static final String CANNONBALL_IMAGE_PATH = "data/cannonball.png";
14    public static final String TURRET_ATTACK_TO_ENEMY_IMAGES = "data/turretAttackToEnemy
15                                     .png";
16    public static final String ENEMY_SPAWNER_DEFINES_PATH = "data/EnemySpawnerDefines.
17                                     dat";
17    public static final String TURRET_SOCKET_IMAGE = "data/turretSocket.png";
18    public static final String TURRET_BUTTON_IMAGE_PATH = "data/frame4.png";
```

```

18     public static final String GRAPH_PATH = "data/graph.dat";
19     public static final String TURRET_SOCKET_POSITION_PATH = "data/turretSocketPosition.
20         dat";
21     public static final String CANDIDATE_VERTEX_LIST_PATH = "data/candidateVertexList.
22         dat";
23     public static final String UEC_MAP_GRAPH = "data/UECMapGraph.dat";
24     public static final String ENEMY_SEARCH_FIELD_IMAGE = "data/enemySearchField.png";
25     public static final String TOWER_CANDIDATE_PATH = "data/towerCandidate.dat";
26     public static final String EFFECT_BOMB_IMAGES = "data/bombEffect.png";
27     public static final String ENEMY1_IMAGES = "data/Enemy1.png";
28     public static final String ENEMY2_IMAGES = "data/Enemy2.png";
29     public static final String ENEMY3_IMAGES = "data/Enemy3.png";
30     public static final String LASER_IMAGE_PATH = "data/Laser.png";
31     public static final String CANNON_TURRET_IMAGES = "data/CannonTurret.png";
32     public static final String LASER_TURRET_IMAGE = "data/LaserTower.png";
33     public static final String BALLISTA_TURRET_IMAGES = "data/Ballista.png";
34     public static final String ARROW_IMAGES = "data/Arrow.png";
35     public static final String ENEMY_SPAWNER_MARK_IMAGE = "data/mahouzin.png";
36     public static final String ENEMY_SUMMON_IMAGES = "data/Summon.png";
37     public static final String TITLE_IMAGE = "data/screen/StartScreen3.jpg";
38     public static final String CLEAR_IMAGE = "data/screen/GameClear1.jpg";
39     public static final String GAME_OVER_IMAGE = "data/screen/gameover_screen1.jpg";
40
41     public static final String START_SCREEN_SOUND_PATH = "data/sound/StartScreen.wav";
42     public static final String GAME_CLEAR_SOUND_PATH = "data/sound/GameClear.wav";
43     public static final String GAME_OVER_SOUND_PATH = "data/sound/GameOver.wav";
44     public static final String WAVE1_SOUND_PATH = "data/sound/Wave1.wav";
45     public static final String WAVE2_SOUND_PATH = "data/sound/Wave2.wav";
46     public static final String WAVE3_SOUND_PATH = "data/sound/Wave3.wav";
47     public static final String CANNON_TURRET_SOUND_PATH = "data/sound/SE/Cannon1.wav";
48     public static final String COUNTDOWN_SOUND_PATH = "data/sound/SE/Countdown.wav";
49     public static final String TOWER_COLLAPSE_SOUND_PATH = "data/sound/SE/TowerCollapse.
50         wav";
51     public static final String ENEMY_DIE_SOUND_PATH = "data/sound/SE/EnemyDie.wav";
52     public static final String LASER_SOUND_PATH = "data/sound/SE/Laser.wav";
53     public static final String SUMMON_SOUND_PATH = "data/sound/SE/Summon.wav";
54     public static final String TURRET_ATTACK_TO_ENEMY_SOUND_PATH = "data/sound/SE/
55         EnemyAttack.wav";
56     public static final String ENEMY_DIE_EFFECT_IMAGES = "data/pipo-btleffect175_192.png
57         ";
58 }

```

---

ソースコード 59: uectd.game.resultScene.ResultScene

```

1 package uectd.game.resultScene;
2
3 import uectd.gameSystem.Scene;
4 import uectd.gameSystem.SceneChangeListener;
5
6 public class ResultScene extends Scene {
7
8     public ResultScene(SceneChangeListener sceneChangeListener) {
9         super(sceneChangeListener);
10        this.model = new ResultSceneModel(sceneChangeListener);
11        this.view = new ResultSceneView(model);
12        this.controller = new ResultSceneController(model, view);
13    }
14
15 }

```

---

ソースコード 60: uectd.game.resultScene.ResultSceneController

```

1 package uectd.game.resultScene;
2
3 import java.awt.event.*;
4
5 import uectd.gameSystem.SceneController;
6 import uectd.gameSystem.SceneModel;
7 import uectd.gameSystem.SceneView;
8

```

```

9 public class ResultSceneController extends SceneController {
10
11     public ResultSceneController(SceneModel sceneModel, SceneView view) {
12         super(sceneModel, view);
13     }
14
15     @Override
16     public void mouseClicked(MouseEvent e) {
17     }
18
19     @Override
20     public void mousePressed(MouseEvent e) {
21     }
22
23     @Override
24     public void mouseReleased(MouseEvent e) {
25     }
26
27     @Override
28     public void mouseEntered(MouseEvent e) {
29     }
30
31     @Override
32     public void mouseExited(MouseEvent e) {
33     }
34
35     @Override
36     public void mouseDragged(MouseEvent e) {
37     }
38
39     @Override
40     public void mouseMoved(MouseEvent e) {
41     }
42
43     @Override
44     public void keyTyped(KeyEvent e) {
45     }
46
47     @Override
48     public void keyPressed(KeyEvent e) {
49         int keyCode = e.getKeyCode();
50         if (keyCode == KeyEvent.VK_ESCAPE) {
51             ((ResultSceneModel) model).toTitle();
52         }
53     }
54
55     @Override
56     public void keyReleased(KeyEvent e) {
57     }
58
59     @Override
60     public void actionPerformed(ActionEvent e) {
61         super.actionPerformed(e);
62     }
63 }

```

---

#### ソースコード 61: uectd.game.resultScene.ResultSceneModel

```

1 package uectd.game.resultScene;
2
3 import uectd.game.titleScene.TitleScene;
4 import uectd.gameSystem.SceneChangeListener;
5 import uectd.gameSystem.SceneModel;
6 import uectd.gameSystem.util.Intent;
7
8 public class ResultSceneModel extends SceneModel {
9     public int score;
10    public boolean cleared;
11
12    public ResultSceneModel(SceneChangeListener sceneChangeListener) {
13        super(sceneChangeListener);
14    }
15
16    @Override

```

```

17     public void start(Intent intent) {
18         super.start(intent);
19         cleared = intent.getBooleanValue("Cleared");
20         score = intent.getIntegerValue("Score");
21     }
22
23     public void toTitle() {
24         sceneChangeListener.sceneChanged(new TitleScene(sceneChangeListener), null);
25     }
26
27 }
```

---

ソースコード 62: uectd.game.resultScene.ResultSceneView

```

1 package uectd.game.resultScene;
2
3 import java.awt.Color;
4 import java.io.File;
5 import java.io.IOException;
6
7 import javax.swing.JLabel;
8 import java.awt.*;
9
10 import uectd.game.ResourcePathDefines;
11 import uectd.gameSystem.FatalError;
12 import uectd.gameSystem.SceneModel;
13 import uectd.gameSystem.SceneView;
14 import uectd.gameSystem.util.GameSound;
15 import uectd.gameSystem.util.ImageManager;
16
17 public class ResultSceneView extends SceneView {
18
19     private Font font;
20     private Image backgroundImage;
21     public static GameSound gameSound;
22
23     public ResultSceneView(SceneModel model) {
24         super(model);
25         this.setBackground(Color.BLACK);
26         try {
27             font = Font.createFont(Font.TRUETYPE_FONT, new File(ResourcePathDefines.
28                 FONT_PATH));
29         } catch (FontFormatException e) {
30             e.printStackTrace();
31             FatalError.quit("フォント形式エラー");
32         } catch (IOException e) {
33             e.printStackTrace();
34             FatalError.quit("フォント読み込みエラー");
35         }
36     }
37
38     @Override
39     public void start() {
40         if (((ResultSceneModel) model).cleared) {
41             backgroundImage = ImageManager.getInstance().getImage(ResourcePathDefines.
42                 CLEAR_IMAGE);
43
44             JLabel gameOverLabel = new JLabel("GameClear!");
45             gameOverLabel.setForeground(Color.RED);
46             gameOverLabel.setFont(font.deriveFont(100f));
47             gameOverLabel.setVerticalAlignment(JLabel.CENTER);
48             this.add(gameOverLabel);
49
50             JLabel messageLabel = new JLabel("あなたは電通大をゾンビから守り切った!");
51             messageLabel.setForeground(Color.RED);
52             messageLabel.setFont(font.deriveFont(50f));
53             messageLabel.setVerticalAlignment(JLabel.CENTER);
54             this.add(messageLabel);
55
56             JLabel scoreLabel = new JLabel("Score:" + ((ResultSceneModel) model).score);
57             scoreLabel.setForeground(Color.RED);
58             scoreLabel.setFont(font.deriveFont(70f));
59             scoreLabel.setVerticalAlignment(JLabel.CENTER);
60             this.add(scoreLabel);
61         }
62     }
63 }
```

```

59     JLabel messageLabel2 = new JLabel("エスケープキーでタイトルメニューへ");
60     messageLabel2.setForeground(Color.RED);
61     messageLabel2.setFont(font.deriveFont(70f));
62     messageLabel2.setVerticalAlignment(JLabel.CENTER);
63     this.add(messageLabel2);
64
65     gameSound = new GameSound(ResourcePathDefines.GAME_CLEAR_SOUND_PATH);
66     gameSound.init();
67     gameSound.start();
68     gameSound.volume(3);
69 } else {
70     backgroundImage = ImageManager.getInstance().getImage(ResourcePathDefines.
71         GAME_OVER_IMAGE);
72     JLabel gameOverLabel = new JLabel("GameOver...");
73     gameOverLabel.setForeground(Color.RED);
74     gameOverLabel.setFont(font.deriveFont(100f));
75     gameOverLabel.setVerticalAlignment(JLabel.CENTER);
76     this.add(gameOverLabel);
77     JLabel messageLabel = new JLabel("電通大はゾンビたちの手に落ちた... ");
78     messageLabel.setForeground(Color.RED);
79     messageLabel.setFont(font.deriveFont(70f));
80     messageLabel.setVerticalAlignment(JLabel.CENTER);
81     this.add(messageLabel);
82     JLabel scoreLabel = new JLabel("Score:" + ((ResultSceneModel) model).score);
83     scoreLabel.setForeground(Color.RED);
84     scoreLabel.setFont(font.deriveFont(70f));
85     scoreLabel.setVerticalAlignment(JLabel.CENTER);
86     this.add(scoreLabel);
87     JLabel messageLabel2 = new JLabel("エスケープキーでタイトルメニューへ");
88     messageLabel2.setForeground(Color.RED);
89     messageLabel2.setFont(font.deriveFont(70f));
90     messageLabel2.setVerticalAlignment(JLabel.CENTER);
91     this.add(messageLabel2);
92
93     gameSound = new GameSound(ResourcePathDefines.GAME_OVER_SOUND_PATH);
94     gameSound.init();
95     gameSound.start();
96     gameSound.volume(3);
97 }
98
99 @Override
100 public void draw(Graphics g) {
101     g.drawImage(backgroundImage, 0, 0, this.getWidth(), this.getHeight(), null);
102     super.draw(g);
103 }
104
105 }
106 }
```

ソースコード 63: uectd.game.titleScene.TitleScene

```

1 package uectd.game.titleScene;
2
3 import uectd.gameSystem.*;
4
5 public class TitleScene extends Scene {
6
7     public TitleScene(SceneChangeListener sceneChangeListener) {
8         super(sceneChangeListener);
9         this.model = new TitleSceneModel(sceneChangeListener);
10        this.view = new TitleSceneView(model);
11        this.controller = new TitleSceneController(model, view);
12    }
13
14 }
```

ソースコード 64: uectd.game.titleScene.TitleSceneController

```

1 package uectd.game.titleScene;
2
3 import java.awt.event.*;
4
```

```

5 import uectd.gameSystem.SceneController;
6 import uectd.gameSystem.SceneModel;
7 import uectd.gameSystem.SceneView;
8
9 public class TitleSceneController extends SceneController {
10
11     public TitleSceneController(SceneModel sceneModel, SceneView view) {
12         super(sceneModel, view);
13         ((TitleSceneView) view).startButton.addActionListener(this);
14         ((TitleSceneView) view).exitButton.addActionListener(this);
15     }
16
17     @Override
18     public void mouseClicked(MouseEvent e) {
19     }
20
21     @Override
22     public void mousePressed(MouseEvent e) {
23     }
24
25     @Override
26     public void mouseReleased(MouseEvent e) {
27     }
28
29     @Override
30     public void mouseEntered(MouseEvent e) {
31     }
32
33     @Override
34     public void mouseExited(MouseEvent e) {
35     }
36
37     @Override
38     public void mouseDragged(MouseEvent e) {
39     }
40
41     @Override
42     public void mouseMoved(MouseEvent e) {
43     }
44
45     @Override
46     public void keyTyped(KeyEvent e) {
47     }
48
49     @Override
50     public void keyPressed(KeyEvent e) {
51     }
52
53     @Override
54     public void keyReleased(KeyEvent e) {
55     }
56
57     @Override
58     public void actionPerformed(ActionEvent e) {
59         if (e.getSource() == ((TitleSceneView) view).startButton) {
60             TitleSceneView.gameSound.stop();
61             ((TitleSceneModel) model).gameStart();
62         } else if (e.getSource() == ((TitleSceneView) view).exitButton) {
63             TitleSceneView.gameSound.stop();
64             ((TitleSceneModel) model).exit();
65         }
66     }
67 }

```

---

ソースコード 65: uectd.game.titleScene.TitleSceneModel

```

1 package uectd.game.titleScene;
2
3 import uectd.game.gameScene.GameScene;
4 import uectd.gameSystem.SceneChangeListener;
5 import uectd.gameSystem.SceneModel;
6 import uectd.gameSystem.util.Intent;
7
8 public class TitleSceneModel extends SceneModel {
9

```

```

10  public TitleSceneModel(SceneChangeListener sceneChangeListener) {
11      super(sceneChangeListener);
12  }
13
14  @Override
15  public void calc(float deltaTime) {
16  }
17
18
19  public void gameStart() {
20      var intent = new Intent();
21      intent.setBooleanValue("CannonTurret", true);
22      intent.setBooleanValue("LaserTurret", true);
23      intent.setBooleanValue("BallistaTurret", true);
24      sceneChangeListener.sceneChanged(new GameScene(sceneChangeListener), intent);
25  }
26
27  public void exit() {
28      System.exit(0);
29  }
30
31 }
```

---

### ソースコード 66: uectd.game.titleScene.TitleSceneView

```

1 package uectd.game.titleScene;
2
3 import javax.swing.JButton;
4 import javax.swing.JLabel;
5 import javax.swing.JPanel;
6
7 import uectd.game.ResourcePathDefines;
8 import uectd.gameSystem.Define;
9 import uectd.gameSystem.FatalError;
10 import uectd.gameSystem.SceneModel;
11 import uectd.gameSystem.SceneView;
12 import uectd.gameSystem.util.GameSound;
13 import uectd.gameSystem.util.ImageManager;
14
15 import java.awt.*;
16 import java.io.File;
17 import java.io.IOException;
18
19 public class TitleSceneView extends SceneView {
20
21     public JButton startButton, exitButton;
22     private Image image;
23     public static GameSound gameSound;
24     private Font font;
25
26     public TitleSceneView(SceneModel model) {
27         super(model);
28         try {
29             font = Font.createFont(Font.TRUETYPE_FONT, new File(ResourcePathDefines.
29             FONT_PATH));
30         } catch (FontFormatException e) {
31             e.printStackTrace();
32             FatalError.quit("フォント形式エラー");
33         } catch (IOException e) {
34             e.printStackTrace();
35             FatalError.quit("フォント読み込みエラー");
36         }
37         font = font.deriveFont(30f);
38
39         this.setLayout(null);
40         startButton = new JButton("Start");
41         startButton.setBounds(Define.HALF_WINDOW_WIDTH - 200, 400, 400, 80);
42         startButton.setBackground(new Color(100, 100, 100));
43         startButton.setFont(font);
44         this.add(startButton);
45         exitButton = new JButton("Exit");
46         exitButton.setBounds(Define.HALF_WINDOW_WIDTH - 200, 550, 400, 80);
47         exitButton.setBackground(new Color(100, 100, 100));
48         exitButton.setFont(font);
```

```
49         this.add(exitButton);
50         // BGM
51         gameSound = new GameSound(ResourcePathDefines.START_SCREEN_SOUND_PATH);
52         gameSound.init();
53         gameSound.loop();
54         gameSound.volume(3);
55         image = ImageManager.getInstance().getImage(ResourcePathDefines.TITLE_IMAGE);
56     }
57
58     @Override
59     public void draw(Graphics g) {
60         super.draw(g);
61         g.drawImage(image, 0, 0, this.getWidth(), this.getHeight(), null);
62     }
63
64     @Override
65     public void start() {
66     }
67 }
```

---

ソースコード 67: uectd.gameSystem.Define

```
1 package uectd.gameSystem;
2
3 public class Define {
4     public static final int WINDOW_WIDTH = 1024;
5     public static final int WINDOW_HEIGHT = 768;
6     public static final int HALF_WINDOW_WIDTH = WINDOW_WIDTH / 2;
7     public static final int HALF_WINDOW_HEIGHT = WINDOW_HEIGHT / 2;
8     public static final int FPS = 30;
9     public static final String WINDOW_TITLE = "UEC-TowerDefense";
10 }
```

---

ソースコード 68: uectd.gameSystem.FatalError

```
1 package uectd.gameSystem;
2
3 import javax.swing.JFrame;
4 import javax.swing.JOptionPane;
5
6 public class FatalError {
7     public static void quit(String message) {
8         JFrame frame = new JFrame();
9         JOptionPane.showMessageDialog(frame, message, "エラー",
10             JOptionPane.ERROR_MESSAGE);
11     }
12 }
```

---

ソースコード 69: uectd.gameSystem.GameObject

```
1 package uectd.gameSystem;
2
3 import java.util.List;
4 import java.util.Stack;
5 import java.util.ArrayList;
6
7 import uectd.gameSystem.util.*;
8
9 public class GameObject implements Cloneable {
10     protected GameObject root; // 木構造の根のGameObject
11     public GameObject parent; // 同じく親
12     public List<GameObject> children; // 同じく子
13
14     private boolean enabled; // オブジェクトが有効であることを示すフラグ。有効な時のみ処理が行
15     われる。
16     public Vector2 position, offset; // オブジェクトの存在する座標の基準点とそこからずれる距
17     離。相対座標を用いて表現した方が適切な場合はoffsetを用いる。
18     public int depth; // 深さ。Z座標とも言い換えられる。
19 }
```

```

18     private List<GameObject> addObjectList, removeObjectList; // 子オブジェクトを追加・削除
19     //する時のバッファ
20
21     public Collider collider; // 当たり判定
22
23     public GameObject(GameObject root, GameObject parent, Vector2 position) {
24         this.enabled = true;
25         this.root = root;
26         this.parent = parent;
27         this.position = position;
28         this.offset = new Vector2();
29         this.children = new ArrayList<>();
30         this.addObjectList = new ArrayList<>();
31         this.removeObjectList = new ArrayList<>();
32     }
33
34     public GameObject(GameObject root, GameObject parent) {
35         this(root, parent, new Vector2());
36     }
37
38     public boolean getEnabled() {
39         return enabled;
40     }
41
42     public void setEnabled(boolean enabled) {
43         this.enabled = enabled;
44         if (enabled) {
45             onEnabled();
46         }
47     }
48
49     public void destroy() {
50         this.enabled = false;
51         if (parent != null)
52             parent.removeChild(this);
53         onDestroyed();
54         for (var child : children) {
55             child.destroy();
56         }
57     }
58
59     public void onDestroyed() {
60
61     public void addChild(GameObject child) {
62         addObjectList.add(child);
63     }
64
65     public void removeChild(GameObject child) {
66         removeObjectList.add(child);
67     }
68
69     protected void _start() {
70         start();
71         if (enabled)
72             onEnabled();
73
74         for (GameObject gameObject : children) {
75             if (gameObject.enabled)
76                 gameObject._start();
77         }
78
79         for (GameObject gameObject : addObjectList) {
80             children.add(gameObject);
81         }
82         for (GameObject gameObject : removeObjectList) {
83             children.remove(gameObject);
84         }
85         addObjectList.clear();
86         removeObjectList.clear();
87     }
88
89     public void start() {
90
91

```

```

92     public void _calc(float deltaTime) {
93         calc(deltaTime);
94
95         for (GameObject gameObject : children) {
96             if (gameObject.enabled)
97                 gameObject._calc(deltaTime);
98         }
99
100        for (GameObject gameObject : addObjectList) {
101            children.add(gameObject);
102        }
103        for (GameObject gameObject : removeObjectList) {
104            gameObject.onSummoned();
105        }
106        for (GameObject gameObject : removeObjectList) {
107            children.remove(gameObject);
108        }
109        addObjectList.clear();
110        removeObjectList.clear();
111    }
112
113    protected void onSummoned() {
114    }
115
116    public void calc(float deltaTime) {
117    }
118
119    public void _pause() {
120        pause();
121
122        for (GameObject gameObject : children) {
123            if (gameObject.enabled)
124                gameObject._pause();
125        }
126
127        for (GameObject gameObject : addObjectList) {
128            children.add(gameObject);
129        }
130        for (GameObject gameObject : removeObjectList) {
131            children.remove(gameObject);
132        }
133        addObjectList.clear();
134        removeObjectList.clear();
135    }
136
137    public void pause() {
138    }
139
140    public void _unpause() {
141        unpause();
142
143        for (GameObject gameObject : children) {
144            if (gameObject.enabled)
145                gameObject._unpause();
146        }
147
148        for (GameObject gameObject : addObjectList) {
149            children.add(gameObject);
150        }
151        for (GameObject gameObject : removeObjectList) {
152            children.remove(gameObject);
153        }
154        addObjectList.clear();
155        removeObjectList.clear();
156    }
157
158    public void unpause() {
159    }
160
161    public void onEnabled() {
162    }
163
164    public <G> GameObject findChild(Class<G> class_) {
165        for (GameObject child : children) {
166            if (child.getClass().isAssignableFrom(class_)) {

```

```

167         return child;
168     }
169 }
170 return null;
171 }
172
173 public <G> GameObject findChildFromChildren(Class<G> class_) {
174     var res = findChild(class_);
175     if (res != null)
176         return res;
177     for (GameObject child : children) {
178         GameObject go = child.findChild(class_);
179         if (go != null)
180             return go;
181     }
182     return null;
183 }
184
185 public <G> ArrayList<GameObject> findChildren(Class<G> class_) {
186     var res = new ArrayList<GameObject>();
187     for (GameObject child : children) {
188         if (child.getClass().isAssignableFrom(class_))
189             res.add(child);
190     }
191 }
192 return res;
193 }
194
195 public <G> ArrayList<GameObject> findChildrenFromChildren(Class<G> class_) {
196     var res = new ArrayList<GameObject>();
197
198     Stack<GameObject> stack = new Stack<>();
199
200     stack.push(this);
201     while (!stack.empty()) {
202         var currentGameObject = stack.pop();
203         if (currentGameObject.getClass().isAssignableFrom(class_))
204             res.add(currentGameObject);
205         for (var nextGameObject : currentGameObject.children) {
206             stack.push(nextGameObject);
207         }
208     }
209     return res;
210 }
211
212 @Override
213 public GameObject clone() {
214     GameObject gameObject = null;
215
216     try {
217         gameObject = (GameObject) super.clone();
218         gameObject.root = this.root;
219         gameObject.parent = this.parent;
220         gameObject.children = new ArrayList<>();
221         gameObject.enabled = this.enabled;
222         gameObject.addObjectList = new ArrayList<>();
223         gameObject.removeObjectList = new ArrayList<>();
224         gameObject.position = this.position.clone();
225         gameObject.offset = this.offset.clone();
226         if (gameObject.collider != null) {
227             gameObject.collider = this.collider.clone();
228             gameObject.collider.gameObject = gameObject;
229         }
230     } catch (CloneNotSupportedException ce) {
231         ce.printStackTrace();
232         FatalError.quit("オブジェクトのクローンに失敗しました");
233     }
234     return gameObject;
235 }
236 }

```

ソースコード 70: uectd.gameSystem.MainFrame

---

1 package uectd.gameSystem;

```

2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.util.Stack;
6
7 import uectd.game.titleScene.*;
8 import uectd.gameSystem.util.Intent;
9
10 public class MainFrame extends JFrame implements SceneChangeListener {
11
12     private JPanel mainPanel; // 基本的に全てのSwingの要素はこのパネルの下に置く
13     private CardLayout layout; // カードレイアウトでシーンを切り替える
14
15     private Stack<Scene> sceneStack; // シーンを保存しておくスタック
16
17     public MainFrame() {
18         sceneStack = new Stack<>();
19
20         SwingUtilities.invokeLater(new Runnable() {
21             public void run() {
22                 createWindow();
23             }
24         });
25     }
26
27
28     public static void main(String argv[]) {
29         new MainFrame();
30     }
31
32     private void createWindow() {
33         this.mainPanel = new JPanel();
34         this.layout = new CardLayout();
35         this.mainPanel.setLayout(layout);
36
37         this.add(mainPanel);
38
39         this.setTitle(Define.WINDOW_TITLE);
40         this.setSize(Define.WINDOW_WIDTH, Define.WINDOW_HEIGHT);
41         this.setLocationRelativeTo(null);
42         this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
43         this.setResizable(false);
44
45         TitleScene scene = new TitleScene(this);
46         sceneStack.push(scene);
47         this.mainPanel.add(scene.view, scene.toString());
48         scene.controller.start(null);
49
50         this.setVisible(true);
51     }
52
53     @Override
54     public void sceneChanged(Scene scene, boolean stackClearFlag, Intent intent) {
55         sceneStack.peek().controller.stop();
56         this.mainPanel.remove(sceneStack.peek().view);
57         sceneStack.pop();
58         if (stackClearFlag) {
59             while (!sceneStack.empty()) {
60                 sceneStack.peek().controller.stop();
61                 this.mainPanel.remove(sceneStack.peek().view);
62                 sceneStack.pop();
63             }
64         }
65
66         sceneStack.push(scene);
67         this.mainPanel.add(scene.view, scene.view.toString());
68         layout.addLayoutComponent(scene.view, null);
69         layout.show(mainPanel, scene.view.toString());
70
71         this.mainPanel.revalidate();
72         scene.controller.start(intent);
73     }
74
75     @Override
76     public void scenePopped(Intent intent) {

```

```

77     sceneStack.peek().controller.stop();
78     this.mainPanel.remove(sceneStack.peek().view);
79     this.mainPanel.revalidate();
80     sceneStack.pop();
81     sceneStack.peek().controller.unpause(intent);
82     layout.addLayoutComponent(sceneStack.peek().view, null);
83     layout.show(mainPanel, sceneStack.peek().toString());
84 }
85
86 @Override
87 public void scenePushed(Scene scene, Intent intent) {
88
89     sceneStack.peek().controller.pause();
90     sceneStack.push(scene);
91     this.mainPanel.add(scene.view, scene.view.toString());
92     layout.addLayoutComponent(scene.view, null);
93     layout.show(mainPanel, scene.view.toString());
94     layout.last(mainPanel);
95
96     this.mainPanel.revalidate();
97     scene.controller.start(intent);
98 }
99
100
101 @Override
102 public void sceneChanged(Scene scene, Intent intent) {
103     sceneChanged(scene, false, intent);
104 }
105
106 }

```

---

#### ソースコード 71: uectd.gameSystem.Scene

```

1 package uectd.gameSystem;
2
3 public abstract class Scene {
4     protected SceneModel model;
5     protected SceneView view;
6     protected SceneController controller;
7     protected SceneChangeListener sceneChangeListener;
8
9     public Scene(SceneChangeListener sceneChangeListener) {
10         this.sceneChangeListener = sceneChangeListener;
11     }
12 }

```

---

#### ソースコード 72: uectd.gameSystem.SceneChangeListener

```

1 package uectd.gameSystem;
2
3 import uectd.gameSystem.util.Intent;
4
5 public interface SceneChangeListener {
6     void sceneChanged(Scene scene, boolean stackClearFlag, Intent intent);
7
8     void sceneChanged(Scene scene, Intent intent);
9
10    void scenePopped(Intent intent);
11
12    void scenePushed(Scene scene, Intent intent);
13 }

```

---

#### ソースコード 73: uectd.gameSystem.SceneController

```

1 package uectd.gameSystem;
2
3 import java.awt.event.*;
4 import java.util.Stack;
5 import java.awt.*;
6
7 import javax.swing.Timer;
8

```

```

9 import uectd.gameSystem.util.IClickable;
10 import uectd.gameSystem.util.Intent;
11 import uectd.gameSystem.util.Vector2;
12
13 public abstract class SceneController implements MouseListener, MouseMotionListener,
14     KeyListener, ActionListener {
14     protected SceneModel model;
15     protected SceneView view;
16     protected javax.swing.Timer timer;
17
18     private long prevTime = 0; // フレーム時間計算用、前のフレームの時刻
19
20     public SceneController(SceneModel model, SceneView view) {
21         this.model = model;
22         this.view = view;
23         this.timer = new Timer(1, this);
24         this.timer.start();
25
26         view.getPanel().addMouseListener(this);
27         view.getPanel().addMouseMotionListener(this);
28         view.getPanel().addKeyListener(this);
29         view.getPanel().setFocusable(true);
30         view.getPanel().requestFocus();
31         KeyboardFocusManager.getCurrentKeyboardFocusManager().clearGlobalFocusOwner();
32
33         prevTime = System.nanoTime();
34     }
35
36     public void start(Intent intent) {
37         model.start(intent);
38         model.rootGameObject._start();
39         view.start();
40     }
41
42     public void stop() {
43         view.getPanel().setFocusable(false);
44         timer.stop();
45         model.stop();
46         view.stop();
47     }
48
49     public void pause() {
50         view.getPanel().setFocusable(false);
51         timer.stop();
52         model.pause();
53         model.rootGameObject._pause();
54         view.pause();
55     }
56
57     public void unpause(Intent intent) {
58         view.getPanel().setFocusable(true);
59         timer.restart();
60         prevTime = System.nanoTime();
61         model.unpause();
62         model.rootGameObject._unpause();
63         view.unpause();
64     }
65
66     public void actionPerformed(ActionEvent e) {
67         if (KeyboardFocusManager.getCurrentKeyboardFocusManager().getFocusOwner() != view.
68             getPanel()) {
69             // フォーカスが外れていたらリクエスト
70             KeyboardFocusManager.getCurrentKeyboardFocusManager().clearGlobalFocusOwner();
71             view.getPanel().requestFocus();
72         }
73         if (e.getSource() == timer) {
74             long time = System.nanoTime();
75             float deltaTime = (float) (time - prevTime) / 1000000000f;
76             prevTime = time;
77
78             model.calc(deltaTime);
79             model.rootGameObject._calc(deltaTime);
80             view.repaint();
81         }
81     }

```

```
82     protected IClickable findClickedObject(Vector2 position) {
83         Stack<GameObject> stack = new Stack<>();
84         IClickable clickedObject = null;
85         stack.push(model.rootGameObject);
86         int minDepth = Integer.MAX_VALUE;
87         while (!stack.empty()) {
88             var currentGameObject = stack.pop();
89             if (currentGameObject.collider != null && currentGameObject instanceof
90                 IClickable
91                 && currentGameObject.depth < minDepth && currentGameObject.collider.
92                     isPointIn(position)) {
93                 clickedObject = (IClickable) currentGameObject;
94                 minDepth = currentGameObject.depth;
95             }
96             if (currentGameObject.getEnabled()) {
97                 for (var nextGameObject : currentGameObject.children) {
98                     stack.push(nextGameObject);
99                 }
100            }
101        }
102        return clickedObject;
103    }
```

---

ソースコード 74: uectd.gameSystem.SceneModel

```
1 package uectd.gameSystem;
2
3 import uectd.gameSystem.util.*;
4 import java.util.Observable;
5
6 public abstract class SceneModel extends Observable {
7     protected SceneChangeListener sceneChangeListener;
8     protected GameObject rootGameObject;
9
10    public void start(Intent intent) {
11    }
12
13    public void stop() {
14        rootGameObject.destroy();
15    }
16
17    public void pause() {
18    }
19
20    public void unpause() {
21    }
22
23    public void calc(float deltaTime) {
24    }
25
26    public SceneModel(SceneChangeListener sceneChangeListener) {
27        rootGameObject = new GameObject(null, null);
28        rootGameObject.setEnabled(true);
29        this.sceneChangeListener = sceneChangeListener;
30    }
31
32    public void addGameObject(GameObject gameObject) {
33        rootGameObject.addChild(gameObject);
34    }
35
36 }
```

---

ソースコード 75: uectd.gameSystem.SceneView

```
1 package uectd.gameSystem;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.util.*;
6
7 import uectd.gameSystem.util.*;
```

```

8
9 public abstract class SceneView extends JPanel implements Observer {
10     protected SceneModel model;
11
12     public Camera camera; // カメラを表す
13     GameObject。描画にこのオブジェクトの情報を用いることもできるが、用いないこともできる。
14
15     public JPanel getPanel() {
16         return this;
17     }
18
19     public SceneView(SceneModel model) {
20         this.setBackground(Color.white);
21         this.model = model;
22         model.addObserver(this);
23     }
24
25     @Override
26     protected void paintComponent(Graphics g) {
27         super.paintComponent(g);
28         this.draw(g);
29     }
30
31     @Override
32     public void update(Observable o, Object arg) {
33         repaint();
34     }
35
36     public void start() {
37         camera = (Camera) model.rootGameObject.findChild(Camera.class);
38         if (camera == null) {
39             System.err.println("カメラがツリー内に存在しません");
40         }
41     }
42
43     public void stop() {
44     }
45
46     public void pause() {
47     }
48
49     public void unpause() {
50     }
51
52     public void draw(Graphics g) {
53         // DFSして描画できるオブジェクトだけまとめる
54         Stack<GameObject> stack = new Stack<>();
55         ArrayList<Drawable> drawableList = new ArrayList<>();
56
57         stack.push(model.rootGameObject);
58         while (!stack.empty()) {
59             var currentGameObject = stack.pop();
60             if (currentGameObject.getEnabled()) {
61                 if (currentGameObject instanceof Drawable)
62                     drawableList.add((Drawable) currentGameObject);
63                 for (var nextGameObject : currentGameObject.children) {
64                     if (nextGameObject.getEnabled())
65                         stack.push(nextGameObject);
66                 }
67             }
68         }
69
70         // Depthでソートして draw
71         drawableList.sort((a, b) -> b.depth - a.depth);
72
73         for (Drawable iDrawable : drawableList) {
74             if (camera == null) {
75                 iDrawable.draw(g, iDrawable.position, 1);
76             } else {
77                 iDrawable.draw(g,
78                             new Vector2(
79                                 (iDrawable.position.x + iDrawable.offset.x - camera.position.x
80                                     ) * camera.ratio
81                                     + Define.HALF_WINDOW_WIDTH,
82

```

```

81             (iDrawable.position.y + iDrawable.offset.y - camera.position.y
82                 ) * camera.ratio
83                 + Define.HALF_WINDOW_HEIGHT),
84             camera.ratio);
85     }
86 }
87
88 public Vector2 convertScreenPosToWorldPos(Vector2 screenPos) {
89     return new Vector2((screenPos.x - Define.HALF_WINDOW_WIDTH) / camera.ratio +
90         camera.position.x,
91         (screenPos.y - Define.HALF_WINDOW_HEIGHT) / camera.ratio + camera.position.
92             y);
93 }
94
95 public Vector2 convertWorldPosToScreenPos(Vector2 worldPos) {
96     return new Vector2((worldPos.x - camera.position.x) * camera.ratio + Define.
97         HALF_WINDOW_WIDTH,
98         (worldPos.y - camera.position.y) * camera.ratio + Define.HALF_WINDOW_HEIGHT
99     );
100 }
101
102 }

```

---

#### ソースコード 76: uectd.gameSystem.util.AnimationSprite

```

1 package uectd.gameSystem.util;
2
3 import uectd.gameSystem.GameObject;
4 import java.awt.event.*;
5 import java.awt.*;
6
7 // アニメーションするオブジェクト
8 public class AnimationSprite extends Sprite implements ActionListener {
9
10    private int timePerFrame;
11    private Image[] images;
12    private int[] imageIndexes;
13    private int currentImageIndex;
14    private int currentImagesNum;
15    private GameTimer timer;
16
17    @Override
18    public AnimationSprite clone() {
19        AnimationSprite animationSprite = null;
20
21        animationSprite = (AnimationSprite) super.clone();
22        animationSprite.images = this.images.clone();
23        animationSprite.imageIndexes = this.imageIndexes.clone();
24        animationSprite.timer = new GameTimer(this.timer.getDelay(), animationSprite);
25        animationSprite.timer.setInitialDelay(this.timer.getInitialDelay());
26        return animationSprite;
27    }
28
29    public AnimationSprite(GameObject root, GameObject parent) {
30        super(root, parent);
31    }
32
33    protected void setImages(Image[] images, int timePerFrame) {
34        this.images = images;
35        this.timePerFrame = timePerFrame;
36    }
37
38    protected void setImageIndexes(int[] imageIndexes) {
39        this.imageIndexes = imageIndexes;
40        this.currentImagesNum = imageIndexes.length;
41        this.currentImageIndex = 0;
42        this.setImage(images[imageIndexes[0]]);
43    }
44
45    protected void startAnimation() {
46        timer = new GameTimer(timePerFrame, this);
47        timer.start();
48    }

```

```
49     protected void stopAnimation() {
50         timer.stop();
51     }
52
53     @Override
54     public void actionPerformed(ActionEvent e) {
55         currentImageIndex++;
56         if (currentImageIndex >= currentImagesNum) {
57             currentImageIndex = 0;
58         }
59         setImage(images[imageIndexes[currentImageIndex]], false);
60     }
61
62     @Override
63     public void pause() {
64         super.pause();
65         timer.pause();
66     }
67
68     @Override
69     public void unpause() {
70         super.unpause();
71         timer.unpause();
72     }
73
74     @Override
75     public void onDestroyed() {
76         super.onDestroyed();
77         timer.stop();
78     }
79
80 }
81 }
```

---

#### ソースコード 77: uectd.gameSystem.util.Camera

```
1 package uectd.gameSystem.util;
2
3 import uectd.gameSystem.GameObject;
4
5 public class Camera extends GameObject {
6     public float ratio;
7
8     public Camera(GameObject root, GameObject parent, float ratio) {
9         super(root, parent);
10        this.ratio = ratio;
11    }
12
13 }
```

---

#### ソースコード 78: uectd.gameSystem.util.CircleCollider

```
1 package uectd.gameSystem.util;
2
3 import uectd.gameSystem.GameObject;
4
5 public class CircleCollider extends Collider {
6     public double radius;
7
8     public CircleCollider(GameObject gameObject, double radius) {
9         super(gameObject);
10        this.radius = radius;
11    }
12
13     // 座標がコライダーの中にあるか
14     @Override
15     public boolean isPointIn(Vector2 point) {
16         return ((gameObject.position.x + gameObject.offset.x - point.x)
17                 * (gameObject.position.x + gameObject.offset.x - point.x)
18                 + (gameObject.position.y + gameObject.offset.y - point.y)
19                 * (gameObject.position.y + gameObject.offset.y - point.y)) < radius *
20                     radius;
21     }
22 }
```

```

21 // コライダー同士の衝突判定
22 @Override
23 public boolean isColliderOn(Collider other) {
24     if (other instanceof CircleCollider) {
25         CircleCollider otherCirc = (CircleCollider) other;
26         return Vector2.diff(other.getPosition(), this.getPosition()).norm() < (this.
27             radius + otherCirc.radius)
28             * (this.radius + otherCirc.radius);
29     }
30     return false;
31 }
32
33 @Override
34 public CircleCollider clone() {
35     CircleCollider collider = null;
36     collider = (CircleCollider) super.clone();
37     return collider;
38 }
39 }
```

---

ソースコード 79: uectd.gameSystem.util.Collider

```

1 package uectd.gameSystem.util;
2
3 import java.util.ArrayList;
4
5 import uectd.gameSystem.FatalError;
6 import uectd.gameSystem.GameObject;
7
8 public abstract class Collider implements Cloneable {
9
10    public GameObject gameObject;
11
12    public Collider(GameObject gameObject) {
13        this.gameObject = gameObject;
14    }
15
16    public abstract boolean isPointIn(Vector2 point);
17
18    public abstract boolean isColliderOn(Collider collider);
19
20    public Vector2 getPosition() {
21        return new Vector2(gameObject.position.x + gameObject.offset.x, gameObject.
22            position.y + gameObject.offset.y);
23    }
24
25    public ArrayList<GameObject> findCollidedChildren(GameObject parent) {
26        var res = new ArrayList<GameObject>();
27        for (var gameObject : parent.children) {
28            if (gameObject.collider != null && this.isColliderOn(gameObject.collider)) {
29                res.add(gameObject);
30            }
31        }
32        return res;
33    }
34
35    @Override
36    public Collider clone() {
37        Collider collider = null;
38        try {
39            collider = (Collider) super.clone();
40        } catch (CloneNotSupportedException ce) {
41            ce.printStackTrace();
42            FatalError.quit("オブジェクトのクローンに失敗しました");
43        }
44        return collider;
45    }
46 }
```

---

ソースコード 80: uectd.gameSystem.util.Drawable

```
1 package uectd.gameSystem.util;
2
3 import java.awt.*;
4
5 import uectd.gameSystem.GameObject;
6
7 public abstract class Drawable extends GameObject {
8     public Drawable(GameObject root, GameObject parent) {
9         super(root, parent);
10    }
11
12    public abstract void draw(Graphics g, Vector2 screenPosition, float ratio);
13
14 }
```

ソースコード 81: uectd.gameSystem.util.Drawable

```
1 package uectd.gameSystem.util;
2
3 import uectd.gameSystem.GameObject;
4 import java.awt.*;
5
6 public class Effect extends Sprite {
7
8     protected Image[] images;
9     public boolean loopFlag;
10    public float frameTime, elapsedTime;
11    public int currentFrame;
12    private int frameNumber;
13
14    public Effect(GameObject root, GameObject parent, Image[] images, boolean loopFlag,
15                  float frameTime, int depth) {
16        super(root, parent);
17        this.loopFlag = loopFlag;
18        if (images != null) {
19            this.frameNumber = images.length;
20            if (images.length > 0)
21                this.images = new Image[images.length];
22            setImage(images[0]);
23        }
24        this.images = images;
25        this.depth = depth;
26        this.frameTime = frameTime;
27        this.elapsedTime = 0;
28        this.currentFrame = 0;
29    }
30
31    @Override
32    public void onEnabled() {
33        super.onEnabled();
34        if (images != null)
35            setImage(images[0]);
36
37        this.elapsedTime = 0;
38        this.currentFrame = 0;
39    }
40
41    @Override
42    public void calc(float deltaTime) {
43        elapsedTime += deltaTime;
44        if (elapsedTime >= frameTime) {
45            currentFrame++;
46            if (currentFrame >= frameNumber) {
47                if (loopFlag) {
48                    currentFrame = 0;
49                } else {
50                    this.destroy();
51                    return;
52                }
53            }
54            if (images != null)
55                setImage(images[currentFrame], false);
56            elapsedTime = 0;
57        }
58        super.calc(deltaTime);
59    }
60}
```

```

58     }
59
60     @Override
61     public void draw(Graphics g, Vector2 screenPosition, float ratio) {
62         super.draw(g, screenPosition, ratio);
63     }
64 }

```

---

ソースコード 82: uectd.gameSystem.util.GameSound

```

1 package uectd.gameSystem.util;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.net.MalformedURLException;
6
7 import javax.sound.sampled.AudioFormat;
8 import javax.sound.sampled.AudioInputStream;
9 import javax.sound.sampled.AudioSystem;
10 import javax.sound.sampled.Clip;
11 import javax.sound.sampled.DataLine;
12 import javax.sound.sampled.FloatControl;
13 import javax.sound.sampled.LineUnavailableException;
14 import javax.sound.sampled.UnsupportedAudioFileException;
15
16 public class GameSound {
17     private String filePath;
18     private Clip clip;
19
20     public GameSound(String filePath) {
21         this.filePath = filePath;
22     }
23
24     public void init() {
25         clip = createClip(new File(filePath));
26     }
27
28     private static Clip createClip(File path) {
29         // 指定されたURLのオーディオ入力ストリームを取得
30         try (AudioInputStream ais = AudioSystem.getAudioInputStream(path)) {
31
32             // ファイルの形式取得
33             AudioFormat af = ais.getFormat();
34
35             // 単一のオーディオ形式を含む指定した情報からデータラインの情報オブジェクトを構築
36             DataLine.Info dataLine = new DataLine.Info(Clip.class, af);
37
38             // 指定された Line.Info オブジェクトの記述に一致するラインを取得
39             Clip c = (Clip) AudioSystem.getLine(dataLine);
40
41             // 再生準備完了
42             c.open(ais);
43
44             return c;
45         } catch (MalformedURLException e) {
46             e.printStackTrace();
47         } catch (UnsupportedAudioFileException e) {
48             e.printStackTrace();
49         } catch (IOException e) {
50             e.printStackTrace();
51         } catch (LineUnavailableException e) {
52             e.printStackTrace();
53         }
54         return null;
55     }
56
57     // 再生開始
58     public void start() {
59         clip.setFramePosition(0);
60         clip.start();
61     }
62 }

```

```

63 // ループ再生
64 public void loop() {
65     clip.loop(Clip.LOOP_CONTINUOUSLY);
66 }
67
68 // 一時停止
69 public void pause() {
70     clip.stop();
71 }
72
73 // 停止
74 public void stop() {
75     clip.stop();
76     clip.close();
77 }
78
79 public boolean isPlaying() {
80     return clip.isRunning();
81 }
82
83 // 音量調整
84 public void volume(int soundVolume) {
85     FloatControl ctrl = (FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);
86     ctrl.setValue((float) Math.log10((float) soundVolume / 100) * 20);
87 }
88 }

```

---

ソースコード 83: uectd.gameSystem.util.GameTimer

```

1 package uectd.gameSystem.util;
2
3 import javax.swing.Timer;
4 import java.awt.event.*;
5
6 public class GameTimer extends Timer implements ActionListener {
7     private long offsetTime; // ポーズ時に次回の初期遅延を計算するためのオフセット時間
8     private int initialDelay;
9     private boolean isRunning;
10
11    public GameTimer(int delay, ActionListener listener) {
12        super(delay, listener);
13        this.addActionListener(this);
14        this.initialDelay = delay;
15    }
16
17    public void start() {
18        this.offsetTime = System.currentTimeMillis();
19        this.initialDelay = getDelay();
20        this.isRunning = true;
21        super.start();
22    }
23
24    public void stop() {
25        this.isRunning = false;
26        super.stop();
27    }
28
29    public void pause() {
30        long nowTime = System.currentTimeMillis();
31        initialDelay -= nowTime - offsetTime;
32        super.stop();
33        super.setInitialDelay(Math.max(0, initialDelay));
34        this.offsetTime = nowTime;
35    }
36
37    public void unpause() {
38        this.offsetTime = System.currentTimeMillis();
39        if (isRunning)
40            super.start();
41    }
42
43    @Override
44    public void actionPerformed(ActionEvent e) {
45        this.offsetTime = System.currentTimeMillis();

```

```
46     this.initialDelay = getDelay();
47 }
48
49     @Override
50     public void setInitialDelay(int initialDelay) {
51         super.setInitialDelay(initialDelay);
52         this.initialDelay = initialDelay;
53     }
54
55     @Override
56     public int getInitialDelay() {
57         return super.getInitialDelay();
58     }
59 }
```

---

ソースコード 84: uectd.gameSystem.util.IClickable

```
1 package uectd.gameSystem.util;
2
3 // クリック可能なオブジェクトのマーカーインターフェース
4 public interface IClickable {
5 }
```

---

ソースコード 85: uectd.gameSystem.util.ImageManager

```
1 package uectd.gameSystem.util;
2
3 import java.awt.*;
4 import java.awt.image.BufferedImage;
5 import java.io.File;
6 import java.io.IOException;
7 import java.nio.file.*;
8 import java.util.HashMap;
9 import java.util.Map;
10
11 import javax.imageio.ImageIO;
12
13 import uectd.gameSystem.FatalError;
14
15 public class ImageManager {
16     // Singletonパターンを使用
17     private static ImageManager instance;
18     // パス -> 適当なユニークの番号と
19     // 番号 -> 画像の対応関係がある
20     // エフェクトなど 1つの画像に複数の画像をまとめることがよくあるのでパスと画像の対応は 1対 1で
21     // ない
22     private Map<Integer, Image> images;
23     private Map<String, Integer> ids;
24
25     // 番号との対応付けのためのカウンタ
26     private static int counter = 0;
27
28     private ImageManager() {
29         images = new HashMap<>();
30         ids = new HashMap<>();
31     }
32
33     public static ImageManager getInstance() {
34         if (instance == null) {
35             instance = new ImageManager();
36         }
37         return instance;
38     }
39
40     private int _loadImage(String absolutePath) throws IOException {
41         BufferedImage image = ImageIO.read(new File(absolutePath));
42         int id = counter++;
43         ids.put(absolutePath, id);
44         images.put(id, image);
45         return id;
46     }
}
```

```

47     public int loadImage(String filePath) {
48         try {
49             String absolutePath = Paths.get(filePath).toRealPath(LinkOption.NOFOLLOW_LINKS
50                                         ).toString();
51             return _loadImage(absolutePath);
52         } catch (IOException e) {
53             e.printStackTrace();
54             FatalError.quit("画像ファイル入出力エラー");
55             return -1;
56         }
57     }
58     // 一つの画像ファイルから複数の画像を切り出したものを取得
59     public Image[] getDivImage(String filePath, int xNum, int yNum, int xSize, int ySize)
60     {
61         Image[] res = new Image[xNum * yNum];
62         try {
63             String absolutePath = Paths.get(filePath).toRealPath(LinkOption.NOFOLLOW_LINKS
64                                         ).toString();
65             if (ids.containsKey(absolutePath)) {
66                 int id = ids.get(absolutePath);
67                 for (int i = 0; i < res.length; i++) {
68                     res[i] = images.get(id + i);
69                 }
70             } else {
71                 ids.put(absolutePath, counter);
72                 BufferedImage image = ImageIO.read(new File(absolutePath));
73                 for (int y = 0; y < yNum; y++) {
74                     for (int x = 0; x < xNum; x++) {
75                         images.put(counter, image.getSubimage(xSize * x, ySize * y, xSize,
76                                                 ySize));
77                         res[y * xNum + x] = images.get(counter++);
78                     }
79                 }
80             }
81             return res;
82         } catch (IOException e) {
83             e.printStackTrace();
84             FatalError.quit("画像ファイル入出力エラー");
85             return null;
86         }
87     }
88     public Image getImage(String filePath) {
89         try {
90             String absolutePath = Paths.get(filePath).toRealPath(LinkOption.NOFOLLOW_LINKS
91                                         ).toString();
92             if (!ids.containsKey(absolutePath)) {
93                 _loadImage(absolutePath);
94             }
95             return images.get(ids.get(absolutePath));
96         } catch (IOException e) {
97             e.printStackTrace();
98             FatalError.quit("画像ファイル入出力エラー");
99             return null;
100        }
101    }
102    public Image getImage(int id) {
103        return images.get(id);
104    }
105 }
```

---

ソースコード 86: uectd.gameSystem.util.Intent

---

```

1 package uectd.gameSystem.util;
2
3 import java.util.HashMap;
4
5 public class Intent {
6     private HashMap<String, Integer> integerMap;
7
8     private HashMap<String, Boolean> booleanMap;
9 }
```

```
10  public Intent() {
11      integerMap = new HashMap<>();
12      booleanMap = new HashMap<>();
13  }
14
15  public void setIntegerValue(String key, int value) {
16      integerMap.put(key, value);
17  }
18
19  public int getIntegerValue(String key) {
20      return integerMap.get(key);
21  }
22
23  public void setBooleanValue(String key, boolean value) {
24      booleanMap.put(key, value);
25  }
26
27  public boolean getBooleanValue(String key) {
28      return booleanMap.get(key);
29  }
30
31  public boolean containsIntegerKey(String key) {
32      return integerMap.containsKey(key);
33  }
34
35  public boolean containsBooleanKey(String key) {
36      return booleanMap.containsKey(key);
37  }
38 }
```

---

ソースコード 87: uectd.gameSystem.util.ObservableComponent

```
1 package uectd.gameSystem.util;
2
3 import java.util.Observable;
4
5 public class ObservableComponent<T> extends Observable {
6     private T value;
7
8     public ObservableComponent(T value) {
9         setValue(value);
10    }
11
12    public ObservableComponent() {
13    }
14
15    public T getValue() {
16        return value;
17    }
18
19    public void setValue(T value) {
20        this.value = value;
21        setChanged();
22        notifyObservers();
23    }
24
25    public void update() {
26        setChanged();
27        notifyObservers();
28    }
29 }
```

---

ソースコード 88: uectd.gameSystem.util.SoundManager

```
1 package uectd.gameSystem.util;
2
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.Map;
6
7 import uectd.gameSystem.FatalError;
8
9 import java.io.IOException;
```

```

10 import java.nio.file.LinkOption;
11 import java.nio.file.Paths;
12
13 public class SoundManager {
14
15     private static SoundManager instance;
16     private Map<String, ArrayList<GameSound>> soundsMap;
17     private int volume = 5;
18
19     private SoundManager() {
20         soundsMap = new HashMap<>();
21     }
22
23     public static SoundManager getInstance() {
24         if (instance == null) {
25             instance = new SoundManager();
26         }
27         return instance;
28     }
29
30     public void setVolume(int volume) {
31         this.volume = volume;
32     }
33
34     public void play(String filePath) {
35         try {
36             String fullPath = Paths.get(filePath).toRealPath(LinkOption.NOFOLLOW_LINKS).
37                 toString();
38             if (soundsMap.containsKey(fullPath)) {
39                 var sounds = soundsMap.get(Paths.get(filePath).toRealPath(LinkOption.
40                     NOFOLLOW_LINKS).toString());
41                 for (var sound : sounds) {
42                     if (!sound.isPlaying()) {
43                         sound.start();
44                         return;
45                     }
46                     sounds.add(new GameSound(fullPath));
47                     sounds.get(sounds.size() - 1).init();
48                     sounds.get(sounds.size() - 1).volume(volume);
49                     sounds.get(sounds.size() - 1).start();
50                 } else {
51                     var sounds = new ArrayList<GameSound>();
52                     var sound = new GameSound(fullPath);
53                     sounds.add(sound);
54                     soundsMap.put(fullPath, sounds);
55                     sound.init();
56                     sound.volume(volume);
57                     sound.start();
58                 }
59             } catch (IOException e) {
60                 e.printStackTrace();
61                 FatalError.quit("音声ファイル入出力エラー");
62             }
63         }
64         public void allStop() {
65             for (var sounds : soundsMap.values()) {
66                 for (var sound : sounds) {
67                     if (sound.isPlaying()) {
68                         sound.stop();
69                     }
70                 }
71             }
72         }
73     }
74 }

```

---

#### ソースコード 89: uectd.gameSystem.util.Sprite

```

1 package uectd.gameSystem.util;
2
3 import uectd.gameSystem.Define;
4 import uectd.gameSystem.GameObject;

```

```

5 import java.awt.*;
6
7 public class Sprite extends Drawable {
8
9     public Image image;
10
11    protected int width, height; // 画像サイズ
12
13    private Vector2 drawSize; // 描画サイズ(ゲームオブジェクト空間基準)
14
15    @Override
16    public Sprite clone() {
17        Sprite sprite = null;
18
19        sprite = (Sprite) super.clone();
20        sprite.drawSize = drawSize.clone();
21        return sprite;
22    }
23
24
25    public Sprite(GameObject root, GameObject parent) {
26        super(root, parent);
27        this.drawSize = new Vector2();
28    }
29
30    protected void setImage(String filePath, boolean setDrawSize) {
31        this.image = ImageManager.getInstance().getImage(filePath);
32        this.width = image.getWidth(null);
33        this.height = image.getHeight(null);
34        if (setDrawSize) {
35            this.drawSize.x = this.width;
36            this.drawSize.y = this.height;
37        }
38    }
39
40    protected void setImage(int id, boolean setDrawSize) {
41        this.image = ImageManager.getInstance().getImage(id);
42        this.width = image.getWidth(null);
43        this.height = image.getHeight(null);
44        if (setDrawSize) {
45            this.drawSize.x = this.width;
46            this.drawSize.y = this.height;
47        }
48    }
49
50    protected void setImage(Image image, boolean setDrawSize) {
51        this.image = image;
52        if (image != null) {
53            this.width = image.getWidth(null);
54            this.height = image.getHeight(null);
55        }
56        if (setDrawSize) {
57            this.drawSize.x = this.width;
58            this.drawSize.y = this.height;
59        }
60    }
61
62    protected void setImage(String filePath) {
63        this.setImage(filePath, true);
64    }
65
66    protected void setImage(int id) {
67        this.setImage(id, true);
68    }
69
70    protected void setImage(Image image) {
71        this.setImage(image, true);
72    }
73
74    protected void setDrawSize(Vector2 drawSize) {
75        this.drawSize = drawSize;
76    }
77
78    @Override
79    public void draw(Graphics g, Vector2 screenPosition, float ratio) {

```

```

80     if (image != null) {
81         float half_width = (float) drawSize.x * 0.5f * ratio;
82         float half_height = (float) drawSize.y * 0.5f * ratio;
83
84         int x = (int) (screenPosition.x - half_width);
85         int y = (int) (screenPosition.y - half_height);
86         int w = (int) (drawSize.x * ratio);
87         int h = (int) (drawSize.y * ratio);
88         if (x + w >= 0 && y + h >= 0 && x <= Define.WINDOW_WIDTH && y <= Define.
89             WINDOW_HEIGHT) {
90             g.drawImage(image, x, y, w, h, null);
91         }
92     }
93 }
94 }
```

---

### ソースコード 90: uectd.gameSystem.util.Vector2

```

1 package uectd.gameSystem.util;
2
3 import uectd.gameSystem.FatalError;
4
5 public class Vector2 implements Cloneable {
6     private static final double EPS = 1e-9;
7
8     public static final int COUNTER_CLOCKWISE = 1;
9     public static final int CLOCKWISE = -1;
10    public static final int ONLINE_BACK = 2;
11    public static final int ONLINE_FRONT = -2;
12    public static final int ON_SEGMENT = 0;
13
14    public double x, y;
15
16    public Vector2(double x, double y) {
17        this.x = x;
18        this.y = y;
19    }
20
21    public Vector2() {
22        this(0., 0.);
23    }
24
25    public String toString() {
26        return String.format("[%f, %f]", x, y);
27    }
28
29    public void add(Vector2 vector) {
30        this.x += vector.x;
31        this.y += vector.y;
32    }
33
34    public static Vector2 sum(Vector2 vector1, Vector2 vector2) {
35        return new Vector2(vector1.x + vector2.x, vector1.y + vector2.y);
36    }
37
38    public void sub(Vector2 vector) {
39        this.x -= vector.x;
40        this.y -= vector.y;
41    }
42
43    public static Vector2 diff(Vector2 a, Vector2 b) {
44        return new Vector2(a.x - b.x, a.y - b.y);
45    }
46
47    public double norm() {
48        return x * x + y * y;
49    }
50
51    public double magnitude() {
52        return Math.sqrt(x * x + y * y);
53    }
54
55    public Vector2 normalized() { // 正規化
```

```

56     double m = this.magnitude();
57     if (Math.abs(m) < EPS) {
58         return new Vector2();
59     }
60     return new Vector2(this.x / m, this.y / m);
61 }
62
63 public static Vector2 scale(Vector2 v, double s) {
64     return new Vector2(v.x * s, v.y * s);
65 }
66
67 public static double dot(Vector2 a, Vector2 b) {
68     return a.x * b.x + a.y * b.y;
69 }
70
71 public static double cross(Vector2 a, Vector2 b) {
72     return a.x * b.y - a.y * b.x;
73 }
74
75 public static int ccw(Vector2 p0, Vector2 p1, Vector2 p2) {
76     Vector2 v1 = new Vector2(p1.x - p0.x, p1.y - p0.y);
77     Vector2 v2 = new Vector2(p2.x - p0.x, p2.y - p0.y);
78     double c = cross(v1, v2);
79     if (c > EPS)
80         return COUNTER_CLOCKWISE;
81     if (c < -EPS)
82         return CLOCKWISE;
83     if (dot(v1, v2) < -EPS)
84         return ONLINE_BACK;
85     if (v1.norm() < v2.norm())
86         return ONLINE_FRONT;
87     return ON_SEGMENT;
88 }
89
90 @Override
91 public Vector2 clone() {
92     Vector2 v = null;
93     try {
94         v = (Vector2) super.clone();
95     } catch (CloneNotSupportedException ce) {
96         ce.printStackTrace();
97         FatalError.quit("オブジェクトのクローンに失敗しました");
98     }
99     return v;
100 }
101 }
```

---