2020 － 後学期

メディア情報学
プログラミング演習
グループプログラミングレポート

横スクロールアクションゲーム

グループ番号	17
1910343	新谷大樹
1910361	妹尾拓武
1910601	前田裕作

1. プログラムの概要

作成した横スクロールアクションゲームはキャラを移動させてスコアを獲得しながら，Boss を倒す又はゴール地点にたどり着くことを目指すゲームである．スコアは敵を倒す又はコインを取得することで獲得することができる．敵を倒す手段は火の玉を出して攻撃する又は敵をジャンプで踏みつける(Boss には無効)の２つである．敵にぶつかる又は敵の攻撃を受けることで HP が１つ減少する．HP が０になる又は崖から落ちることでゲームオーバーとなる．

グループワークの第 1 週目は，どのようなゲームを作成するかを話し合い，第 2 週目にマリオのような横スクロールアクションゲームを作成することに決定し，MVC モデルで新谷が Model，妹尾が View，前田が Controller を担当することにした．当初は横スクロールアクションゲームの基礎となるマップ，キャラクター(これら二つは最初四角形で表現)，当たり判定，キャラ移動(ジャンプを含む)を実装することができた．そこから新谷は画像読み込みと scv ファイルでマップを作成した．この時点で前田は Controller の作成を終えていたため，敵である Enemy クラスを複数作成することになった．妹尾はタイトル画面の作成を担当した．１月中旬ごろに，BGM や Boss，攻撃の Ball などを実装目的とし，グループ発表の週までに全ての機能を実装することができた．

Googledrive や github を利用してコードの共有・統合を行った．Github の管理を新谷が担当した．

図 1.1 役割分担

文責：妹尾

2. 設計方針

2.1. MVC モデル

複数人で共同開発するにあたり、MVC モデルを用いて開発を行った。MVC モデルはシステムを大きく分けて、データを扱う Model、表示を扱う View、ユーザからの入力を処理する Controller の 3 つの部分に分けて構築する。複数人で開発を行う場合の役割を分担す

る上で MVC によって分けることで分担が明確になるため適していると考えた。

2.2. ゲームの解析

MVC を用いて本ゲームを作るにあたり必要な機能を以下に挙げる。

- Model：キャラクターやフィールドといったデータの管理、処理
- View：画面への描画処理
- Controller：キーボードからの入力処理
- Character：Model で管理するプレイヤーや敵などのオブジェクト
- Field：キャラクターが動くフィールドデータの管理
- SceneManager：タイトル、ゲーム、クリア、ゲームオーバーといったシーン遷移の管理

2.3. 初期設計

上記の解析を基に全体像を把握するために簡易なクラス図を作った。これを基に開発を行った。

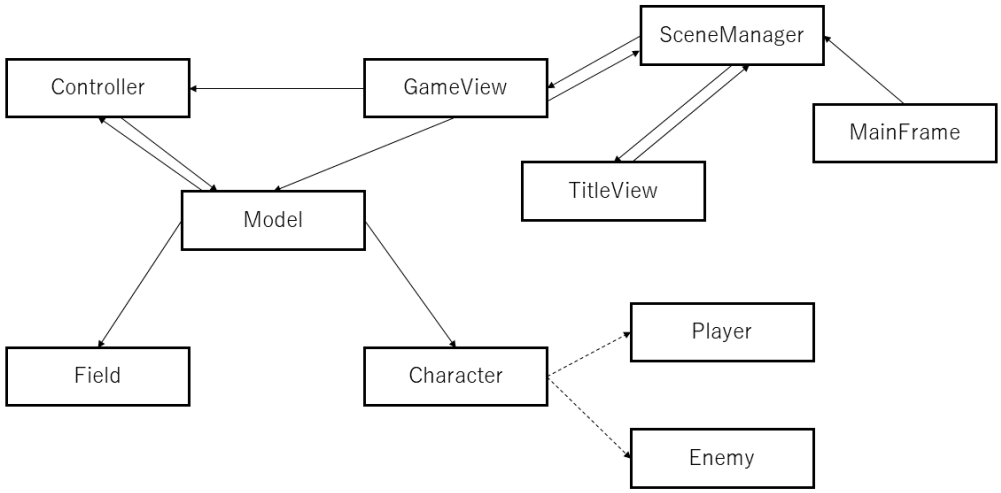


図 2.1. 初期簡易版クラス図

2.4. 共同開発の方針

コードの共有には Googledrive と github を用いた。Googledrive は各自のコードを共有するために用い、github は現状の最新バージョン、動くものができてからは動作が確認されているものの共有に用いた。これによって、各自のコードの共有と最新バージョンの共有を行えるようにした。また、各クラスでカプセル化を試みた。各クラスのフィールドは基本的に private な変数として定義し、異なるクラスからの値の呼び出し、書き換えは public な get 関数、set 関数を用いて行うことで、それぞれのオブジェクトの独立性を高め、予期し

ない値の書き換えを防ぐことができる。

2.5. 開発の順序

初期の目標としてゲームとしてとりあえず動くものを作ることを目標にしていた。そのため、開発の順序はゲームを動かすために最小限必要な Model、GameView、Controller、Field、Character、Player、MainFrame クラスの実装を行った。その後、シーン遷移に必要な SceneManager クラス、敵クラスと機能を拡張していくように実装した。このような順序で設計を行うことで、常にゲームが動く状態で開発を行えるようになった。

2.6. 最終設計

開発を行いながら設計を見直し、最終的なクラス図は以下のようになった。

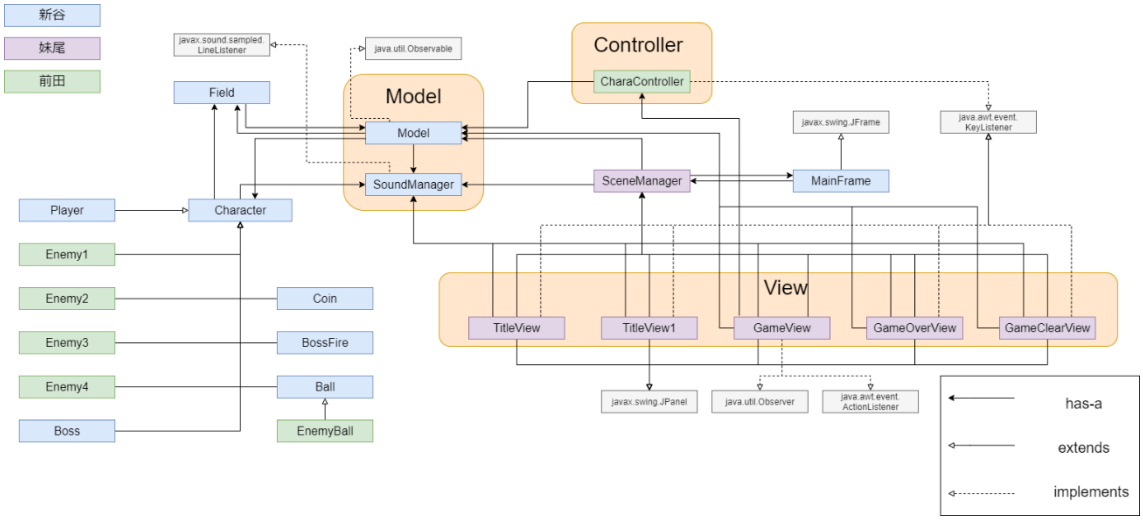
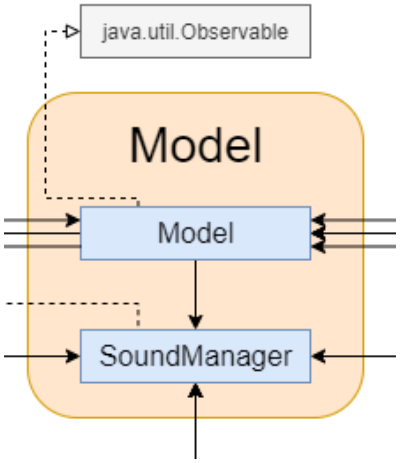


図2.2. クラス図

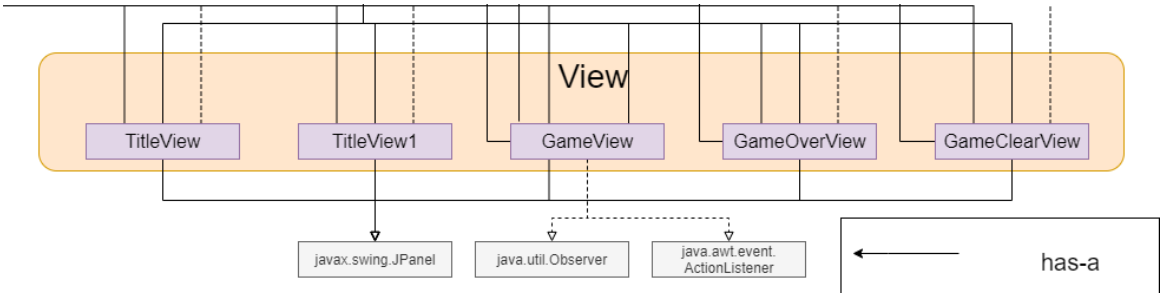
ゲームの解析で必要な機能として挙げた Model、View、Controller、Character、Field、SceneManager の最終的な設計についてそれぞれ説明する。

2.6.1. Model



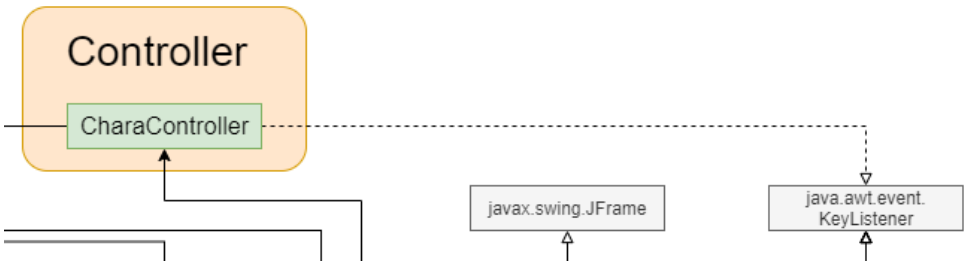
Model は Model クラスと SoundManager クラスから成立している。Model クラスでは Character クラス同士の衝突判定、SoundManager クラスと Character クラスの結び付け、Character クラスの管理といったゲーム中に動的に変化する Character クラスの管理を行っている。また、ゲーム中のスコア、シーン遷移のフラグといった内部パラメータの管理も行っている。Character クラス同士、Character クラスと Field クラス、Character クラスと SoundManager クラスのように異なるオブジェクト同士を繋げる役割を果たしている。SoundManager クラスは音楽データに関する記録を専門的請け負っている。すべての音楽再生、管理の責任を SoundManager クラスで持つことでエラーが発生したときの場所の特定が簡単になる。

2.6.2. View



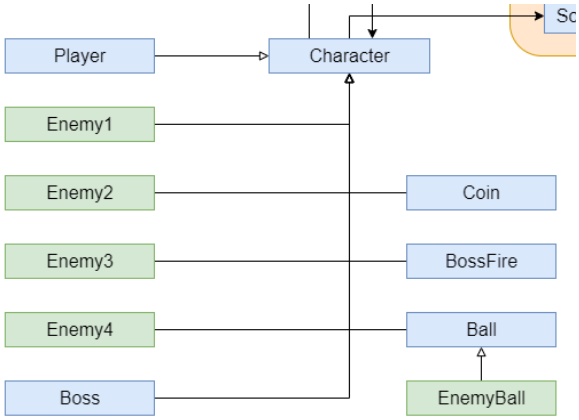
View は TitleView、TitleView1、GameView、GameOverView、GameClearView クラスから成立している。それぞれ JPanel を継承しておりフレームにパネルとして設置することでシーンの役割を果たしている。

2.6.3. Controller



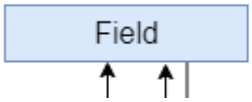
Controller は CharaController クラスがある。CharaController クラスではユーザが操作するプレイヤーへの入力を処理を行い、Model に伝えている。キーボードからの入力を受け付けるので KeyListner インタフェースをインプリメントしている。なお、ゲームシーン以外のタイトルやゲームオーバシーンでもキーボード入力の受付を行っているが、Model とやり取りをするような複雑な処理を行っていないため、View にまとめた。

2.6.4. Character



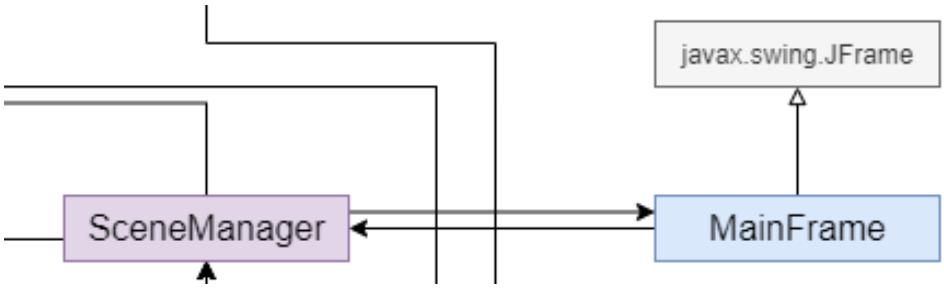
Character はゲーム中に現れるすべてのオブジェクトを指している。Character クラスやそれを継承した Player、Enemy1~4、Boss、Ball、BossFire、EnemyBall、Coin クラスが該当する。Character クラスには座標、速度、体力などの基本的な情報、Field を考慮した移動、ダメージや落下中の重力処理といった最低限の機能が組み込まれいる。このCharacter クラスは抽象的なクラスとして捉え、実際にインスタンス化するのはそれを継承した子クラスとなっている。Character クラスを継承した子クラスは最低限の機能はついているので、それをオーバーライドすることで簡単に独自の機能や描画処理の拡張を行うことができる。また、オブジェクト指向のポリモーフィズムの考え方により、Model クラスで Character クラスの子クラスのオブジェクトを管理するとき、Player、Enemy、Boss、Coin といった型に関係なく、Character クラスで一括して管理することができる。オーバーライドしたものは子クラスの関数が優先して用いられるので、Character クラスで管理しても問題はない。

2.6.5. Field



Field は Field クラスで定義している。Field クラスではフィールドデータの管理、Character を初期位置に設置、スクロール処理用の計算を行っている。フィールドデータは scv ファイルで管理できるようにしたことで、フィールド作成や管理の自由度が上がった。スクロール処理の詳細はプログラムの説明で行う。

2.6.6. SceneManager



SceneManager は SceneManager クラスと MainFrame クラスで構成している。SceneManager クラスではシーンの切り替え、切り替え前の初期化処理を行っている。MainFrame クラスは JFrame を継承しているので、SceneManager からの命令でパネルの切り替えを行うことでシーン遷移を実装している。

2.7. シーン遷移の設計

SceneManager が行うシーン遷移の流れを次の図で示す。

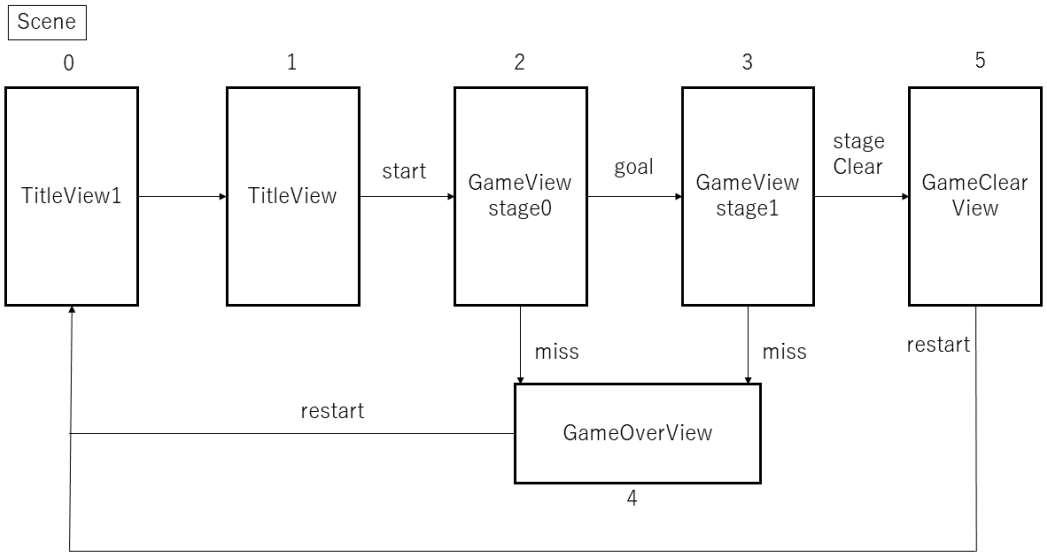


図2.2. シーン遷移の流れ

まず TitleView1 から始まり、操作説明を行う TitleView へ遷移する。その後１つ目のステージの GameView へ遷移し、途中でミスすれば gameOverView へ、ゴールできれば、２つ目のステージの GameView へと遷移する。ミスすれば GameOverView で、ステージをクリアできれば GameClearView へと遷移し、その後 TitleView1 に戻る。この遷移の間で適切に Model や Field の初期化処理を行うことでシーン遷移を実装する。

文責：新谷

3. プログラムの説明

3.1. Model クラス

このクラスでは Character の update 処理、Character 同士のあたり判定と処理、その他クラスの繋ぎ、ゲームの終了判定などを行っている。

3.1.1. 主なフィールド

Character クラスを管理する用の ArrayList 型である chara があります。ArrayList を用いることで動的に配列の長さを変えることができるので、キャラの消去や消去を頻繁に行うこのゲームに適していると考えた。さらに Character 型のリストにしておくことで、ポリモーフィズムにより Character クラスを継承した子クラスまでまとめて扱うことができる。また、pressedKeyRight、pressedKeyLeft はキーボードからの入力をモデル側で保存しておくことでキーボードの同時入力を可能にした。sounNames は soundManager に読み込みたいファイルの名前を指定している。これは静的で変更できないため文字の書き変わりを防いでいる。goal、gameOver、bossFlag、stageClear はシーン遷移のためのフラグとして用いている。その他、Field や SoundManager、Player のインスタンスを保持するための変数を定義している。

3.1.2. init 関数

init 関数はフィールドの変数の初期化やフィールドクラスのインスタンス作成などの初期化処理を行っている。これはコンストラクタでも呼ぶ関数ですが、引数でステージ番号を受け取ることでスコアの初期化などステージ遷移で初期化したくない変数の初期化は行わないようにした。

3.1.3. createPlayer 関数

createPlayer 関数は設置したい x, y 座標を受け取り Player のインスタンスを生成する。このとき player には音関連の処理を行う soundManager クラスを持たせる。さらに、キャラのリストの０番目にプレイヤーを設置する。これによってプレイヤーの番号が固定されるので、キャラのリストの中からプレイヤーのみに特別な処理をしたいときに便利になった。これは Field クラスでキャラを配置するときに用いる。

3.1.4. update 関数

update は GameView の Timer で一定時間ごとに呼び出され更新されている。ここではキーボードの入力からプレイヤーを移動させたり、キャラリストの落下と移動処理、消滅判定、攻撃フラグのチェック、キャラ同士の衝突判定の関数を呼び出している。またプレイヤー以外が画面外で不用意に移動するのを防ぐためにプレイヤーとの距離が一定以内でないとキャラクタークラスの update を実行しないようにしている。ここでキャラリストを for 文で繰り返すとき for(Character f:chara)とすればいいが、ここではキャラの消滅判定を行っているため for(int i=0;i<cgara.size();i++)としなければリストから外れているのにも関わらずアクセスしてしまいエラーになってしまう。

3.1.5. move 関数

move 関数は CharaController クラスで呼ばれており、キーボードの入力からどの向きに移動したいか、押しているかをモデルに伝える役割がある。

3.1.6. jump 関数

Jump 関数も CharaController クラスで呼び出され、呼び出されるとプレイヤークラスのジャンプ関数を呼び出し、ジャンプを行うようにした。

3.1.7. shoot 関数

shoot 関数も CharacterController クラスで呼び出され、soundManager から発射音を再生し、向きに応じてプレイヤーから発射されるように見える位置に Ball のインスタンスを設置し、速度を与えている。ここで作った Ball はキャラリストに入れて管理します。

3.1.8. enemyAttackFlagCheck 関数

enemyAttackFlagCheck 関数では敵キャラクタークラスを受け取り、そのキャラクター番号と attackFlag が true になっていることを確認し、キャラクター番号に応じた攻撃を行う。ここでの攻撃は攻撃用に新しくキャラリストに加える必要があるものを指している。特に Boss の攻撃ではプレイヤーと自分の位置から攻撃を発射する速度の単位ベクトルを計算するようにした。

3.1.9. endCheck 関数

endCheck 関数では hp が 0 になったキャラをキャラリストから消去することを行っている。また消去と同時にキャラクター番号に応じて score を加えている。ボスを倒したときのクリアフラグをここで行っている。

3.1.10. collisionCheck 関数

collisionCheck 関数ではキャラ同士の衝突判定を行っている。絶対値を利用した矩形同士の判定を x 軸方向、y 軸方向について行い、それがプレイヤーとのあたり判定かどうか

で場合分けをしている。矩形同士のあたり判定は次の図のようにして判定する。

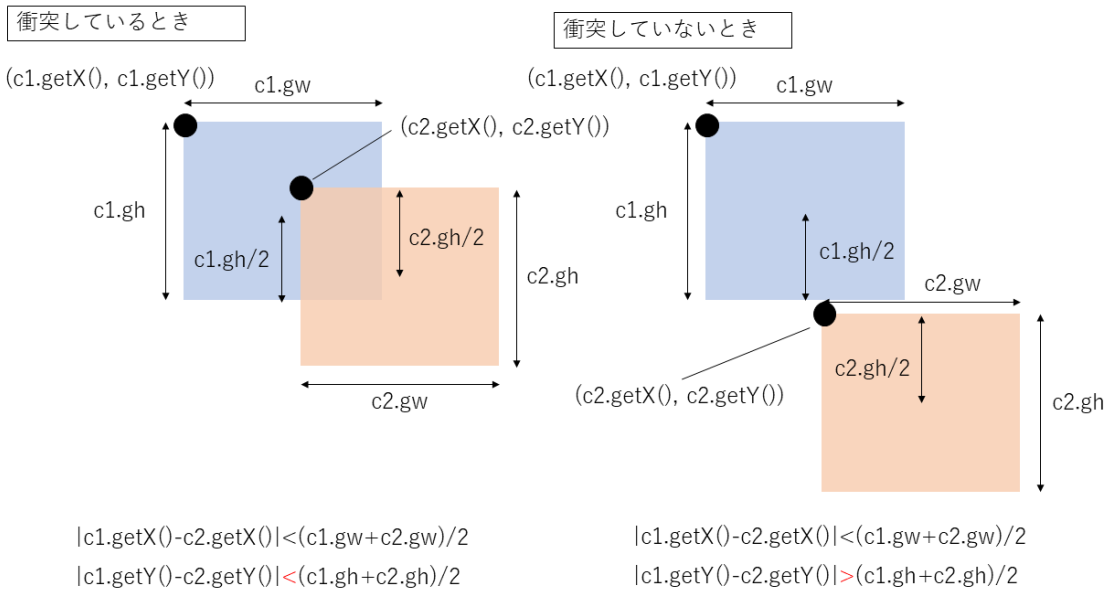


図3.1. 矩形同士のあたり判定

プレイヤーが出す攻撃用の球とのあたり判定を無視したり、coin と衝突したときに音を鳴らす、プレイヤーが上からあたるときは敵にダメージを与えて、上方向に速度を与える、それ以外はプレイヤーにダメージを与えるといった処理になっている。プレイヤー以外の衝突判定は特別なもの同士のあたり判定を除いて、両方がダメージを受けるようにしている。矩形同士のあたり判定のアルゴリズムは https://qiita.com/hp0me/items/57f901e9b0babe1a320e を参考にした。

3.1.11. gameOverCheck 関数

gameOverCheck 関数はプレイヤーの hp や座標が規定の値を超えたとき、例えば穴に落ちたときに GameOverView に遷移するフラグ gameOver を立てる。

3.1.12. loadSound 関数

loadSound 関数は init 関数内で呼ばれ、まだモデルは SounManager クラスの音データの読み込みを行っていない場合、soundNames の配列を使って順番に音データを読み込む SoundManager クラスの関数を呼び出す処理を行っている。

3.1.13. その他関数

他クラスで呼び出す用の set 関数、get 関数を定義している。

3.2. SoundManager クラス

SoundManager クラスは音ファイルの管理、再生、停止を行っている。このクラスを作成

するにあたり https://aidiary.hatenablog.com/entry/20061105/1275137770 を参考にした。そのまま用いたフィールド、関数には※をつけている。

3.2.1. 主なフィールド※

フィールドは clip を最大値を定める maxClips、作成したオーディオクリップを管理する clipMap、clip 数を数える counter から成っている。clipMap での clip の管理では HashMap を用いている。これによってキーとなる文字とそれに対する値として clip というように管理できるため、配列のインデックスで管理するより直感的に管理できる。またここでの管理は主に再生のためにキーから clip を探すという作業であるため、キーの探索が速い Hash での管理は適していると考えた。コンストラクタは引数が指定されないときは maxClips を 256 にする。指定された場合はそれを maxClips にし、HashMap 型の clipMap を作る。

3.2.2. load 関数※

load 関数は管理する用の名前とファイル名を受け取り、音ファイルの読み込みを行う。読み込んだファイルを再生できるクリップにし、clipMap に保存する。

3.2.3. play 関数※

play 関数は呼び出したいキーの名前を引数に受け取り、clipMap から clip を探し、あればそれを再生することを行っている。

3.2.4. loop 関数

loop 関数では play 関数と同じようにキーの名前を受け取り、clip を探索する。見つかれば、それをループ再生する。

3.2.5. stop 関数

stop 関数ではキーとなる文字を受け取り、それが clipMap にあれば、再生を止めて最初に戻す処理をしている。これは SE や効果音を任意のタイミングで止めるために自作した。

3.2.6. update 関数※

update 関数は SoundManager クラスが LineListener をインプリメントしているので作成している。LineEvent で再生中の clip のデータを管理でき、再生が止まると、次の再生に備えて最初の再生位置に戻すことを行っているが、このゲームではこの機能は用いていない。

3.3. Field クラス

Field クラスはフィールドデータの読み込み、フィールドデータに基づいた Character の設置、フィールドと Character のあたり判定時の処理、スクロール処理を行っている。

3.3.1. 主なフィールド

map はフィールドの情報を格納しておく二次元配列で、-1 のときは何もなし、0 はプレイヤーの初期位置、1 以降は数字に対応したブロック、-2 以下はそれに対応したキャラの初期位置が入っている。cs は一ブロックあたりの大きさセルサイズを示している。ROW、COL は二次元配列 map の列数と行数を示している。WIDTH、HEIGHT はマップの幅、高さ、IMAGESIZE は読み込む画像の一辺の長さを定義する。offsetX、offsetY はスクロール処理をするためのオフセットを格納する。

3.3.2. init 関数

init 関数は初期化を行う関数で、コンストラクタで呼ばれている。引数として Model と整数値を受け取り、整数値によってどのステージを読み込むかを決定する。画像ファイルの読み込み、その他値の初期化、キャラクターを設置する charaSet 関数を呼び出す。

3.3.3. update 関数

update 関数は GameView の Timer で呼ばれており繰り返し実行されている。引数として Dimension 型のインスタンスを受け取り、updateOffset 関数に渡している。

3.3.4. updateOffset 関数

updateOffset 関数はスクロールでプレイヤーが画面の中央に来るように調節するオフセットを計算する関数である。画面に描画する範囲は x 座標では 0 からフレームの幅、y 座標は 0 からフレームの高さまでである。そのためフレームの大きさを超えるフィールドを移動させるとき、プレイヤーの位置に合わせて画面に描画する範囲に入るように計算する必要がある。

```
private void updateOffset(Dimension size){
    offsetX = size.width/2-(int)model.player.getX();
    offsetY = size.height/2-(int)model.player.getY();
    offsetX = Math.min(offsetX, 0);
    offsetX = Math.max(offsetX, size.width - WIDTH);
    offsetY = Math.min(offsetY, 0);
    offsetY = Math.max(offsetY, size.height - HEIGHT);
}
```

Dimension クラスは開いているフレームの大きさを知ることができる。この大きさとプレイヤーの座標を用いて描画するときにどれだけずらして描画するかを計算する。これが offsetX、offsetY となる。スクロール処理は https://aidiary.hatenablog.com/entry/20050624/1255786339 を参考にした。例として offsetX の計算を図に示す。

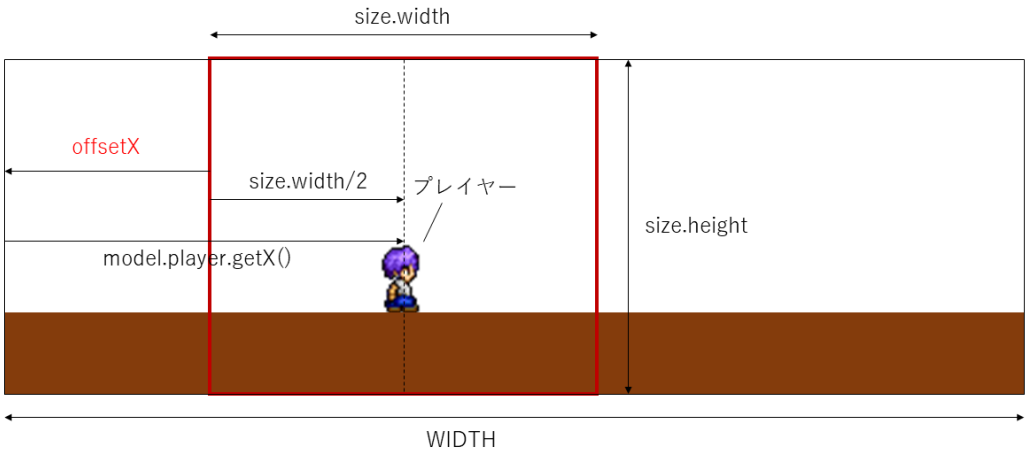


図3.2. offsetX の計算

またプレイヤーは常に中央にくるようにしたいが、画面の端ではフィールド外が見えないようにするため offset に最大値と最小値を決めておくことでフレームは常にフィールド内を映すことができる。このオフセットはスクロールで動くすべての描画で利用される。

3.3.5. draw 関数

draw 関数はフィールドのブロックや地面などの描写処理を行っている。描画を行うために Graphics 型のインスタンスを受け取っている。ブロックの位置は x 座標は何列目かとブロックの一つあたりの大きさである cs を用いて計算できる。しかしスクロールに合わせて描画させるので描画位置は offsetX を加えて補正している。y 座標も同じように何行目かと cs から計算でき、描画位置は offsetY を加えて補正する。二次元配列 map はマップの種類を表しているので、その値によって描画する画像を変えることで様々な画像を描画している。

3.3.6. collisionCheck 関数

collisionCheck 関数は引数でブロックの行と列、character を受け取り、そのキャラとそのブロックの種類を考慮してが衝突しているかを判定する関数である。返り値として、衝突していれば true、していなければ false を返す。衝突するブロックの種類であるかの判定はその位置の map の値が正であればブロックであるため衝突する。またここでは次のステージに移動するゴールとの衝突判定を行い、ゴールにプレイヤーが衝突すればモデルにゴールしたと伝える goal のフラグを立てる役割もある。

3.3.7. loadField 関数

loadField 関数は scv ファイルを読み込み、二次元配列 map を作る関数である。scv ファイルからフィールド情報を読み込めるようにすることで、フリーのマップエディタ等でス

テージの作成ができるようになり、マップ作製の自由度が上げられ、ステージ管理、作成が楽になる。ファイルの読み込みは https://aidiary.hatenablog.com/entry/20050624/1255786968 を参考にした。BuffredReader でファイルを読み込み、そこから列数と行数を ROW、COL に入れる。二次元配列 map は ROW と COL を用いて定義できる。また一つあたりの大きさ cs を定義しているので、フィールド全体の幅と高さである WIDTH、HEIGHT もわかる。読み込んだファイルは文字列になっているので、それを整数に変換しながら map に格納していく。

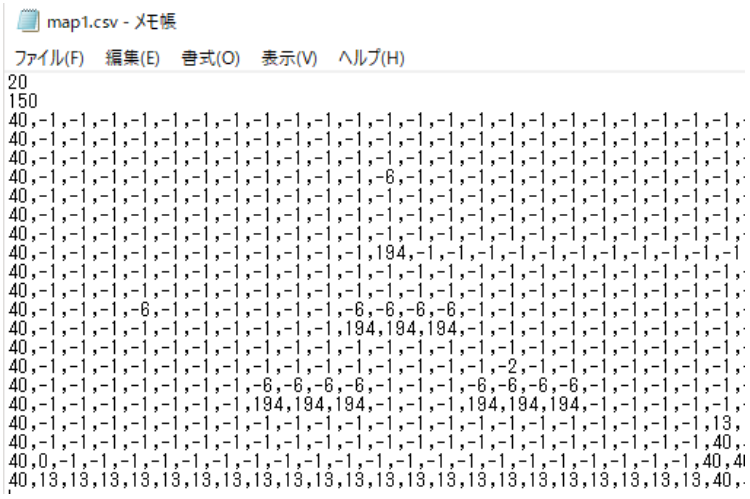


図3.3. csv ファイルの中

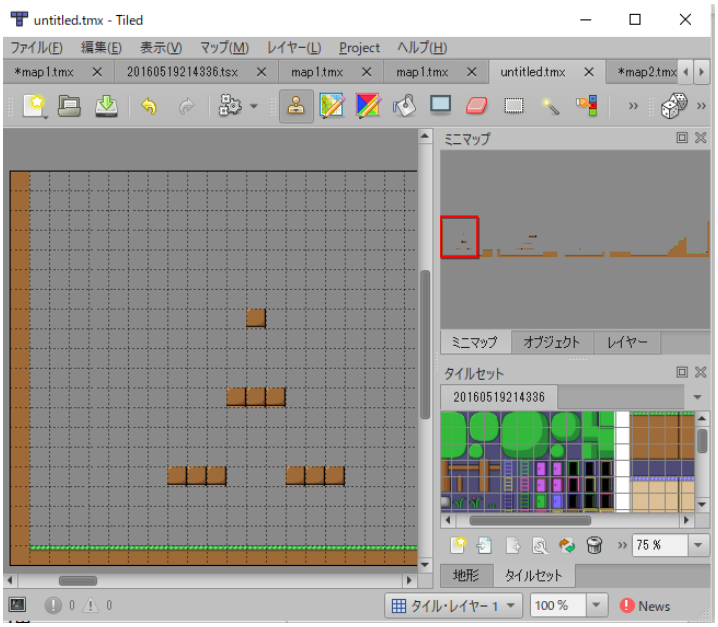


図3.4. フリーのマップエディタによるステージ作成

3.3.8. charaSet 関数

charaSet 関数は scv ファイルを読み込んで得られた map からその位置にキャラを設置する関数である。map の値が 0 なら Player、-2 なら Enemy1、-9 なら Boss と番号で判別してキャラを設置する。キャラの設置にはプレイヤーは Model クラスの createPlayer 関

数、それ以外は Model クラスの serCharacter 関数を用いて設置し、x、y 座標とどの型のキャラクターを設置するか決定する。

3.3.9. その他関数

get、set 関数を定義して、他クラスと ROW や COL、offsetX や offsetY、map の値などのやり取りを可能にしている。

3.4. Character クラス

Character クラスはフィールドとのあたり判定､移動などの継承して使うための基本的機能を定義している。

3.4.1. フィールド

width、height は地面とのあたり判定に用いられる。characterNum はキャラの種類判別用の数である。isGound、isCollisionX、isCollisionY、isDamaged、attackFlag はそれぞれ地面にいるか、x 軸方向で衝突しているか、y 軸方向で衝突しているか、ダメージを受けたか、攻撃のフラグが立っているかを示すフラグである。x、 y 、vx、vy、g はそれぞれキャラの x、y 座標、x、 y 方向の速度、重力の大きさを示している。RIGHT、LEFT、dir はキャラクターの向きを管理するのに用いている。向きはアニメーション等に用いられる。animationCount、count はアニメーションの遷移用であり、damageCount はダメージを受けたときの一定時間無敵をカウントする用の変数である。gw、gh はキャラの描画用の幅と高さであり、これを用いてキャラ同士のあたり判定を行う。playerX と playerY はプレイヤーの x、y 座標を保存している。

3.4.2. コンストラクタ

コンストラクタでは x,y 座標、幅、高さ、hp、キャラクター識別用の characterNum を受け取り各種変数の初期化を行っている。

3.4.3. update 関数

update 関数は Model クラスの update で呼ばれており、fall 関数、move 関数を呼び出し、ダメージを受けたときの無敵時間 damageCount が 0 以上なら減らしている。update 関数は Model の update で呼ばれ、繰り返し実行されている。

3.4.4. move 関数

move 関数は引数として Field クラスのインスタンスを受け取っている。ここでは field とのあたり判定を考慮して移動する collisionX 関数と collisionY 関数を呼び出している。

3.4.5. moveX 関数

moveX 関数はキーボードで受け取った入力から Character を動かすときに用いる。引数

は向きを表しており、正なら右向き、それ以外は左向きにし、入力の２倍を x 軸方向の速度にする。

3.4.6. fall 関数

fall 関数は重力処理で y 軸方向の速度 vy に対して重力加速度 g を加えている。

3.4.7. damaged 関数

damaged 関数は damageCount が 0 のとき、つまり無敵時間ではないとき引数だけ hp を減らし、無敵時間を設定している。

3.4.8. draw 関数

draw 関数は描画処理用の関数であるが、子クラスでオーバーライドして使うためここでは何も書いていない。

3.4.9. coliisionX 関数と collisionY 関数について

coliisionX 関数と collisionY 関数は Field とのあたり判定を考慮した x 軸方向、y 軸方向の移動の処理をしている。これは https://aidiary.hatenablog.com/entry/20081129/1281614716 を参考にした。update で collisionX 関数と collisionY 関数を呼び出すとき、X 軸方向の判定、y 軸方向の判定と順番に判定することができるので、まとめて実装するより単純に考えることができる。

3.4.10. collisionX 関数

collisionX 関数は次の移動先の x 座標である newX が Field と衝突していないかを調べている。２重の for 文を用いて Field の map と衝突するかを判定する関数 collisionCheck を用いて調べ、衝突していたらキャラの速度を用いて左右どちらから衝突したかを判定する。速度が正なら右から衝突しているので衝突した部分にめり込まないように x 座標を調整し、速度は 0 にする。x 座標の調整の図を示す。

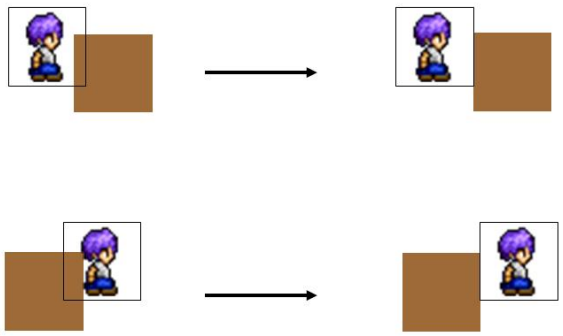


図3.5. 衝突時の x 座標の調整

速度が負の場合でも同じように x 座標の調整を行う。Field との衝突がなければ newX を新しい x 座標にする。このとき衝突していれば isCollisionX のフラグを立て、していなければフラグを下す。

3.4.11. collisionY 関数

collisionY 関数は基本的に collisionX を y 軸方向の判定に変えるだけだが、速度が正のときに Field に衝突する、つまり下から衝突したとき、それが壊れるブロックならその部分の数を-1（-1 はなにもないことを表す数字）にするという処理を追加している。

3.4.12. その他関数

他クラスで呼び出す get 関数、set 関数を定義している。例として getX 関数、getY 関数、getHp 関数、getCharacterNum 関数、setVy 関数、setSoundManager 関数、setPlayerLocate 関数などがある。

3.5. Player クラス

Player クラスではプレイヤーが操作するキャラを定義している。Player クラスは Character クラスを継承しているので、Character クラスで定義したフィールドや関数を使用することができる。

3.5.1. 主なフィールド

IMAGESIZE は読み込みたい画像サイズを定義している。icon は ImageIcon 型の変数で、画像の読み込みに用いられる。

3.5.2. コンストラクタ

x、y 座標を引数に受け取り、親クラスのコンストラクタを呼び出す。描画用の幅、高さを設定している。x、y 座標を引数として受け取れるようにしているのはフィールドデータを保存している scv ファイルに Player の初期位置で設定して、Field クラスで scv ファイルを読み込んだときに、その位置に Player を設置できるようにするためである。これは Character クラスを継承しているクラス全般で言える。

3.5.3. jump 関数

jump 関数はキーボードでジャンプの入力がされたときに Model クラスの jump 関数を呼び出すが、その Model クラスの jump 関数で Player クラスの jump 関数を呼び出している。プレイヤーが地面にいるとき、つまり isGround のフラフが立っているときは y 軸方向に速度を与え、jump の音をだし、isGround のフラグを下すようにする。

3.5.4. draw 関数

draw 関数は描画処理やアニメーションの処理を行っている。Character クラスの draw 関数をオーバーライドしている。まず画像を読み込むが、この画像はプレイヤーのアニメーション用に横にいくつか並んだ画像になっている。draw 関数内ではアニメーション用に count 変数が呼び出されるたびに増えており、これを用いてアニメーションを行う。この count を用いてアニメーション画像切り替えの番号の変数である animationCount を切り替える。damegeCount 変数を割ったあまりを用いて、描画をする、しないを切り替えることでダメージを受けたときの無敵時間中はプレイヤーをチカチカさせるアニメーションを行っている。また、横長のアニメーション用画像を切り抜いて描画する処理は次のように行っている。

```
g.drawImage(image, (int)x +offsetX, (int)y+offsetY, (int)x +offsetX+(int)width+3,
    (int)y+offsetY+(int)height+3,
    IMAGESIZE+animationCount*(int)width+dir*IMAGESIZE*2+2, 0,
    IMAGESIZE+animationCount*(int)width+(int)width+dir*IMAGESIZE*2-
    2,IMAGESIZE, null);
```

この切り替えは図 3.6 のようになっている。横長の画像から切り取りたい箇所を向きと animationCount と切り取りたい大きさ IMAGESIZE を用いて切り取って表示している。

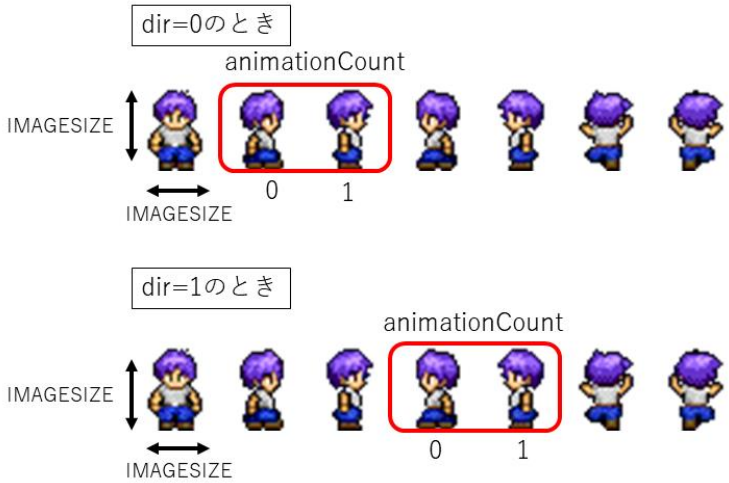


図3.6. 画像の切り取り方

3.5.5. damaged 関数

Character クラスの damaged 関数をオーバーライドしている。ほとんど Character クラスの damaged 関数と同じだが、無敵時間の長さが変わっている。

3.6. Ball クラス

Ball クラスは Player が攻撃するときに発射する球に関するクラスである。Ball クラスは Character クラスを継承している。

3.6.1. 主なフィールド

球が何回地面に当たったかを保存する fallcount がある。

3.6.2. コンストラクタ

親クラスのコンストラクタの呼び出し、fallcount の初期化、描画サイズの設定を行っている。

3.6.3. update 関数

update 関数は Character クラスの update 関数をオーバーライドしている。Character クラスの update に加えて、向きが右なら x 軸の正の向きに速度を与え、左なら負の即祖を与えるようにしている。また isGround が true のとき y 軸の負の方向に速度を与え、跳ねさせると同時に fallcount を数える。fallcount が一定数を超えると、hp を 0 にする。また x 軸方向で何かに衝突したときも hp を 0 にしている。

3.6.4. draw 関数

draw 関数はオフセットを考慮した描画処理を行っている。

3.7. Coin クラス

Coin クラスはプレイヤーが触れると、スコアになるコインに関するクラスである。Coin クラスは Character クラスを継承している。

3.7.1. 主なフィールド

アニメーション用の画像を保存する Image 型の配列 image がある。

3.7.2. コンストラクタ

Character クラスのコンストラクタに加えて、描画サイズの設定、画像を読み込み image に格納といったことを行っている。またコインは空中に浮かせたいため重力加速度 g は 0 にしている。

3.7.3. draw 関数

draw 関数はオフセットを考慮した描画処理を行っている。animationCount を呼び出されるたびに増やしながら、animationCount の剰余を用いて count を設定する。この count を使って表示する画像を切り替えることでアニメーションを行っている。

3.8. Boss クラス

Boss クラスは敵であるボスに関するクラスである。Boss クラスは Character クラスを継承している。

3.8.1. 主なフィールド

ボスの動きを設定するためのカウンターとして moveCounter、画像読み込みのための ImageIcon 型と Image 型の変数、ボスが空中で浮いているかを判定する isFly、乱数を用いて動かすための Random 型の変数が定義されている。ppx、ppy でプレイヤーの座標を保存する。

3.8.2. コンストラクタ

Character クラスのコンストラクタを呼び出し、描画サイズの設定、変数の初期化、player の位置を格納する setPlayer 関数を呼び出しを行っている。

3.8.3. draw 関数

draw 関数はオフセットを考慮した描画処理とアニメーションを行っている。アニメーションは向きと animationCount を用いて処理されている。animationCount は速度を持って移動しているときのみインクリメントすることで動いているときのみ、それに応じてアニメーションされるようになっている。

3.8.4. update 関数

update 関数は Character クラスの update 関数をオーバーライドしている。親クラスの update 関数を呼び出しに加えて、ボスの動きの処理を行っている bossMove 関数を呼び出す。

3.8.5. bossMove 関数

bossMove 関数はボスの動き関連の処理を行っている。ボスは 1 ループの間に 2 回プレイヤーの座標を得て、その向きを向く。これによって、行動中にプレイヤーとボスの位置関係が変わるたびに向きが反転することを防いでいる。そしてその 1 ループ中に 2 回行動を行う。そこに乱数要素を加えることでボスの複雑な動きを作っている。ボスの動きは moveCounter をインクリメントし、ある値を超えると 0 に戻ることで動きのループを作りながら、ボスの動きを制御している。ボスの動きの流れは図のようになっている。

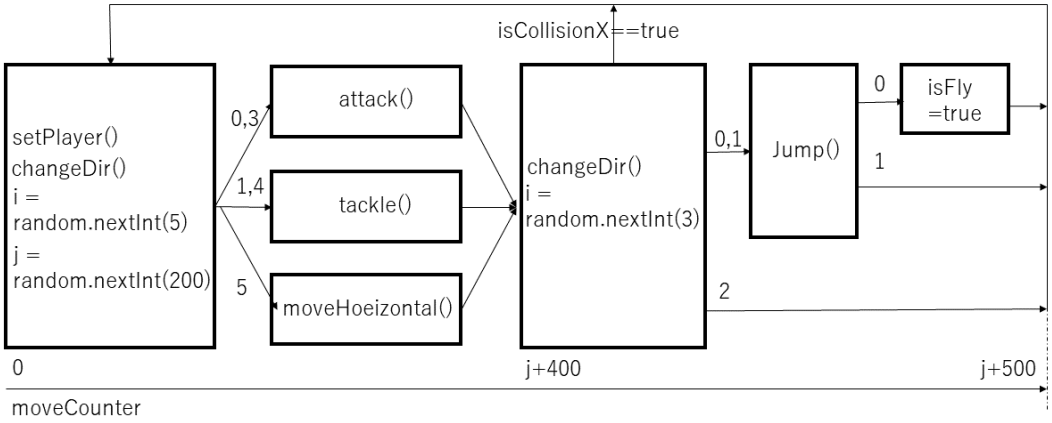


図3.7. ボスの動きの流れ

重力の制御を行う fly 関数もここで呼ばれている。moveCounter が 0 のときボスに攻撃の目標となるプレイヤーの位置を保存する setPlayer 関数、ボスをプレイヤーの方向を向かせる changeDir 関数を呼び、次の行動を決定するためにランダムに 0 から 4 の整数を得て i に入れる。j に 0 から 199 までの整数をランダムに入れる。j はボスの行動の一ループを不規則にするために用いる。その後 moveCounter を増やしながら、i の値に応じて球を出して攻撃する attack 関数、プレイヤーに突進して攻撃する tackle 関数、プレイヤーの方に移動する moveHorizontal 関数を呼び出す。moveCounter が j+400 になると、向きを変える changeDir 関数、次の行動を決定するランダムな数を求め、それに応じてボスが上に飛ぶ jump 関数や飛んだ後浮遊するため isFly のフラグを立てるといった行動を行う。j+500 で 1 ループが終わり、moveCounter を 0 にする。

3.8.6. jump 関数

jump 関数はボスに y 軸方向に負の速度を与えることでジャンプさせる関数である。

3.8.7. fly 関数

fly 関数はボスが浮いているときは重力の影響を受けず、それ以外のときは重力を受けるようにする関数である。

3.8.8. attack 関数

attack 関数はボスが球出して攻撃するとき呼び出す関数である。moveCounter の剰余から攻撃のタイミングを決定し、攻撃をするフラグ attackFlag を立て、soundManager から音を出す。この attackFlag は Model クラスの enemyAttackFlagCheck 中で監視され、true になると球を発射しフラグを下す処理をしている。

3.8.9. setPlayer 関数

setPlayer 関数は攻撃の目標として保存しておく ppx、ppy にプレイヤーの座標を入れる関数である。

3.8.10. tackle 関数

tackle 関数はプレイヤーの向けて突進攻撃を行う関数である。ppx を目標にして、ボスが目標より左にいれば正の向きに速度を与え、右なら負の向きに速度を与えている。プレイヤーの位置を持つ playerX、playerY という変数もあるが、これを目標にするとプレイヤーを追い越したときに向きの反転が起こってしまう。突進攻撃ではプレイヤーを追い越してもそのまま進んでほしいので ppx、ppy を目標にしている。

3.8.11. moveHorizontal 関数

moveHorizontal 関数はプレイヤーに向けて動く関数である。tackle 関数の移動の速度を変えたものになっている。

3.8.12. changeDir 関数

changeDir 関数はボスの向きをプレイヤーの方向に変える関数である。

3.9. BossFire クラス

BossFire クラスはボスが攻撃時に出す球に関するクラスである、BossFire クラスは Character クラスを継承している。

3.9.1. 主なフィールド

画像の読み込みで用いる ImageIcon 型と Image 型の変数、時間で消滅させるために使う count がある。

3.9.2. コンストラクタ

Character クラスに渡すための x、y、攻撃する速度の単位ベクトルを表す vx、vy を引数として受け取る。Character クラスのコンストラクタを呼び出し、速度の単位ベクトルを二倍したものを速度とする。さらに描画用サイズを設定し、変数の初期化、画像の読み込みを行う。

3.9.3. update 関数

update 関数は Character クラスの update 関数をオーバーライドしている。Character クラスの update を呼び出し、x、y 軸方向で衝突したとき、count が一定の値を超えたとき hp を 0 にする。

3.9.4. draw 関数

draw 関数ではオフセットを考慮した描画処理を行っている。

3.10. MainFrame クラス

MainFrame クラスはフレームを作成するためのクラスである。MainFrame クラスは javax.swing.JFrame クラスを継承している。

3.10.1. コンストラクタ

親クラスである JFrame クラスのコンストラクタを呼び出し、フレームをボタンで消せるように設定、フレームサイズの設定、サイズを変えられないようにする。

3.10.2. change 関数

change 関数では JPanel 型の panel を引数として受け取り、フレームに受け取った panel を設定する関数である。パネルの再描画とフォーカスの移動も行っている。これは SceneManager クラスの中でシーン遷移を行うときに呼ばれる。

3.10.3. main 関数

main 関数はコンソール等で呼ぶ実行関数である。SceneManager のインスタンスを作り、changeScene 関数を呼び出している。

3.11. GameView クラス

GameView クラスは MainFrame クラスに設定するゲームシーンのパネルであり、Timer による一定時間ごとの実行、すべての描画処理の呼び出しも行っている。GameView クラスは JPanel を継承しており、Observer と ActionListner をインプリメントしている。

3.11.1. 主なフィールド

Model、CharaController、Field、Timer、SceneManager、Dimension、SoundManager 型を格納する変数が定義されている。

3.11.2. コンストラクタ

コンストラクタでは Model、SoundManager、SceneManager を引数に受け取って、格納している。またパネルの背景、フォーカス、KeyListner への登録といった設定を行っている。また ActionPerformed 関数を一定時間間隔で繰り返し実行する Timer をインスタンス化し、繰り返し実行を始めている。SoundManager からステージごとの BGM のループ再生命令も行っている。

3.11.3. actionPerformed 関数

actionPerformed 関数は timer によって一定時間ごとに呼ばれるため、すべての update

処理を行っている。ここでは model、Field の update 関数を呼び出し、画面の再描画命令を行っている。Dimension 型の size はフレームの大きさを取得しており、Field クラスのオフセット計算で使われる。またシーン遷移のフラグ監視も行っており、ゲームオーバーシーン、次にステージのシーン、クリアシーンへのシーン遷移も行っている。フラグが立っているときは、フラグのリセット、BGM の停止、timer の停止、シーン番号切り替え、シーン遷移を行う。paintComponent 関数は『3.15.GameView の一部』に後述.

3.1-3.11.3　文責：新谷

3.12. TitleView1 クラス

このクラスはゲームのタイトル画面を描画している.

3.12.1. フィールド

SceneManager 型の変数 sceneManager と SoundManager 型の soundManager は画面や BGM 切り替え用の数値を代入するために宣言した. int x,y と boolean up,down,right,left は Title 画面でハートを動かせるおまけ要素用の宣言である.

3.12.2. TitleView1 関数

soundManager と sceneManager を引数とし，インスタンス変数にローカル変数を代入する．Thread 型のローカル変数 thread を用意した．これは処理を並行して行うことができるメソッドであり，sleep()でミリ秒で描画処理を止めることに用いた．これにより文字の RGM 値をランダムで決める処理を指定したミリ秒ごとに行うことを実現した．Catch(InterruptedException e)は別のスレッドが現在のスレッドに割り込んだ場合の例外処理だが，<参考１>を参考に今回は形式的に書いた．該当箇所を図 3.8　とする.

```
23              Thread thread = new Thread(){
24                  @Override
25                  public void run(){
26                      while(true){
27                          try{
28                              sleep(30);
29                          }catch(InterruptedException e){
```

図3.8　threadを用いた処理の並行

30 以降の if 分は Title 画面でハートを動かせるというだけのおまけ要素用で, down などが true の場合，座標を移動する.

3.12.3. paintComponent 関数

Graphics g を引数とし，まず super.paintComponent(g)で親クラスの paintComponent

メソッドを呼び出す．g.setColor();で setColor メソッドを呼び出し色を指定．g.fillRect()で fillRect メソッドを呼び出しパネルを描画．ここまでで背景を描画した．Font 型のローカル変数 font を定義し font オブジェクトを生成しフォント指定．g.setFont()で Graphics クラスの setFont メソッドに指定する．60 の setColor()は Color オブジェクトで random メソッドを呼び出し，RGB 値をランダムで指定することで文字の色をランダムに決定するようにした．drawString()で drawString メソッドを呼び出し文字を描画．83 は Image 型のローカル変数 image を宣言し，84 で Toolkit クラスの getDefaultToolkit メソッドを呼び出し getImage()で画像を取得した．drawImage で取得した画像を描画．83~86 は先述したおまけ要素である．

3.12.4. changeScene 関数

画面切り替え用のメソッドである．soundManager クラスの play,stop メソッドを呼び出し BGM を切り替えする．sceneManager クラスの setSceneNum メソッドを呼び出し sceneManager クラスで割り当てられている画面の番号をセットする．sceneManager クラスの changeScene メソッドを呼び出し，画面切り替え．

3.12.5. keyReleased 関数

Switch で離された各キーによって場合分けを行う．Space キーが押された場合，changeScene()メソッドを呼び出し画面切り替えし，System.ouy.println で space と出力した後 break で終了する．次の 37,38,39,40 は十字キーに割り当てられている番号である．各キーが離された場合，十字キーに対応する up などを false とする．

3.12.6. keyPressed 関数

Switch で押された各キーによって場合分けを行う．次の 37,38,39,40 は十字キーに割り当てられている番号である．各キーが離された場合，十字キーに対応する up などを true とする．

おまけ要素のプログラムを図　とする

```
32                  if(down){
33                    y+=10;
34                  }
35                  if(up){
36                    y-=10;
37                  }
38                  if(left){
39                    x-=10;
40                  }
41                  if(right){
42                    x+=10;
43                  }
```

図 3.9　　おまけの座標移動

```
83              Image hart;
84              hart = Toolkit.getDefaultToolkit().getImage("pictures/hart.png");
85
86              g.drawImage(hart,x,y,this);
```

図3.12.6.2　　おまけの描画

```
103         boolean up;
104         boolean down;
105         boolean right;
106     boolean left;
```

図 3.10　　おまけのフィールド

```
120             case 37:
121                 left = false;
122                 break;
123             case 38:
124                 up = false;
125                 break;
126             case 39:
127                 right = false;
128                 break;
129             case 40:
130                 down = false;
131                 break;
```

図 3.11　　KeyReleased 関数のおまけ要素に関わる部分

```
140                 case 37:
141                 left = true;
142                 break;
143                case 38:
144                 up = true;
145                 break;
146                case 39:
147                 right = true;
148                 break;
149                case 40:
150                 down = true;
```

```
151                     break;
```

図 3.12　　KeyPressed 関数のおまけ要素に関わる部分

<参考 1>『スレッドの一時停止』https://www.javadrive.jp/applet/thread/index7.html

3.13. TitleView,GameClearView,GameOverView クラス

TitleView クラスはゲームの操作説明画面を描画している．TitleView クラスについては文字の色をランダム指定しないため TitleView1 から Thread を省略した．またおまけ要素を省略した．GameClearView クラスはゲームをクリアした時に GameClear とスコアを描画している．67 で int 型のローカル変数を宣言し，model クラスの getScore メソッドを呼び出しスコアの値を取得する．drawString()で取得したスコアを描画する．GameOverView クラスはゲームをクリアした時に GameOver とスコアを描画している．66 で int 型のローカル変数を宣言し，model クラスの getScore メソッドを呼び出しスコアの値を取得する．drawString()で取得したスコアを描画する．GameClearView,GameOverView クラスは TitleView1 クラスからおまけ要素を省略した．これら三つのクラスは先述した部分以外のフィールドや関数は TitleView1 とほぼ同様であるため説明を省略する．

3.14. SceneManager クラス

SceneManager クラスは画面遷移の管理を行なっている．このクラス内で各画面の番号を設定することで画面遷移を実現している．

3.14.1. フィールド

sceneNum や model など各型の変数を宣言する．Private であるため，このクラス外からは干渉することはできない．

3.14.2. SceneManager 関数

フィールドで宣言した各変数に各型のオブジェクトを生成するようにする．sceneNum は初めは 0 とする．これはタイトル画面である TitleView1 が 0 として設定されているからである．

3.14.3. setSceneNum 関数

値を受け取り，その値を sceneNum とする．

3.14.4. getSceneNum 関数

sceneNum の値を取得する(返す)関数である．

3.14.5. changeScene 関数

Switch で場合分けを行い，各画面に対応する番号を設定し管理する関数である．

3.15. GameView クラスの一部

妹尾は GameView クラス内の paintComponent メソッド内のスコア表示と HP 表示を担当した.

スコア表示は, Font 型のローカル変数 font を定義し font オブジェクトを生成しフォント指定. g.setFont()で Graphics クラスの setFont メソッドに指定する. setColor()で setColor メソッドを呼び出し色を指定. Int 型のローカル変数を定義し, model クラスの getScore メソッドを呼び出しスコアを取得する. drawString()で drawString メソッドを呼び出しスコアを画面端に描画する.

HP 表示は, Font 型のローカル変数 font を定義し font オブジェクトを生成しフォント指定. g.setFont()で Graphics クラスの setFont メソッドに指定する. setColor()で setColor メソッドを呼び出し色を指定. Int 型のローカル変数を定義し, model クラスの getPlayerHp メソッドを呼び出し HP の値を取得する. drawString()で drawString メソッドを呼び出し HP を画面端に描画する. drawImage()で drawImage メソッドを呼び出し HP のアイコンとして画像を表示する.

3.12. TitleView1 クラスからここまでの文責 : 妹尾

3.16. CharaController クラス

CharaController クラスはキーボードの入力を処理してキャラクターの操作を行う、コントローラの役割を実現したクラスである。インターフェースとして KeyListener を実装している。

3.16.1. 主なフィールド

Model クラス内の move 関数を呼ぶために、Model 型の変数 model を使用している。

3.16.2. コンストラクタ

コンストラクタでは CharaController で処理を行う Model 型の引数を、上で述べた変数 model に格納する。

3.16.3. keyTyped 関数

今回、キーボード入力の処理は keyPressed 関数と keyReleased 関数にて行うため、keyTyped 関数の中身は何も書いていない。

3.16.4. keyReleased 関数

keyEvent 型の引数 e に対して getKeyCode 関数を用い、switch 文での条件分岐とし、該当するキー入力に対してそれぞれ Model クラスの move, shoot, jump 関数を呼び出すことで処理を行っている。左右への移動については move 関数に該当する方向の整数とフラグを

渡すことで実現している。

3.16.5. keyTyped 関数

keyReleased 関数と同様に、引数 e に対して getKeyCode を用い、switch 文での条件分岐として Model クラスの move 関数を呼び出すことで処理を行っている。

3.17. Enemy1 クラス

Enemy1 クラスでは左右に移動するスライム型の敵に関するクラスである。Enemy1 クラスは Character クラスを継承している。

3.17.1. 主なフィールド

進行方向とその速度を設定するための整数 wayx と描画用の画像を格納する ImageIcon 型の変数 icon1, icon2 がある。

3.17.2. コンストラクタ

コンストラクタでは Character クラスのコンストラクタを呼び出すことで x 座標や y 座標等の変数の初期化に加え、当たり判定の大きさに関する変数 gw, gh の設定を行っている。

3.17.3. update 関数

update 関数は Character クラス内の update 関数をオーバーライドしており、Character クラスの update 関数を呼び出したうえで、フラグ変数である isCollisionX を用いて Enemy1 が壁と衝突しているかを判断し、衝突している場合には wayx の+-を逆転させることで方向の反転を実現している。この時、isCollisionx を false に切り変えることで複数回反転の処理が行われないようにし、その後 moveX 関数を呼び出すことで左右への移動を実現している。

3.17.4. moveX 関数

moveX 関数は Character クラス内の update 関数をオーバーライドしており、Character クラスの moveX 関数に加え、改めて vx の値を設定しなおすとこで移動速度の変更を行っている。

3.17.5. draw 関数

draw 関数では描画処理を行っている。ImageIcon 型の変数 icon1, icon2 に描画用の画像をセットし、方向を表す整数 dir の値に応じて描画する画像を icon1 または icon2 に帰ることで描画上での Enemy1 の方向を切り替え、drawImage 関数を呼び出すことでオフセットを考慮した描画を行っている。

図 3.13. Enemy の描画切り替えの様子

3.18. Enemy2 クラス

Enemy2 クラスも左右に移動するスライム型の敵に関するクラスであり、Enemy2 クラスは Character クラスを継承している。Enemy1 クラスのバージョン違いとなっており、変更点としては整数 wayx やコンストラクタ内で設定する HP の値および描画用の画像が変わったくらいでほとんど同じである。実際、内容が似ていることから結果として Enemy2 は Enemy1 を継承して作った方が記述量が少なく早く書くことができると考えられるが、当初の設計方針では Enemy1, Enemy2 に機能を足していく可能性があったため、それぞれの拡張性を考慮して別々に作成したという経緯がある。

3.19. Enemy3 クラス

Enemy3 クラスはジャンプしながら左右に移動するゾンビ型の敵に関するクラスであり、Character クラスを継承している。

3.19.1. 主なフィールド

進行方向とその速度を設定する整数 wayx と描画用の画像を格納する ImageIcon 型の変数 icon1, icon2 に加え、整数 jumpcount がある。jumpcount はジャンプした回数を記録す変数であり、その回数に応じて定期的に高くジャンプするような処理を行う目的で作成している。

3.19.2. jum 関数

jum 関数では Enemy3 のジャンプの機能を実現している。フラグ変数 isGround を用いて Enemy3 が地面に接している時に限り vy の値を変え、isGround を false とすることでジャンプを可能にしている。ジャンプの内容は整数 jumpcount の値によって分かれ、jumpcount が 2 未満であれば低いジャンプとともに jumpcount に 1 を加える。Jumpcount が 2 以上であれば高いジャンプとともに jumpcount を 0 に戻すことで、3 回に 1 回のペースで高いジャンプをするような仕組みを導入している。

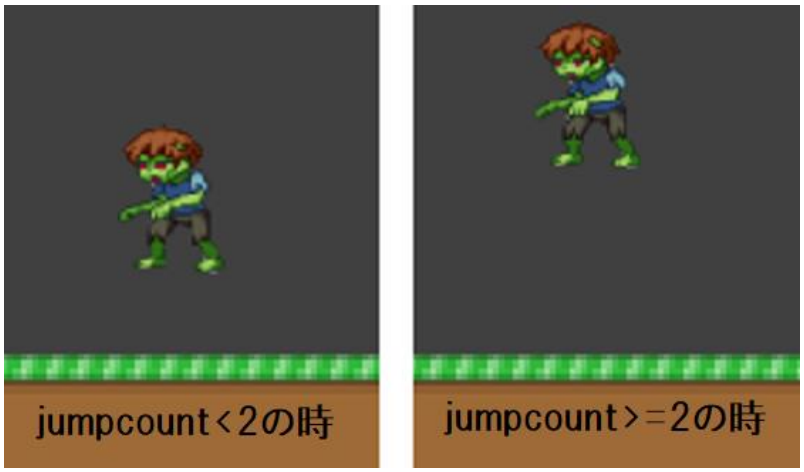


図 3.14. Enemy3 のジャンプの様子

3.19.3. update 関数

update 関数は Character クラス内の update 関数をオーバーライドしている。Character クラスの update 関数を呼び出した後、壁と接触時の反転処理と moveX 関数を呼ぶことまでは Enemy1 と同じであり、追加でフラグ変数 isGround が true のときに jum 関数を呼ぶことでジャンプを行うようにしている。

3.19.4. moveX 関数

moveX 関数は Character クラス内の update 関数をオーバーライドしており、Enemy1 クラスの moveX 関数と同様に、Character クラスの moveX 関数に加え、改めて vx の値を設定しなおすとこで移動速度の変更を行っている。

3.19.5. draw 関数

draw 関数では Enemy1 の draw 関数と同様にしてオフセットを考慮した描画処理を行っている。

3.20. Enemy4 クラス

Enemy4 クラスはプレイヤーが打つ攻撃の弾と同様の弾を用いて攻撃を行う魔法使い型の敵に関するクラスであり、Character クラスを継承している。Enemy4 は移動は行わず、一定の間隔でジャンプをしながら攻撃をする敵であり、X 座標に関してプレイヤーが追い越した際にはプレイヤーのいる方向に攻撃の方向を切り替えて攻撃する仕様となっている。

3.20.1. 主なフィールド

描画用の画像を格納する ImageIcon 型の変数 icon1, icon2 に加え、ジャンプや攻撃までの時間を管理するための整数 cooltime がある。

3.20.2. コンストラクタ

コンストラクタでは Character クラスのコンストラクタを呼び出すことで x 座標や y 座標等の変数の初期化、当たり判定の大きさに関する変数 gw, gh の設定を行うとともに、方向 dir の設定と攻撃を管理するフラグ変数 attackFlag の設定を行っている。なお、Enemy1～3 とは異なり、Enemy4 の CharacterNum を２として設定することで、Model クラス内で Enemy4 クラスのみを識別できるようにしている。

3.20.3. jum 関数

jum 関数はジャンプ機能に関する関数である。Enemy4 が地面に接していることを判断するフラグ変数 isGround を用いて vy を変化させることでジャンプを実現しており、機能としては Enemy3 クラス内の jum 関数の簡易版である。

3.20.4. update 関数

update 関数では Character クラス内の update 関数をオーバーライドしている。Character クラスの update 関数を呼び出したのち、update 毎に cooltime のカウントを増やしていき、cooltime の値によって jum を呼び出すことや、attackFlag を切り替えたうえで cooltime をリセットすることで規則的なジャンプ攻撃を実現している。切り替えた attackFlag は Character クラスの getAttackFlag 関数を経由して Model クラス内の enemyAttackFlagCheck 関数で処理を行い、Enemy4 の dir の変更や EnemyBall を追加することによって攻撃として処理、描画されるようにしている。

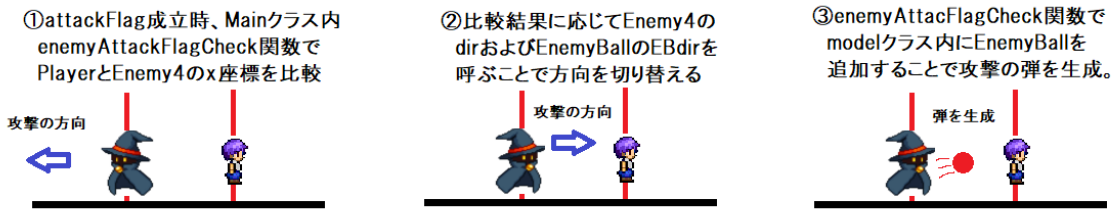


図 15. Enemy4 の方向および攻撃方向の切り替え

3.20.5. draw 関数

draw 関数ではオフセットを考慮した描画処理を行っている。Enemy4 の dir の変更は attackflag が成立した時に処理が行われるため、描画上での左右の反転が行われるのは攻撃時となっている。

3.21. EnemyBall クラス

EnemyBall クラスは Enemy4 の攻撃の弾に関するクラスである。EnemyBall クラスは Ball クラスを継承しており、Ball クラスを Enemy4 ように改良したものとなっている。

3.21.1 主なフィールド

EnemyBall クラスとして新たに設定したフィールドとしては、攻撃の弾の効果時間を管理する整数 fallcount を private として扱うために追加したことが挙げられる

3.21.2 コンストラクタ

コンストラクタでは Ball クラスのコンストラクタを呼び出したのち、changeCharaNum 関数を呼ぶことや各項目について設定することで初期設定を行っている。

3.21.3 changeCharaNum 関数

changeCharaNum 関数では EnemyBall の CharacterNum の変更を行っている。

3.21.4 update 関数

update 関数では Ball クラス内の update 関数をオーバーライドしている。Ball クラスの update 関数を呼び出したのち、方向 dir に応じた攻撃の弾の速さと y 軸方向の跳ね方の調整したうえで設定し、また攻撃の弾の効果時間を変更し、Ball クラスのものと同様に fallcount を用いることにより Model クラス内で消滅の処理を行えるようにしている。

3.21.5 EBdir 関数

EBdir 関数は EnemyBall の方向 dir を変更するための関数である。この関数は Model クラス内の enemyAttackFlagCheck 関数の処理において呼び出され、プレイヤーと Enemy4 の x 座標の位置関係から EBdir に渡す引数を変更することで攻撃の射出方向を切り替える仕組みとなっている。

3.21.6 draw 関数

Draw 関数ではオフセットを考慮した enemy4 の攻撃の弾の描画処理を行っている。

3.16 から 3.21.6 までの文責：前田

4. 実行例

実行例についてはタイトル画面、ゲーム画面、ボス戦と順を追って記載する。

4.1. タイトル画面

タイトル画面からスペースキーを押すことで操作説明ページに、さらにスペースキーを押すことでゲーム画面へ遷移する。

図 4.1.　タイトル画面からゲーム画面へのシーン遷移

4.2.　ゲーム画面

4.2.1.　プレイヤーの移動・攻撃と敵の撃破

プレイヤーは A および D キーで左右へ移動でき、その様子は複数のイラストを切り替えることによってアニメーションとして描写される。またプレイヤーは、S キーを押すことにより、赤い弾を発射することで攻撃を行うことができる。

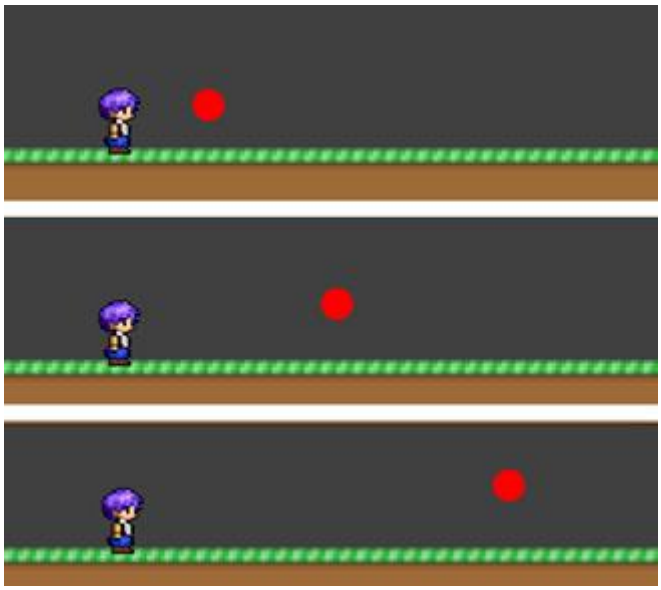


図 4.2. プレイヤーの攻撃の様子

敵は弾による攻撃や上から踏みつけることによって撃破することができ、撃破時にはスコアが加算される。敵の撃破の様子は次のようになり、スコアが加算されていることも確認できた。

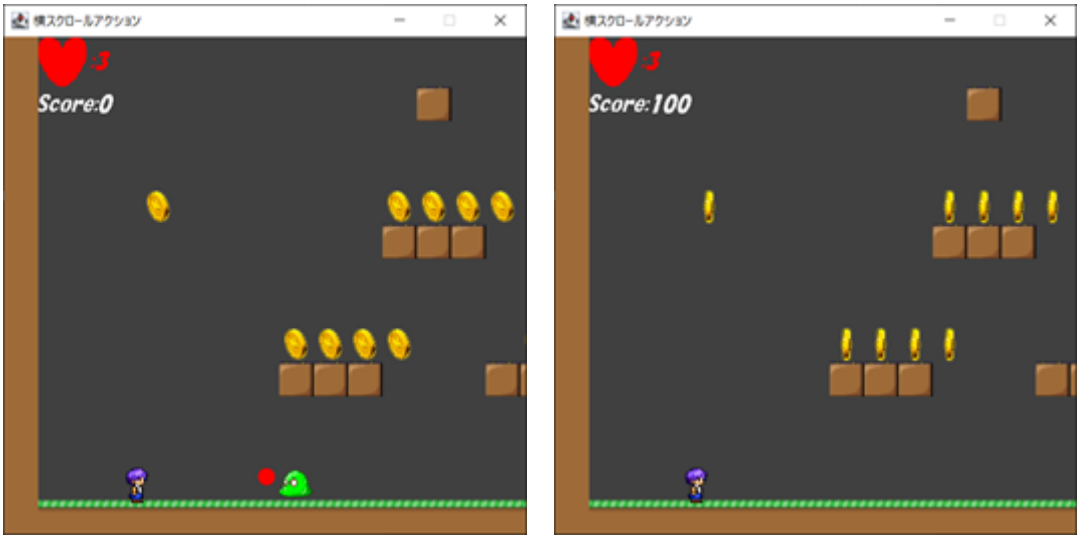


図 4. 3. プレイヤーの弾による攻撃での敵の撃破

4.2.2. ダメージを受けた時の変化

プレイヤーが上方向以外から敵に接触および敵の攻撃の弾に接触時にはプレイヤーはダメージを受け、画面左上に表示されるハートマーク横の HP のカウントが減少する。

図 4. 4. プレイヤーがダメージを受けた時の変化

4.2.3. プレイヤーのジャンプとブロックの破壊

プレイヤーの操作では左右の移動と攻撃に加えて、スペースキーを押すことによるジャンプ機能が追加されている。

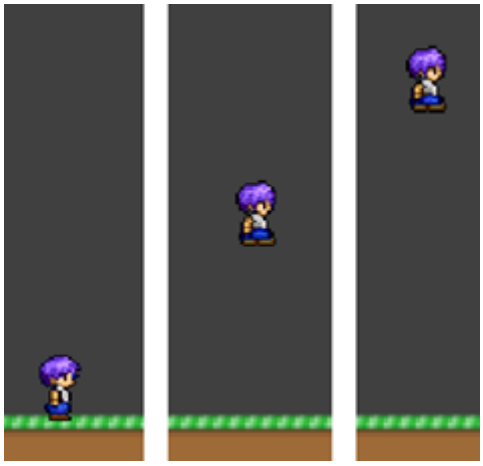


図 4.5. ジャンプの様子

また、フィールド上に設置されているブロックを下から叩くことによって破壊することができる。

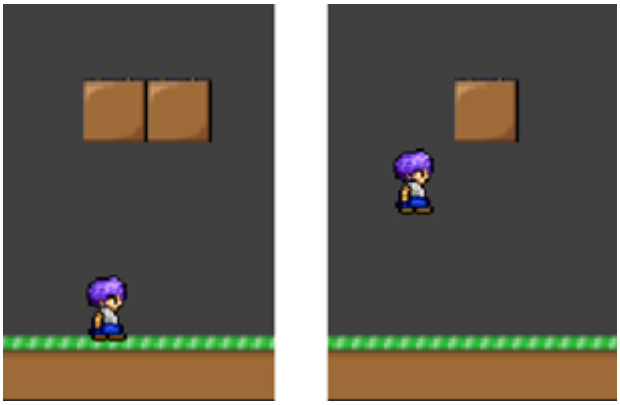


図 4.6. ブロックの破壊

4.2.4. コイン獲得時の変化

プレイヤーがコインに触れることでコインは消滅し、画面左上のスコアに 200 点が足される。コインは複数のイラストを切り替えることによってアニメーションとして描写している。

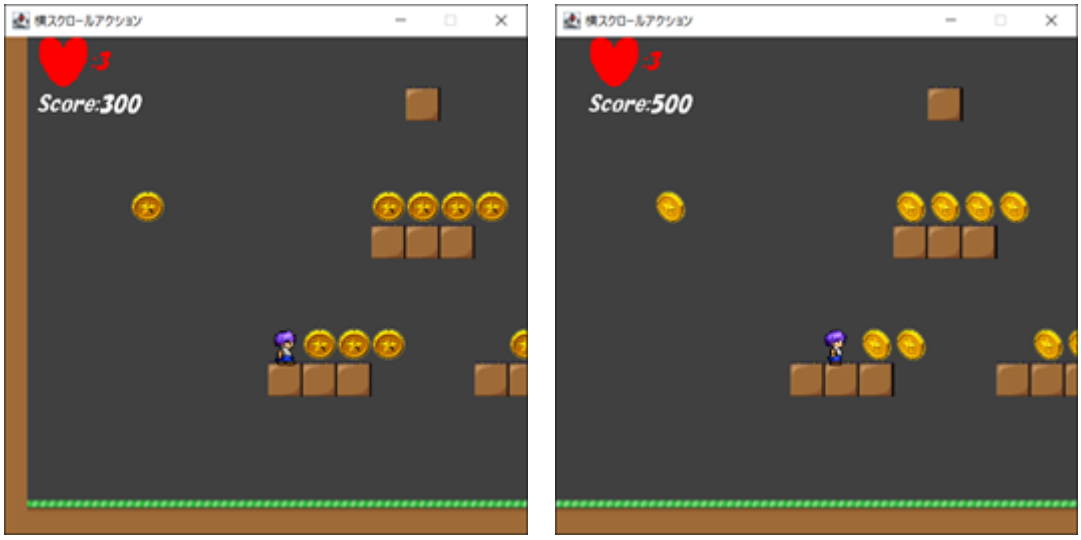


図 4.7. コイン獲得時の変化

4.2.5. の行動の様子

スライム型の敵(Enemy1 と Enemy2 に該当)の行動は左右の移動を繰り返すものとなっている。壁への衝突時には方向を反転し、折り返しでの移動をする。

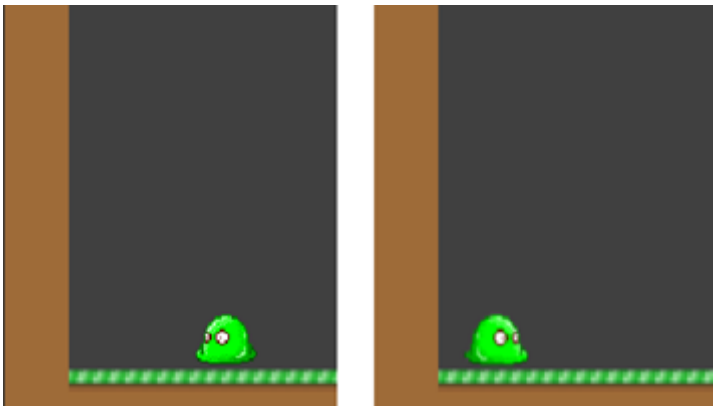


図 4.8. スライム型の敵の壁に衝突時の変化

ゾンビ型の敵(Enemy3 に該当)の行動では、左右の移動に加えてジャンプをしながら移動する。ジャンプは 3 回に 1 回のペースで少し高くなる仕様となっている。

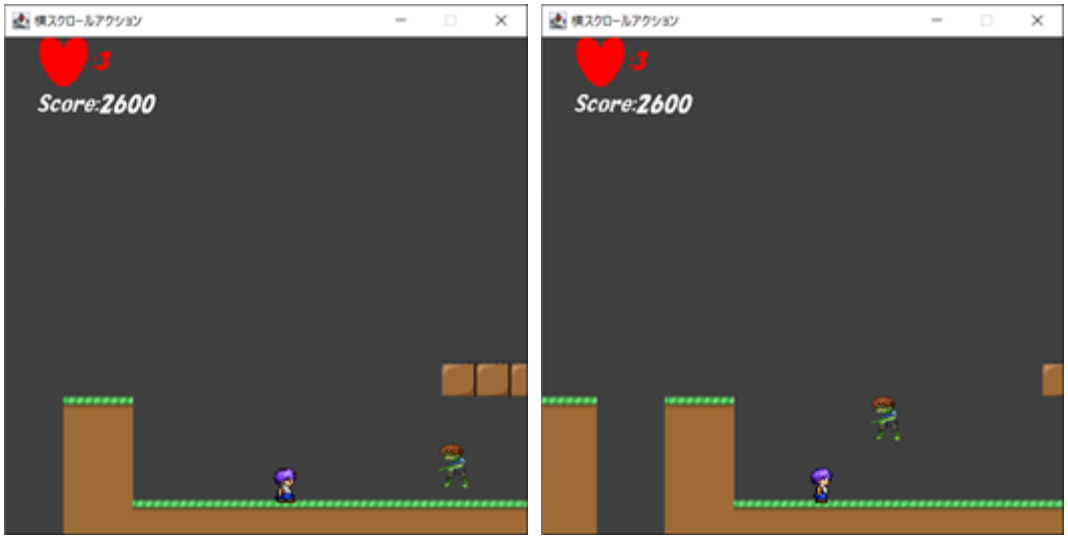


図 4.9. ゾンビ型の敵のジャンプ移動の様子

魔法使い型の敵(Enemy4 に該当)の行動では一定時間おきにプレイヤーが攻撃をする弾と同様の弾を用いたジャンプ攻撃を行うようになっている。

図 4.10. 魔法使い型の敵の攻撃の様子

また、魔法使い型の敵ではプレイヤーがいる方向に合わせて攻撃方向を変える仕様を実装しており、その使用が作動していることも確認できた。

図 4. 11. プレイヤーのいる方向に合わせた魔法使い型の敵の攻撃方向の切り替えの様子

4.2.6. ゲームオーバーへの遷移

プレイヤーの HP が 0 になる、またはフィールド上の穴に落下するとゲーム画面からゲームオーバー画面へと遷移する。ゲームオーバー画面でスペースキーを押すことでタイトル画面へと戻る。

図 4. 12. ゲームオーバー時の画面の遷移

4.2.7. ボス戦への遷移

フィールド上にあるゲートをくぐるとボス戦へとシーンが遷移する。シーン遷移時の動作や、遷移後にも不具合なく動作することが確認できた。

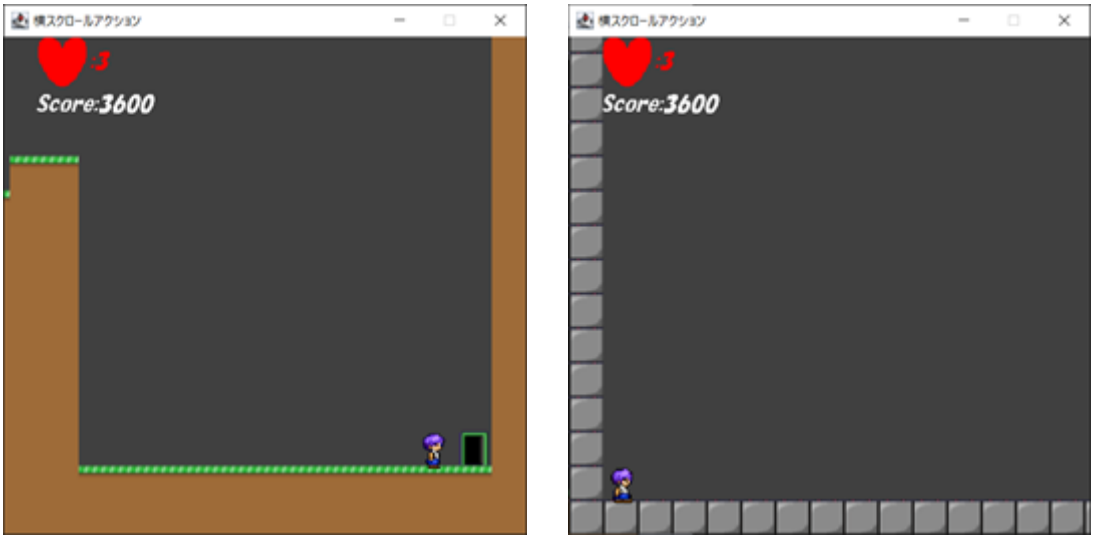


図 4. 13. フィールドからボス戦への遷移

4.3. ボス戦

4.3.1. ボスの行動

ボス戦のシーンではドラゴンがボスとなっており、移動、飛行、ブレス攻撃といった行動をランダムに実行するようになっている。ブレス攻撃ではプレイヤーの上下の位置に合わせて方向を変えた攻撃をする仕様となっている。

図 4. 14. ボスの行動(左から順に移動、飛行、ブレス攻撃)

また、ボスのドラゴンの行動では複数のイラストを切り替えることによりアニメーションとして描写している。

4.3.2. クリア画面への遷移

ボス戦にてボスを倒すことでクリア画面に遷移する。クリア時にはスコアに 10000 点が追加され、クリア画面ではスペースキーを押すことでタイトル画面に戻るようになっている。

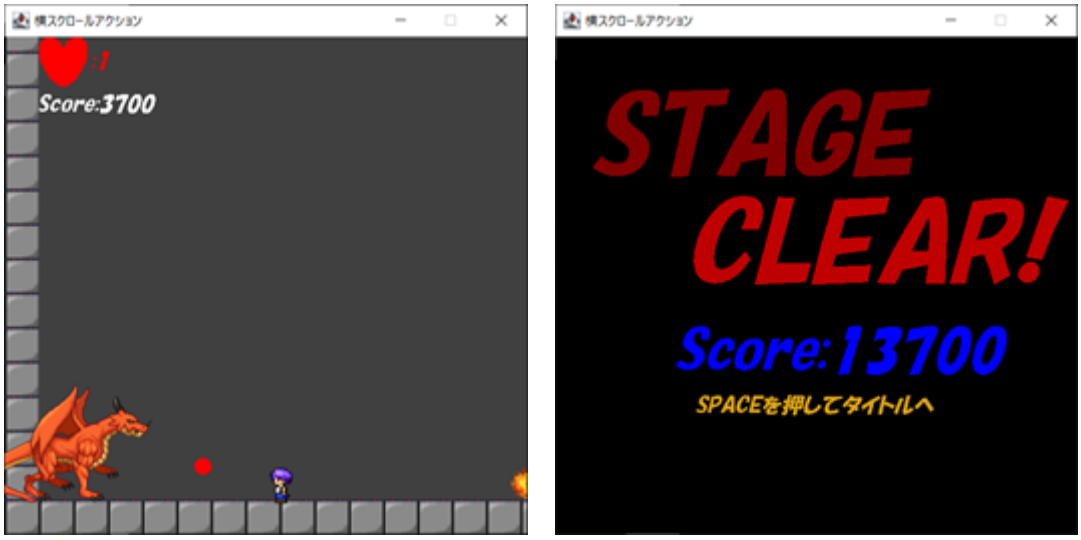


図 4.15.クリア画面への遷移

文責：前田

5. 考察

製作開始時から予定していた、MVC モデルでの分担・作成を前提として、座標や当たり判定の処理、描写やキャラクター操作等の横スクロールアクションにおける最低限の機能を実現できたことに加え、製作途中から追加することを決めた敵の実装やタイトル画面、音声やスコア・HP 表示等のゲーム性や完成度を高める機能についてもおおむね実現することができたため、予定していたものに近いものが作成できたといえる。今回グループプログラミングで作成したクラスの中でも、Player、Enemy、Ball 等のクラスが継承する Character クラスの作成及び同クラス内でのキャラクターを番号で識別する変数 characterNum の導入により、当たり判定や重力等の処理を継承することで実現できるため、敵やコインなどの要素の作成のしやすさに加え、Model 内でのキャラクターの種類ごとの分類・処理を容易にする働きを果たしており、設計のしやすさや拡張性の観点で良い設計であったと考えられる。

今回のグループプログラミングでは、上記のように横スクロールアクションの基本的な機能の実現を最初の目的として制作していたこともあり、今後の改善点としてはユーザーへの配慮を行った機能を追加することが挙げられる。ユーザーへの配慮を行った機能としては大きく２つあり、1 つ目はポーズ画面やオプション機能の追加が挙げられる。現状、この作品においてタイトル画面に戻ることができるのはゲームクリア時またはゲームオーバー時のみとなっており、途中でゲームを中断することができない。また、サウンドの調整機能なども実装されていないため、ポーズ機能からタイトルに戻る機能、オプション機能などを追加することによってユーザーの利便性を考慮した作品になると考えられる。2 つ目はブロックを破壊する機能に対してインセンティブがないことが挙げられる。現状ではブロッ

クを破壊してもスコアや HP などには全く影響しないため、ブロックを壊すことに対しての動機づけが一切ない状態となっている。せっかく実装した機能を活用するために、回復やパワーアップ等のアイテムを導入し、ブロック破壊と関連付けることで新しいゲーム要素の追加につなげることで改善できると考えられる。

文責：前田

今回、ゲーム中に出てくるキャラクターはすべて Character クラスを継承していた。継承した子クラスはモデルで Character クラスの ArrayList で管理できるため、キャラクター全体の管理が簡単になった。しかし、Chatacter クラスは当初フィールド上を動くものを想定して作っていたため、コインのような動かずあたり判定だけ行いたいものまで Character クラスを継承する必要があった。また、Character クラスを継承したクラス同士で被る処理があるため、冗長になっている部分があった。今後アイテムなどを実装するときもすべて Character クラスを継承していくことになる。そこで、Character クラスが継承するさらに抽象的なクラス GameObject クラスのようなものを作り、キャラクターとアイテムのクラスはどちらも GameObject クラスを継承することが考えられる。これによってより深い階層構造ができるため必要な機能ごとの分類を明確にでき、機能が重なる部分は親クラスのものを使えるようになる。また、Field クラスで実装したフィールドは二次元配列の値で管理しているため、固定されたブロックしか作れない。そこでフィールドの管理を GameObject 型の二次元配列で行うことでブロックは GameObject を継承したクラスで定義することができ、動くブロックのようなブロックが作成可能になり、より自由度があるステージ作成が可能になると考える。以下に GameObject を用いて階層を深くした Character 関連の改善案を挙げる。

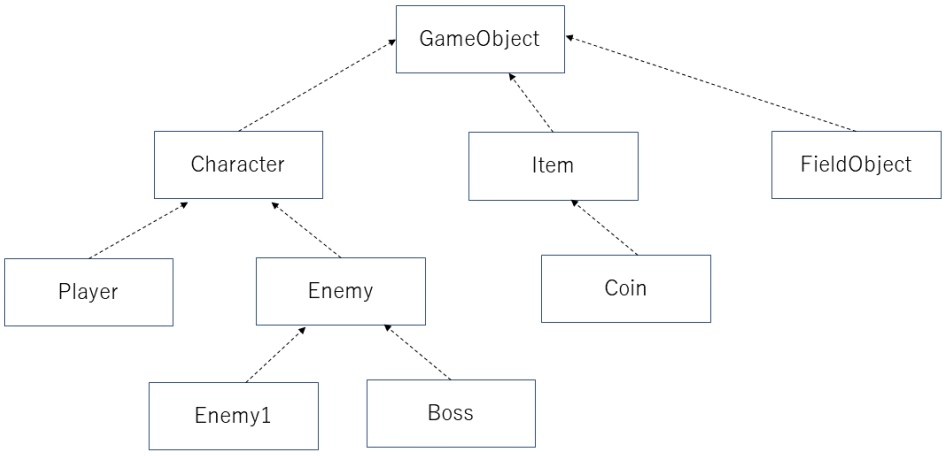


図5.1. Character の改善案

文責：新谷

6.　各自の反省と感想

複数人で共同してプログラムを書いたのは初めてだったので苦労した点がいくつかあった。まず、プログラムを他人が見てもわかるように書く必要があることである。これは、自分だけがわかるプログラムではなく、他の人が見やすいようにインデントやコメントをつけるといったことを意識して行う必要があった。また、書いている人が異なるクラスでやりとりをするため予め必要な変数や関数を相談して決める必要があった。

苦労はしたが、今まで書いていたプログラムでは上記のことが足りていなかったことに気づけ、いい経験になったと感じた。

今後の改善点は考察にも書いた Character 関連のクラスの整理やカプセル化が徹底できていないところがあるので、改善したい。また複数人で遊べるように通信機能をつけるための拡張をモデルでしていきたい。

MVC に分けることでそれぞれの役割がはっきりしたところはよかった。オブジェクト指向によって、関係のないクラスの内容は抽象化して知らなくても動くので複数人での開発に向いていると感じた。

授業に関しては、ただ複数人で開発するだけでなく、最後に発表する機会があったことで開発のモチベーションになった。

文責：新谷

僕はIEDにリモート接続してJavaプログラミング作業を行なっていた．しかし，共有されたファイルをダウンロードして，IEDリモート接続でのコンパイルが出来なくなった期間があった．原因はjavaのバージョンの違いによるエラーだったが，当時は原因不明だったため一週間ほど作業出来ない期間があった．そこで，グループメンバー(新谷君)に僕の作業を一部分担して手伝ってもらったことがあった．他にも自分だけでは解決できなかった問題を話し合って助けてもらうことがあり，共同作業の重要性を実感した．プログラムに関しては，僕は主にViewを担当しタイトル画面などを作成したが，デザインがシンプル過ぎたと感じた．タイトル画面にキャラクターや敵を散りばめ動かすだけでも，より興味を持たれるタイトル画面になっていたと反省した．また，実際にキャラクターを動かす前の段階で，自身の名前を登録し，ゲームクリア後に成績を記録しリストにする実装を行えば，このゲームへのやり込み度が上がると考えた．MVCモデルは事前に何を作成するかを十分に話し合うことを条件に共同作業に適していると感じた．プログラミング演習は前半の個人作業と後半のグループワークの難易度の差が大きいと感じた．

文責：妹尾

グループでの作業を通して一番大変だったことは、使用する変数や関数の変更についての情報共有が不十分だったことにより、互いのプログラムの統合時に必要以上の時間がかかってしまったことである。よく、エンジニアにはコミュニケーション能力も必要という話

を聞くが、今回のこの経験でその理由を身に染みて理解することができた。自分からあまり情報を伝えなかったところや相手側から共有されなかったところもあるため、今後このようなグループプログラミングの機会では自分から積極的に情報共有を行っていくことや、適度に相手側の状態を確認する旨の連絡を取ることによって円滑に進められるように意識したい。また、コードの共有という部分では他人から見ても理解できるよう見やすいコードを書くことの必要性も感じた。現に共有されたコードを読解することに時間がかかり、新しい機能を実装するためのコードを書き始めるまでに時間がかかってしまった部分があり、作業効率が悪くなってしまっていた。おそらく、自分が作成したコードでも他者から見たら読解に時間がかかる部分が多くあると思われるため、プログラミングをする際には他者が見ても見やすいコードを書くことを今まで以上に意識するようにしたい。

自分の担当部分での改善点としては敵の行動がワンパターンであることが挙げられる。今回実装した敵では左右に移動する、ジャンプしながら移動する、攻撃をするという 3 パターンであるが、どれも規則的な動きしかしないものになっている。そのため、ジャンプや攻撃の間隔に乱数を設定することで不規則的な動きにしたり、プレイヤーの動きに連動した行動をとるような敵などを追加したりと敵の行動パターンにもっとレパートリーを持たせればゲーム性および完成度をもっと上げられたと思う。また、Enemy4 クラスの制作時、この敵を反転させる動作を実現するためにやむを得ず Model クラスに追加してもらった部分もあったが、独立性、拡張性という点で考えるとあまり良いものではなくなってしまうため、一工夫することで改善出来たらよかったなと思っている。

今回、MVC モデルでのグループプログラミングをやってみて、MVC に分けることでそれぞれが独立しての修正や拡張ができるため、統合後こそはやしやすく感じたものの、統合前には直接動作を確認できないこともあり、やりづらく感じることもあった。統合前に感じた不便さを改善するためにも MVC モデルを採用する際には実装する機能に応じてクラスを設計し、クラス図を書くことでクラス間の関係を明確にしたうえで分担や共有する変数や関数を決め、連絡を取り合って進めるようにスタートダッシュの部分に力を入れることが重要だと考えられる。

授業の感想としては、前半に対して後半がかなりハードだと感じた。後半のグループプログラミングでは不足している知識を復習や調べることで探り探り進めていく必要があり、最初こそは完成の見通しが立たずストレスに感じる部分も多かったが、次第に完成に近づくにつれて制作意欲などもわいてきたことにより、なんとか形にすることができた。全体的に大変だったと感じてはいるが、今回のように実際にグループプログラミングをする機会は初めてであり、その様子を知ることができたという点でも貴重な体験だったと思う。

文責：前田

7. 付録１：操作法マニュアル(ユーザーズマニュアル)

まずタイトル画面で Space キーを押すと次の画面に遷移する．(おまけ要素として十字キーでハートを動かすことができる.)

図 7.1　タイトル画面

次にこのゲームの操作説明画面である．D キーで進む．A キーで戻る(後退する).S キーで攻撃する(火の玉を出す). SPACE キーでジャンプ. Space キーを押すとゲーム開始である．

図 7.2　説明画面

ゲーム画面である．左上に HP とスコアを表示している．敵を倒したり，コインを取得することでスコアを獲得する. HP ゼロにならずに, 崖に落ちずにゴールを目指してください. ゴール地点にたどり着くと boss 戦に移ります.

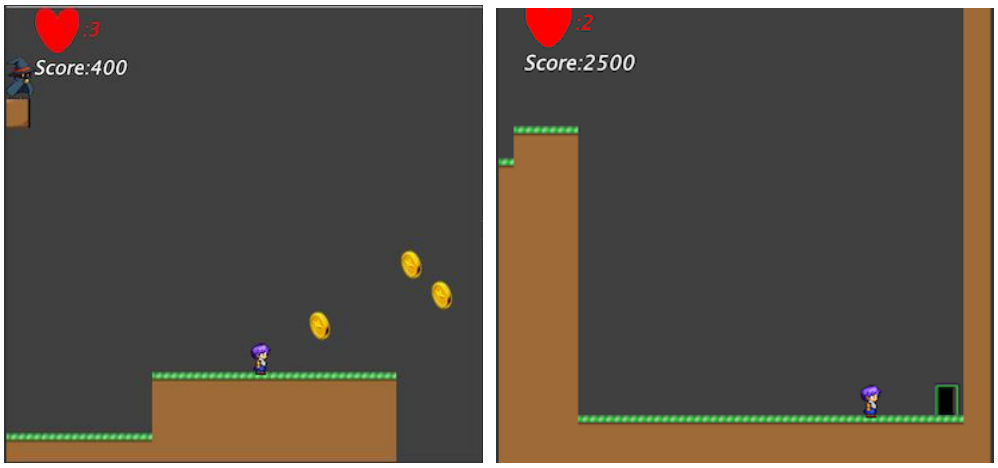


図 7.3　ゲーム画面

HP が 0 になる又は崖から落ちてしまうと GameOver になる．それまでのスコアが表示される．SPACE キーを押すとタイトル画面に遷移する．

図 7.4　ゲームオーバー画面

Boss 戦になると Boss が登場する．この敵を倒してもいいし，避けてゴールすることもできる．火の玉は攻撃しても消すことができず，避けることしかできない．

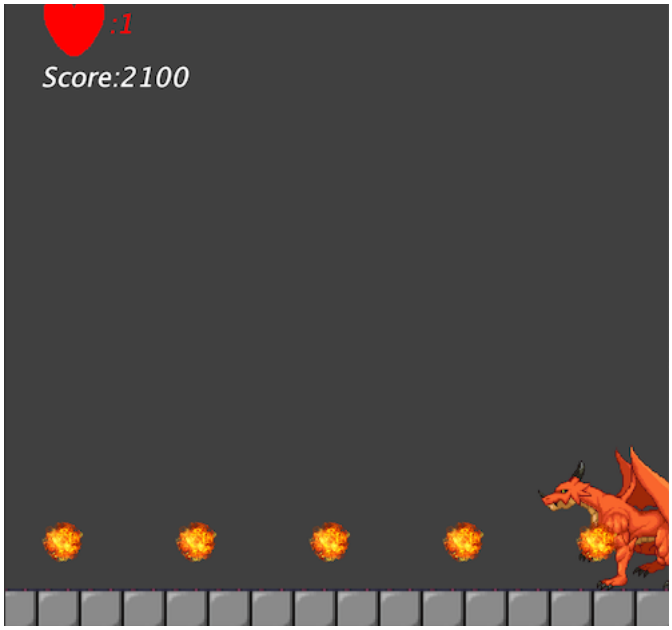


図 7.5　Boss 戦

Boss を倒す又はゴール地点に行くと GameClear 画面に遷移する．これまで獲得したスコアが表示される．SPACE キーを押すとタイトル画面に遷移する．

図 7.6　ゲームクリア画面

文責：妹尾

8. 付録２：プログラムリスト

8.1. Model.java

```
1  import javax.sound.sampled.*;
2  import javax.swing.*;
3
4  import java.applet.AudioClip;
5  import java.awt.*;
6  import java.awt.event.*;
7  import java.util.*;
8  import java.io.File;
9
10
11 public class Model extends Observable{
12     private ArrayList<Character> chara;
13     public Player player;
14     private boolean canMove, canJump;//使われていない
15     public Field field;
16     private boolean pressedKeyRight, pressedKeyLeft;//コントローラーからの入力
17     private boolean isSoundLoad = false;//sounManager が呼び出されているか
18     public boolean goal,gameOver, bossFlag, stageClear;//シーン遷移用フラグ
19     private int score,stageNum;//score と現在のステージの番号
20     private SoundManager soundManager;
21     //読み込む音データ指定
22                 private static final String[] soundNames = {"coin","jump","block","enter","shoot","stomp", "decide", "title","boss","field1","gameclear","gameover","fire","field2","title1"};
23     private int ccc;//debug
24     public Model(int i, SoundManager soundManager){
25         init(i, soundManager);
26     }
27     //初期化用
28     public void init(int i, SoundManager soundManager){
29         chara = new ArrayList<Character>();
30         field = new Field(this, i);
31         this.soundManager = soundManager;
32         pressedKeyRight= false;
```

```
33              pressedKeyLeft = false;
34              goal = false;
35              gameOver = false;
36              bossFlag = false;
37              stageClear = false;
38              stageNum = i;
39              if(i==0){
40                  score = 0;
41
42              }
43              ccc = 0;
44              loadSound();
45          }
46
47          public void createPlayer(int x, int y){//Player 作成用
48              player = new Player(x, y);
49              player.setSoundManager(soundManager);
50              chara.add(0,player);//player は 0 番目
51              //setChanged();
52              //notifyObservers();
53          }
54          //view で毎フレーム更新
55          public void update(){
56              //Controller の入力に対する移動
57              if(pressedKeyRight){
58                  player.moveX(1);
59                  pressedKeyLeft = false;
60                  //System.out.println("right");
61              }else if(pressedKeyLeft){
62                  player.moveX(-1);
63                  pressedKeyRight = false;
64                  //System.out.println("left");
65              }else if(!pressedKeyRight && !pressedKeyLeft){
66                  player.moveX(0);
67              }
68              //落下と移動処理
69              for(int i=0;i<chara.size();i++){
70                  if(i==0){//player
```

```
71                      chara.get(i).update(field);
72                  }else{//player 以外の update は player に一定の距離近づいてから
73                                                  if(Math.sqrt(Math.pow(player.getX()-
chara.get(i).getX(),2)+Math.pow(player.getY()-chara.get(i).getY(),2))<500){
74                          chara.get(i).update(field);
75                      }
76                  }
77                  //player の位置を伝える
78                  chara.get(i).setPlayerLocate(player.getX(), player.getY());
79                  //attackFlag のチェック
80                  enemyAttackFlagCheck(chara.get(i));
81                  //hp が 0 以下でリストから外す
82                  endCheck(chara.get(i));
83              }
84
85              //character 同市のあたり判定
86              for(int i=0;i<chara.size();i++){
87                  for(int j=i+1;j<chara.size();j++){
88                      collisionCheack(chara.get(i), chara.get(j));
89                  }
90              }
91              //前フレームの座標を保存　使っていない
92              for(Character f:chara){
93                  f.pX = f.getX();
94                  f.pY = f.getY();
95              }
96
97              // if(ccc%100==0){
98              //      soundManager.stop("shoot");
99              //      soundManager.play("shoot");
100             // }
101             // ccc++;
102
103             //e1.update(field);
104             gameOverCheck();
105         }
106         //controller で呼び出し
107         public void move(int d, boolean f){
```

```
108             if(d==-1){
109                 pressedKeyLeft = f;
110             }
111             if(d==1){
112                 pressedKeyRight = f;
113             }
114         }
115         //controller で呼び出し
116         public void jump(){
117             if(player!=null){
118                 player.jump();
119                 //shoot();
120             }
121         }
122         //player ball 攻撃  controller で呼び出し
123         public void shoot(){
124             Ball ball;
125             soundManager.stop("shoot");
126             soundManager.play("shoot");
127             if(player.dir==0){
128                     ball = new Ball((int)player.getX()+player.getWidth()+6,
(int)player.getY()+player.getHeight()/4);
129                 ball.dir = 0;
130             }else{
131                 ball = new Ball((int)player.getX()-10, (int)player.getY());
132                 ball.vx = -1.5f;
133                 ball.dir = 1;
134             }
135             chara.add(ball);
136         }
137
138         private void enemyAttackFlagCheck(Character c){
139             //Boss
140             if(c.getCharacterNum()==10){
141                 if(c.getAttackFlag()){
142                                         float vx = (float)((player.getX()-
c.getX())/(Math.sqrt(Math.pow(player.getX()-c.getX(),2)+Math.pow(player.getY()-
c.getY(),2))));
```

```
143                                                 float vy = (float)((player.getY()-
c.getY())/(Math.sqrt(Math.pow(player.getX()-c.getX(),2)+Math.pow(player.getY()-
c.getY(),2))));
144                         if(c.dir==1){
145                             BossFire fire = new BossFire((int)c.getX()-6, (int)c.getY()-
20,vx,vy);
146                             c.attacked();
147                             chara.add(fire);
148                         }else{
149                                 BossFire fire = new BossFire((int)c.getX()+c.gw-48,
(int)c.getY()-20,vx,vy);
150                             c.attacked();
151                             chara.add(fire);
152                         }
153
154                     }
155                 }
156
157                 //(1/28 追加)
158                 //Enemy4
159                 if(c.getCharacterNum()==2){
160                     if(c.getAttackFlag()){
161                                 EnemyBall eball = new EnemyBall((int)c.getX(),
(int)c.getY());
162                         if(player.getX() > c.getX()){
163                             c.dir = 0;
164                             eball.EBdir(0);
165                         }
166                         else{
167                             c.dir = 1;
168                             eball.EBdir(1);
169                         }
170                         c.attacked();
171                         chara.add(eball);
172                     }
173                 }
174             }
175             //ball を消す
```

```
176         private void endCheck(Character c){
177             if( c.getCharacterNum()==1 || c.getCharacterNum()==2 ){
178                 if(c.hp<=0){
179                     chara.remove(chara.indexOf(c));
180                     score += 100;
181                 }
182                 }else  if(c.getCharacterNum()==97  ||  c.getCharacterNum()==98
||c.getCharacterNum()==99){
183                 if(c.hp<=0){
184                     chara.remove(chara.indexOf(c));
185                 }
186             }else if(c.getCharacterNum()==10){
187                 if(c.hp<=0){
188                     chara.remove(chara.indexOf(c));
189                     score += 10000;
190                     stageClear = true;
191                 }
192             }else if(c.getCharacterNum()==6){
193                 if(c.hp<=0){
194                     chara.remove(chara.indexOf(c));
195                     score += 200;
196                     System.out.println("coin");
197                 }
198             }
199
200         }
201         //キャラ同市のあたり判定
202         private void collisionCheack(Character c1, Character c2){
203             //矩形あたり判定
204             if(Math.abs(c1.getX()-c2.getX())<(c1.gw+c2.gw)/2){
205                 if(Math.abs(c1.getY()-c2.getY())<(c1.gh+c2.gh)/2){
206                     if(c1.getCharacterNum()==0){//player と enemy のあたり判定
207                         if(c1.getDamageCount()==0){//無敵でないなら
208                             if(c2.getCharacterNum()==99){//ball との判定はない
209                                 return;
210                             }
211                             if(c2.getCharacterNum()==6){//coin との判定
212                                 c2.damaged(1);
```

```
213                     soundManager.stop("coin");
214                     soundManager.play("coin");
215                     return;
216                 }
217                 if(c1.pY+c1.getHeight()<c2.getY()){//上から
218                     //player.jump();
219                     soundManager.stop("stomp");
220                     soundManager.play("stomp");
221                     c2.damaged(1);//enemy にダメージ
222                     c1.setVy(-3);
223                 }else{
224                     c1.damaged(1);//player にダメージ
225                 }
226                                                 //  if(Math.abs(c1.pX-
c2.pX)>(c1.getWidth()+c2.getWidth())/2){//横から
227                 //      c1.damaged(1);//player にダメージ
228                                                 //  }else  if(c1.getY()-
c2.getY()<(c1.getHeight()+c2.getHeight())/2){//下から
229                 //      c1.damaged(1);
230                 //}
231             }
232                     }else  if((c1.getCharacterNum()==98  &&
c2.getCharacterNum()==10)       ||(c1.getCharacterNum()==10       &&
c2.getCharacterNum()==98)){
233             return;
234                     }else  if(c1.getCharacterNum()==6  ||
c2.getCharacterNum()==6){
235             //return;
236
237                     }else  if(c1.getCharacterNum()==99  ||
c2.getCharacterNum()==99){
238             c1.damaged(1);
239             c2.damaged(1);
240                     }else  if((c1.getCharacterNum()==98  &&
c2.getCharacterNum()==99)       ||(c1.getCharacterNum()==99       &&
c2.getCharacterNum()==98)){
241             if(c1.getCharacterNum()==99){
242                 c1.damaged(1);
```

```
243                         }else if(c2.getCharacterNum()==99){
244                             c2.damaged(1);
245                         }
246                                         }else   if((c1.getCharacterNum()==99   &&
c2.getCharacterNum()==97)          ||          (c1.getCharacterNum()==97          &&
c2.getCharacterNum()==99) ){
247                         //(1/28)EnemyBall 用の Num97 を追加
248                         c1.damaged(1);
249                         c2.damaged(1);
250                     }
251                 }
252             }
253         }
254
255
256         //player の hp がゼロになったかをチェックする
257         private void gameOverCheck(){
258             if(player.getHp()<=0){
259                 gameOver = true;
260             }
261             if(player.getX()<0||player.getX()>field.WIDTH){
262                 gameOver = true;
263             }
264             if(player.getY()<0||player.getY()>field.HEIGHT){
265                 gameOver = true;
266             }
267         }
268         //音ファイル読み込み
269         private void loadSound(){
270             if(!isSoundLoad){
271                 for(int i=0;i<soundNames.length;i++){
272                                             soundManager.load(soundNames[i],
"sounds/"+soundNames[i] + ".wav");
273                 }
274             }
275             isSoundLoad = true;
276         }
277
```

```
278
279         //get.set 関数
280         public ArrayList<Character> getCharactors(){
281             return chara;
282         }
283         public void setCharacter(Character c){
284             c.setSoundManager(soundManager);
285             chara.add(c);
286         }
287         public Character getCharactor(int idx){
288             return chara.get(idx);
289         }
290
291         public int getScore(){
292             return score;
293         }
294
295         public int getPlayerHp(){
296             return player.getHp();
297         }
298
299         public void soundPlay(String name){
300             soundManager.stop(name);
301             soundManager.play(name);
302         }
303
304         public int getStageNum(){
305             return stageNum;
306         }
307
308 }
309
```

8.2. SoundManager.java

```
1  import java.io.IOException;
2  import java.util.HashMap;
3  import javax.sound.sampled.AudioFormat;
4  import javax.sound.sampled.AudioInputStream;
```

```
 5  import javax.sound.sampled.AudioSystem;
 6  import javax.sound.sampled.Clip;
 7  import javax.sound.sampled.DataLine;
 8  import javax.sound.sampled.LineEvent;
 9  import javax.sound.sampled.LineListener;
10  import javax.sound.sampled.LineUnavailableException;
11  import javax.sound.sampled.UnsupportedAudioFileException;
12          // 音ファイル読み込み https://aidiary.hatenablog.com/entry/20061105/1275137770
13  public class SoundManager implements LineListener{
14      private int maxClips;
15      private HashMap<String, Clip> clipMap;
16      private int counter = 0;
17
18
19      SoundManager(){
20          this(256);
21      }
22      SoundManager(int maxClips){
23          this.maxClips = maxClips;
24          clipMap = new HashMap<String, Clip>(maxClips);
25      }
26
27      public void load(String name, String filename) {
28          if (counter == maxClips) {
29              System.out.println("hash is filled");
30              return;
31          }
32
33          try {
34                                  AudioInputStream stream = AudioSystem.getAudioInputStream(getClass().getResource(filename));
35              AudioFormat format = stream.getFormat();
36              if ((format.getEncoding() == AudioFormat.Encoding.ULAW)
37                                  || (format.getEncoding() == AudioFormat.Encoding.ALAW)) {
38                  AudioFormat newFormat = new AudioFormat(
39                          AudioFormat.Encoding.PCM_SIGNED,
```

```
40                                  format.getSampleRate(),
41                                                  format.getSampleSizeInBits() * 2,
format.getChannels(),
42                                  format.getFrameSize() * 2, format.getFrameRate(),
true);
43                          stream = AudioSystem.getAudioInputStream(newFormat,
stream);
44                      format = newFormat;
45                  }
46
47                  DataLine.Info info = new DataLine.Info(Clip.class, format);
48                  if (!AudioSystem.isLineSupported(info)) {
49                      System.out.println("SoundLoadError: "+filename + "does not
exit");
50                      System.exit(0);
51                  }
52
53                  Clip clip = (Clip) AudioSystem.getLine(info);
54                  clip.addLineListener(this);
55                  clip.open(stream);
56                  clipMap.put(name, clip);
57                  stream.close();
58              } catch (UnsupportedAudioFileException e) {
59                  e.printStackTrace();
60              } catch (IOException e) {
61                  e.printStackTrace();
62              } catch (LineUnavailableException e) {
63                  e.printStackTrace();
64              }
65          }
66
67
68          public void play(String name) {
69              Clip clip = clipMap.get(name);
70              //同じ音のときは途中で止めて再生する
71              if (clip != null) {
72
73                  clip.setFramePosition(0);
```

```
74                  clip.start();
75              }
76          }
77
78          public void loop(String name) {
79              Clip clip = clipMap.get(name);
80              //同じ音のときは途中で止めて再生する
81              if (clip != null) {
82
83                  clip.setFramePosition(0);
84                  clip.loop(20);
85              }
86          }
87
88
89          public void stop(String name){
90              Clip clip = clipMap.get(name);
91              if (clip != null) {
92                  clip.stop();
93                  clip.setFramePosition(0);
94                  //clip.start();
95              }
96          }
97
98
99
100         public void update(LineEvent event) {
101             if (event.getType() == LineEvent.Type.STOP) {
102                 Clip clip = (Clip) event.getSource();
103                 clip.stop();
104                 clip.setFramePosition(0);
105
106             }
107         }
108 }
```

8.3. Field.java

```
1  import javax.swing.*;
```

```
 2  import java.awt.*;
 3  import java.awt.event.*;
 4  import java.util.*;
 5  import java.io.*;
 6
 7  public class Field {
 8      private int ROW, COL;
 9      private boolean is83;//使っていない
10      public int WIDTH,HEIGHT;
11      private ImageIcon icon;
12      private Image i13,i40,i117,i194,i195;
13      private int IMAGESIZE = 18,bi,bj;//bi、bj はクリア演出用　今は使っていない
14      private int offsetX,offsetY, count;//count はクリア演出用　今は使っていない
15      private Model model;
16      public Field(Model model,int i){
17          init(model, i);
18      }
19      //初期化用
20      public void init(Model model, int i){
21          if(i==0){
22              loadField("map1.csv");
23          }else if(i==1){
24              loadField("map3.csv");
25          }
26          icon = new ImageIcon(getClass().getResource("map/13.png"));
27          i13 = icon.getImage();
28          icon = new ImageIcon(getClass().getResource("map/40.png"));
29          i40 = icon.getImage();
30          icon = new ImageIcon(getClass().getResource("map/194.png"));
31          i194 = icon.getImage();
32          icon = new ImageIcon(getClass().getResource("map/117.png"));
33          i117 = icon.getImage();
34          icon = new ImageIcon(getClass().getResource("map/195.png"));
35          i195 = icon.getImage();
36          offsetX = 0;
37          offsetY = 0;
38          is83 = false;
```

```
39              this.model = model;
40              count = 0;
41              bi=29;
42              bj=37;
43              //goal = false;
44              //model.createPlayer(100, 250);
45              charaSet();
46          }
47          //view で毎フレーム呼ばれる
48          public void update(Dimension size){
49              updateOffset(size);
50              //bossFinish();
51          }
52          //offset のアップデート
53          private void updateOffset(Dimension size){
54              offsetX = size.width/2-(int)model.player.getX();
55              offsetY = size.height/2-(int)model.player.getY();
56              offsetX = Math.min(offsetX, 0);
57              offsetX = Math.max(offsetX, size.width - WIDTH);
58              offsetY = Math.min(offsetY, 0);
59              offsetY = Math.max(offsetY, size.height - HEIGHT);
60          }
61          //描画処理
62          public void draw(Graphics g){
63              for(int i=0;i<ROW;i++){
64                  for(int j=0;j<COL;j++){
65                      switch(map[i][j]){
66                          case 1://block
67                              g.setColor(Color.BLACK);
68                               g.fillRect(j*(int)cs+offsetX, i*(int)cs+offsetY, (int)cs,
(int)cs);
69                              break;
70                          case 2:
71                              g.setColor(Color.ORANGE);
72                               g.fillRect(j*(int)cs+offsetX, i*(int)cs+offsetY, (int)cs,
(int)cs);
73                              break;
74                          case 13:
```

```
75                              g.setColor(Color.ORANGE);
76                                  g.fillRect(j*(int)cs+offsetX, i*(int)cs+offsetY, (int)cs,
(int)cs);
77                              g.drawImage(i13, j*(int)cs+offsetX, i*(int)cs+offsetY,
j*(int)cs+offsetX+(int)cs, i*(int)cs+offsetY+(int)cs,0, 0, IMAGESIZE,IMAGESIZE, null);
78                              break;
79                          case 40:
80                              g.drawImage(i40, j*(int)cs+offsetX, i*(int)cs+offsetY,
j*(int)cs+offsetX+(int)cs, i*(int)cs+offsetY+(int)cs,0, 0, IMAGESIZE,IMAGESIZE, null);
81                              break;
82                          case 117:
83                              g.drawImage(i117, j*(int)cs+offsetX, i*(int)cs+offsetY,
j*(int)cs+offsetX+(int)cs, i*(int)cs+offsetY+(int)cs,0, 0, IMAGESIZE,IMAGESIZE, null);
84                              break;
85                          case 194:
86                              g.drawImage(i194, j*(int)cs+offsetX, i*(int)cs+offsetY,
j*(int)cs+offsetX+(int)cs, i*(int)cs+offsetY+(int)cs,0, 0, IMAGESIZE,IMAGESIZE, null);
87                              break;
88                          case 195:
89                              g.drawImage(i195, j*(int)cs+offsetX, i*(int)cs+offsetY,
j*(int)cs+offsetX+(int)cs, i*(int)cs+offsetY+(int)cs,0, 0, IMAGESIZE,IMAGESIZE, null);
90                              break;
91                          case 90:
92                              g.drawImage(i117, j*(int)cs+offsetX, i*(int)cs+offsetY,
j*(int)cs+offsetX+(int)cs, i*(int)cs+offsetY+(int)cs,0, 0, IMAGESIZE,IMAGESIZE, null);
93                              break;
94
95                      }
96                  }
97              }
98          }
99          //map とキャラのあたり判定用
100         public boolean collisionCheck(int i, int j, Character c){
101             float bx=j*cs;
102             float by=i*cs;
103             if(Math.abs(c.getX()-bx)<(cs+c.getWidth())/2f){
104                 if(Math.abs(c.getY()-by)<(cs+c.getHeight())/2f){
105                     if(map[i][j]>0){
```

```
106                         if(map[i][j]==117){
107                             if(c.getCharacterNum()==0){
108                                 model.goal = true;
109                                 model.soundPlay("enter");
110                             }
111                         }
112                         if(map[i][j]==90){//使っていない
113                             if(c.getCharacterNum()==0){
114                                 model.bossFlag = true;
115                                 return false;
116                             }
117                         }
118
119                         return true;
120                     }
121
122                 }
123             }
124
125             return false;
126         }
127         //csv ファイル読み込み
128         private void loadField(String fileName){
129             try{
130                                 BufferedReader br = new BufferedReader(new
InputStreamReader(getClass().getResourceAsStream("map/"+fileName)));
131                 String line = br.readLine();
132                 ROW = Integer.parseInt(line);
133                 line = br.readLine();
134                 COL = Integer.parseInt(line);
135                 System.out.println(ROW+", "+COL);
136                 map = new int[ROW][COL];
137                 WIDTH = (int)cs*COL;
138                 HEIGHT = (int)cs*ROW;
139
140                 String col[];
141                 for(int i = 0;i<ROW;i++){
142                     line = br.readLine();
```

```
143                         col = line.split(",");
144                         for(int j=0;j<COL;j++){
145                             map[i][j] = Integer.parseInt(col[j]);
146                         }
147                     }
148                     // for(int i = 0;i<ROW;i++){
149                     //     for(int j=0;j<COL;j++){
150                     //         System.out.print(map[i][j]+",");
151                     //     }
152                     //     System.out.println();
153                     // }
154                 }catch(Exception e){
155                     e.printStackTrace();
156                 }
157 
158         }
159         //map を元にキャラを設置
160         private void charaSet(){
161             for(int i=0;i<ROW;i++){
162                 for(int j=0;j<COL;j++){
163                     if(map[i][j]==0){
164                         model.createPlayer((int)getBlockX(i, j), (int)getBlockY(i,
j));
165                     }else if(map[i][j]==-2){
166                         model.setCharacter(new Enemy1((int)getBlockX(i, j),
(int)getBlockY(i, j)));
167                     }else if(map[i][j]==-9){
168                         model.setCharacter(new Boss((int)getBlockX(i, j),
(int)getBlockY(i, j)));
169                     }else if(map[i][j]==-3){
170                         model.setCharacter(new Enemy3((int)getBlockX(i, j),
(int)getBlockY(i, j)));
171                     }else if(map[i][j]==-4){
172                         model.setCharacter(new Enemy4((int)getBlockX(i, j),
(int)getBlockY(i, j)));
173                     }else if(map[i][j]==-6){
174                         model.setCharacter(new Coin((int)getBlockX(i, j),
(int)getBlockY(i, j)));
```

```
175                     }
176                 }
177             }
178         }
179
180         private void bossFinish(){//使っていない
181             if(model.bossFlag){
182                 count++;
183                 if(count%10==0){
184                     map[bi][bj] = -1;
185                     model.soundPlay("block");
186                     bj--;
187                 }
188
189             }
190             if(count>300){
191                 model.stageClear = true;
192             }
193         }
194
195         //以下 get,set 関数
196         public int getRow(){return ROW;}
197         public int getCol(){return COL;}
198         public float getCs(){return cs;}
199         public int getOffsetX(){
200             return offsetX;
201         }
202         public int getOffsetY(){
203             return offsetY;
204         }
205         public void setNum(int i, int j, int n){
206             map[i][j] = n;
207         }
208         public int getNum(int i, int j){
209             return map[i][j];
210         }
211         public float getBlockX(int i, int j){
212             return j*cs;
```

```
213         }
214         public float getBlockY(int i, int j){
215               return i*cs;
216         }
217         // public boolean isHit(int i, int j){
218         //       if(map[i][j]>0){
219         //             return true;
220         //       }else{
221         //             return false;
222         //       }
223         // }
224
225   }
226
```

8.4. Character.java

```
1   import javax.swing.*;
2   import java.awt.*;
3   import java.awt.event.*;
4   import java.util.*;
5
6   public class Character {
7         protected int width, height;//field とのあたり判定用
8         protected int hp, LimitY, mutekiTime,characterNum;//characterNum:種類判別用のナンバー
9         protected float x, y, vx, vy, g;
10        protected Image image;
11                protected boolean isGround, isCollisionX, isCollisionY, isDamaged,attackFlag;
12        //animation 用の向き
13        protected static final int RIGHT = 0;
14        protected static final int LEFT = 1;
15        protected int dir;
16        //animationCount,count:アニメーション用,damageCount:damage 時の一定時間無敵用
17        protected int animationCount, count, damageCount;
18        //一フレーム前の座標保存用
19        public float pX, pY;//使ってない
```

```
20      public int gw,gh;
21      protected SoundManager soundManager;
22      protected float playerX, playerY;
23
24      public Character(int x, int y, int w, int h, int hp, int characterNum){
25          this.x = x;
26          this.y = y;
27          width = w;
28          height = h;
29          this.hp = hp;
30          vx = 0;
31          vy = 0;
32          g = 0.08f;
33          isCollisionX = false;
34          isCollisionY = false;
35          isGround = false;
36          animationCount = 0;
37          count = 0;
38          isDamaged = false;
39          damageCount = 0;
40          this.characterNum = characterNum;
41          attackFlag = false;
42          playerX = 0;
43          playerY = 0;
44
45      }
46
47
48      //model で毎フレーム呼び出し
49      public void update(Field field){
50          fall();
51          move(field);
52          if(damageCount>0){
53              damageCount--;
54          }
55      }
56      //field とのあたり判定を考慮した移動
57      public void move(Field field){
```

```
58              //System.out.println(vx);
59              collisionX(field);
60              collisionY(field);
61              //System.out.println(y);
62              //System.out.println(vy);
63          }
64          //キーボードの入力に対する速度と向き
65          public void moveX(int d){
66              if(d>0){
67                  dir = RIGHT;
68              }else if(d<0){
69                  dir = LEFT;
70              }
71              vx = d*2;
72          }
73
74          // public void fall(){
75          //      if(isGround!=true){
76          //          vy+=g;
77          //          y+=vy;
78          //      }else{
79          //          //vy=0;
80          //      }
81          // }
82          public void fall(){//重力
83              // if(!isGround){
84              //      vy+=g;
85              // }
86              vy+=g;
87          }
88
89          //ダメージを受けたときに呼び出し
90          public void damaged(int i){
91              if(damageCount==0){
92                  hp -= i;
93                  damageCount = 100;
94                  System.out.println("damage!"+hp);
95              }
```

```
96      }
97      //描画処理
98      public void draw(Graphics g, int offsetX, int offsetY){};
99      //x のあたり判定
100     public void collisionX(Field field){
101         float newx = x + vx;
102         for(int i=0;i<field.getRow();i++){
103             for(int j=0;j<field.getCol();j++){
104                 if(field.collisionCheck(i, j, this)){
105                     if(vx>0){
106                         isCollisionX = true;
107                         x = field.getBlockX(i, j)-width;
108                         vx = 0;
109                     }else if(vx<0){
110                         isCollisionX = true;
111                         x = field.getBlockX(i, j)+field.getCs();
112                         vx = 0;
113                     }
114                     return;
115                 }else{
116                     x = newx;
117                     isCollisionX = false;
118                 }
119             }
120         }
121         //x = newx;
122     }
123
124     //protected void loadImage(){}
125     //y のあたり判定
126     public void collisionY(Field field){
127         float newy = y + vy;
128         //boolean f=false;
129         for(int i=0;i<field.getRow();i++){
130             for(int j=0;j<field.getCol();j++){
131                 if(field.collisionCheck(i, j, this)){
132                     //if(field.getNum(i, j)>0){
133                         if(vy>0){
```

```
134                                     y = field.getBlockY(i, j)-height;
135                                     vy = 0;
136                                     isGround = true;
137                                     //System.out.println("true");
138                                 }else if(vy<0){
139                                     y = field.getBlockY(i, j)+field.getCs();
140                                     vy = 0;
141                                     if(field.getNum(i,j)==194){
142                                         field.setNum(i,j,-1);
143                                         soundManager.play("block");
144                                     }
145                                 }
146                                 //f=true;
147                                 return;
148                             //}else{
149                                 //y = newy;
150                                 //isGround = false;
151                             //}
152                         }else{
153                             y = newy;
154                             isGround = false;
155                         }
156                     }
157                 }
158                 //y = newy;
159                 //isGround = false;
160             }
161             //以下 getset 関数など
162             public float getX(){
163                 return x;
164             }
165             public float getY(){
166                 return y;
167             }
168             public int getWidth(){
169                 return width;
170             }
171             public int getHeight(){
```

```
172             return height;
173         }
174         public boolean getIsDamaged(){
175             return isDamaged;
176         }
177         public int getHp(){
178             return hp;
179         }
180         public float getVx(){
181             return vx;
182         }
183         public float getVy(){
184             return vy;
185         }
186         public void setVy(float f){
187             vy = f;
188         }
189         public int getCharacterNum(){
190             return characterNum;
191         }
192         public boolean getAttackFlag(){
193             return attackFlag;
194         }
195         public void attacked(){
196             attackFlag = false;
197         }
198
199         //今使ってないやつ
200         public int getDamageCount(){
201             return damageCount;
202         }
203         public void setLocation(int x, int y){
204             this.x = x;
205             this.y = y;
206         }
207         public void cheackIsGround(int limitY){
208             LimitY = limitY;
209             if(y>=limitY){
```

```
210                 isGround = true;
211             }else{
212                 isGround = false;
213             }
214         }
215         public void checkIsGround(boolean f){
216             isGround = f;
217         }
218         public void checkIsCollisionX(boolean f){
219             isCollisionX = f;
220             if(f){
221                 vx=0;
222             }
223         }
224         public void checkIsCollisionY(boolean f){
225             isCollisionY = f;
226             if(f){
227                 vy=0;
228             }
229         }
230         public void setSoundManager(SoundManager soundManager){
231             this.soundManager = soundManager;
232         }
233
234         public void setPlayerLocate(float x, float y){
235             playerX = x;
236             playerY = y;
237         }
238 }
```

8.5. Player.java

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.util.*;
5  import javax.imageio.ImageIO;
6  import java.awt.image.BufferedImage;
7  import java.io.*;
```

```
 8
 9  public class Player extends Character{
10      private static   final int IMAGESIZE = 32;
11      private ImageIcon icon;
12      private Boolean isShootBall;//使っていない
13      public Player(int x, int y){
14          super(x, y, 32, 32, 3, 0);
15          gw = gh = 32;
16          isShootBall = false;
17      }
18      //model で呼び出し
19      public void jump(){
20          System.out.println(isGround);
21          if(isGround){
22              vy = -5f;
23              soundManager.play("jump");
24              System.out.println(vy);
25              isGround = false;
26          }
27      }
28      //描画処理
29      @Override
30      public void draw(Graphics g, int offsetX, int offsetY){
31          icon = new ImageIcon(getClass().getResource("pictures/1player.png"));
32          image = icon.getImage();
33          //g.drawRect((int)x +offsetX, (int)y+offsetY, width, height);
34          if(damageCount%50>25){
35              return;
36          }
37                  g.drawImage(image, (int)x +offsetX, (int)y+offsetY, (int)x
+offsetX+(int)width+3,
(int)y+offsetY+(int)height+3,IMAGESIZE+animationCount*(int)width+dir*IMAGESIZ
E*2+2, 0, IMAGESIZE+animationCount*(int)width+(int)width+dir*IMAGESIZE*2-
2,IMAGESIZE, null);
38          if(Math.abs(vx)>0.1f){
39              count++;
40          }
41          if(count>=40){
```

```
42                  count = 0;
43              }
44              if(count<20){
45                  animationCount = 0;
46              }else if(count<40){
47                  animationCount = 1;
48              }
49
50      }
51      @Override
52      public void damaged(int i){
53          if(damageCount==0){
54              hp -= i;
55              damageCount = 200;
56              System.out.println("damage!"+hp);
57          }
58      }
59
60      // public void damaged(){
61      //      if(mutekiTime==0){
62      //          mutekiTime = 100;
63      //          if(hp>0)hp--;
64      //      }
65
66      // }
67
68  }
69
```

8.6. Ball.java

```
1   import javax.swing.*;
2   import java.awt.*;
3   import java.awt.event.*;
4   import java.util.*;
5   public class Ball extends Character{
6       Ball(int x, int y){
7           super(x, y, 32, 32, 1,99);
8           fallcount = 0;
```

```
 9              g = 0.25f;
10              gw = 16;
11              gh = 16;
12          };
13          private int fallcount;
14          @Override
15          public void update(Field field){
16              super.update(field);
17              if(dir==0){
18                  vx = 4f;
19              }else{
20                  vx = -4f;
21              }
22              if(isGround){
23                  System.out.println(hp);
24                  vy += -3;
25                  fallcount+=1;
26                  if(fallcount>7){
27                      hp = 0;
28                  }
29              }
30              if(isCollisionX){
31                  hp = 0;
32              }
33          }
34
35          @Override
36          public void draw(Graphics g, int offsetX, int offsetY){
37              //System.out.println("ok");
38              g.setColor(Color.RED);
39                  g.fillOval((int)x+offsetX+width/4,  (int)y+offsetY+width/4,  width/2,
height/2);
40          }
41
```

8.7. Coin.java

```
 1  import javax.swing.*;
 2  import java.awt.*;
```

```
3  import java.awt.event.*;
4  import java.util.*;
5  import java.io.*;
6
7  public class Coin extends Character{
8      private Image image[];
9      private ImageIcon icon;
10     Coin(int x, int y){
11         super(x, y, 32, 32, 1, 6);
12         gh = 32;
13         gw = 32;
14         g = 0;
15         image = new Image[9];
16         for(int i=0;i<9;i++){
17                                                    icon = new
ImageIcon(getClass().getResource("pictures/coin"+(i+1)+".png"));
18             image[i] = icon.getImage();
19         }
20         //System.out.println("coin");
21     }
22
23
24     @Override
25     public void draw(Graphics g, int offsetX, int offsetY){
26         count = (animationCount/10)%9;
27         //g.fillRect((int)x +offsetX, (int)y+offsetY, width, height);
28         g.drawImage(image[count], (int)x +offsetX, (int)y+offsetY, gw, gh,null);
29         animationCount++;
30     }
31
32
33 }
```

8.8. Boss.java

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.util.*;
```

```
5  public class Boss extends Character{
6      private int moveCounter, i, j;
7      private ImageIcon icon;
8      private Image move0, move1, move2,move3,move4,move5,move0r, move1r,
move2r,move3r,move4r,move5r;
9      private boolean isFly;//浮遊しているか
10     private float ppx, ppy;//行動の的
11     private Random random;
12     public Boss(int x,int y){
13         super(x, y, 32, 32, 25, 10);
14         gw = 150;
15         gh = 120;
16         moveCounter = 0;
17         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 000.png"));
18         move0 = icon.getImage();
19         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 001.png"));
20         move1 = icon.getImage();
21         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 002.png"));
22         move2 = icon.getImage();
23         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 003.png"));
24         move3 = icon.getImage();
25         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 004.png"));
26         move4 = icon.getImage();
27         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 005.png"));
28         move5 = icon.getImage();
29         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 000r.png"));
30         move0r = icon.getImage();
31         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 001r.png"));
32         move1r = icon.getImage();
33         icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
```

```
動 002r.png"));
    34              move2r = icon.getImage();
    35              icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 003r.png"));
    36              move3r = icon.getImage();
    37              icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 004r.png"));
    38              move4r = icon.getImage();
    39              icon = new ImageIcon(getClass().getResource("pictures/ドラゴン A_移
動 005r.png"));
    40              move5r = icon.getImage();
    41              attackFlag = false;
    42              setPlayer();
    43              g = 0.15f;
    44              i = 0;
    45              j =0 ;
    46              random = new Random();
    47          }
    48
    49          @Override
    50          public void draw(Graphics g, int offsetX, int offsetY){
    51              if(dir==LEFT){
    52                  if(animationCount<10){
    53                          g.drawImage(move0, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
    54                  }else if(animationCount<20){
    55                          g.drawImage(move1, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
    56                  }else if(animationCount<30){
    57                          g.drawImage(move2, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
    58                  }else if(animationCount<40){
    59                          g.drawImage(move3, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
    60                  }else if(animationCount<50){
    61                          g.drawImage(move4, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
    62                  }else if(animationCount<60){
```

```
63                          g.drawImage(move5, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
64                  }
65              }else if(dir==RIGHT){
66                  if(animationCount<10){
67                          g.drawImage(move0r, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
68                  }else if(animationCount<20){
69                          g.drawImage(move1r, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
70                  }else if(animationCount<30){
71                          g.drawImage(move2r, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
72                  }else if(animationCount<40){
73                          g.drawImage(move3r, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
74                  }else if(animationCount<50){
75                          g.drawImage(move4r, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
76                  }else if(animationCount<60){
77                          g.drawImage(move5r, (int)x +offsetX-60, (int)y+offsetY-85,
gw+34, gh,null);
78                  }
79              }
80              if(Math.abs(vx)>=0.5f){
81                  animationCount++;
82              }
83              if(animationCount>=60){
84                  animationCount = 0;
85              }
86          }
87          @Override
88          public void update(Field field){
89              super.update(field);
90              bossMove();
91          }
92
93          private void bossMove(){
```

```
94              fly();
95              if(moveCounter==0){
96                  setPlayer();
97                  changeDir();
98                  i = random.nextInt(5);
99                  if(i==4){
100                     i=0;
101                 }else if(i==5){
102                     i = 1;
103                 }
104             }
105             if(moveCounter==0){
106                 //System.out.println(vx+" "+ vy);
107                 if(i==0){
108                     System.out.println("attack!");
109                 }else if(i==1){
110                     tackle();
111                 }else if(i==2){
112                     moveHorizon();
113                 }
114                 j = random.nextInt(200);
115             }
116             if(moveCounter<j+400){
117                 if(i==0){
118                     attack();
119                 }
120             }
121             if(moveCounter==300){
122                 if(isCollisionX){
123                     moveCounter=0;
124                 }
125             }
126             if(moveCounter==j+400){
127                 changeDir();
128                 vx = 0;
129                 i = random.nextInt(3);
130                 if(i==0){
131                     if(!isFly){
```

```
132                         System.out.println("Bossfly!");
133                         jump();
134                         //isFly = true;
135                     }else{isFly = false;}
136                 }else{
137                     if(i==1){
138                         System.out.println("Bossjump");
139                         if(!isFly){
140                             jump();
141                         }
142 
143                     }
144                     isFly = false;
145                 }
146             }
147             if(moveCounter==550){
148                 if(i==0){
149                     isFly = true;
150                 }
151             }
152             moveCounter++;
153             if(moveCounter>j+400+100){
154                 moveCounter = 0;
155             }
156         }
157         private void jump(){
158             vy -= 8;
159         }
160 
161         private void fly(){
162             if(isFly){
163                 g = 0f;
164                 vy = 0;
165             }else{
166                 g = 0.15f;
167             }
168         }
169 
```

```
170         private void attack(){
171                         if(moveCounter%500==100   ||   moveCounter%500==150   ||
moveCounter%500==200 ||moveCounter%500==250 ||moveCounter%500==300){
172                     attackFlag = true;
173                     soundManager.stop("fire");
174                     soundManager.play("fire");
175                 }
176         }
177
178         private void setPlayer(){
179                 ppx = playerX;
180                 ppy = playerY;
181         }
182
183         private void tackle(){
184                 if(ppx>x){
185                     vx = 2.5f;
186
187                 }else{
188                     vx = -2.5f;
189
190                 }
191         }
192
193         private void moveHorizon(){
194                 if(ppx>x){
195                     vx = 1.5f;
196                 }else{
197                     vx = -1.5f;
198                 }
199         }
200
201         private void changeDir(){
202                 if(ppx>x){
203                     dir = RIGHT;
204                 }else{
205                     dir = LEFT;
206                 }
```

```
207         }
208
209
210 }
```

8.9. BossFire.java

```
 1 import javax.swing.*;
 2 import java.awt.*;
 3 import java.awt.event.*;
 4 import java.util.*;
 5 public class BossFire extends Character{
 6         private int count;
 7         private ImageIcon icon;
 8         private Image image;
 9         BossFire(int x, int y, float vx, float vy){
10                 super(x, y, 32, 32,3,98);
11                 characterNum = 98;
12                 this.vx = vx*2;
13                 this.vy = vy*2;
14                 gw = 24;
15                 gh = 24;
16                 count = 0;
17                 g=0;
18                 icon = new ImageIcon(getClass().getResource("pictures/bossfire.png"));
19                 image = icon.getImage();
20         }
21         @Override
22         public void update(Field field){
23                 super.update(field);
24                 if(isCollisionX || count>=800 || isCollisionY){
25                         hp = 0;
26                 }
27                 count++;
28         }
29
30         @Override
31         public void draw(Graphics g, int offsetX, int offsetY){
32                 //System.out.println("ok");
```

```
33              //g.setColor(Color.RED);
34              //g.fillOval((int)x+offsetX+width/4, (int)y+offsetY, gw, gh);
35                      g.drawImage(image, (int)x +offsetX, (int)y+offsetY, (int)x
+offsetX+(int)width, (int)y+offsetY+(int)height,0, 0, 1500,1500, null);
36          }
37  }
```

8.10. MainFrame.java

```
 1
 2  import javax.swing.*;
 3  import java.awt.*;
 4  import java.awt.event.*;
 5  import java.util.*;
 6  public class MainFrame extends JFrame{
 7          public MainFrame(){
 8              super("test");
 9              setDefaultCloseOperation(EXIT_ON_CLOSE);
10                      setSize(500,500);
11                      setLocationRelativeTo(null);
12              setResizable(false);
13              //pack();
14          }
15
16          public void change(JPanel panel){
17              getContentPane().removeAll();
18              super.add(panel);
19              validate();
20              repaint();
21              panel.setFocusable(true);
22              panel.requestFocusInWindow();
23          }
24
25          public static void main(String argv[]){
26              SceneManager sceneManager = new SceneManager();
27              sceneManager.changeScene();
28          }
29  }
30
```

8.11. GameView.java

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.util.*;
5  import java.awt.Font;
6  import java.awt.Image;
7  public class GameView extends JPanel implements Observer, ActionListener{
8     protected Model model;
9     protected CharaController c;
10    protected Field field;
11    private javax.swing.Timer timer;
12    private SceneManager sceneManager;
13    private int viewWidth, viewHeight;//使っていない
14    private Dimension size;
15    private SoundManager soundManager;
16      public GameView(Model m,SoundManager soundManager,SceneManager
sceneManager) {
17        // model = new Model();
18        // c = new CharaController(model);
19        model = m;
20        this.setBackground(Color.DARK_GRAY);
21        this.soundManager = soundManager;
22        //this.requestFocusInWindow();
23        setFocusable(true);
24        //this.requestFocus();
25        //this.setFocusable(true);
26        this.c = new CharaController(model);
27        addKeyListener(c);
28        timer = new javax.swing.Timer(10, this);
29        timer.start();
30        //model = m;
31        field = model.field;
32        this.sceneManager = sceneManager;
33        model.addObserver(this);
34        //System.out.println("a");
35        size= getSize();
```

```
36          if(model.getStageNum()==0){
37              soundManager.stop("field2");
38              soundManager.loop("field2");
39          }else if(model.getStageNum()==1){
40              soundManager.stop("boss");
41              soundManager.play("boss");
42          }
43      }
44      //描画 update 処理
45      public void paintComponent(Graphics g) {
46          super.paintComponent(g);
47
48              //スコア表示とHP表示は妹尾担当
49              //スコア表示
50              Font font = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,20);//Font型のローカル変数を定義し,Fontオブジェクトを生成しフォント指定
51                  g.setFont(font);//setFontメソッドを呼び出す
52                  g.setColor(Color.WHITE);//setColorメソッドを呼び出し色指定
53              int score = model.getScore();//int型のローカル変数を定義し,modelのgetScoreメソッドを呼ぶ
54              g.drawString("Score:"+score,30,70);//drawStringメソッドを呼び出し文字を描画
55
56              //HP表示
57
58              Font font2 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,20);
59                  g.setFont(font2);
60                  g.setColor(Color.RED);
61              int hp = model.getPlayerHp();//int型のローカル変数を定義し,modelのgetPlayerHpメソッドを呼ぶ
62              Image hart;//Image型の変数を宣言
63                  hart = Toolkit.getDefaultToolkit().getImage("pictures/hart.png");  //メソッドを呼び出し画像読み込み
64
65                  g.drawImage(hart,30,0,this);//drawImageメソッドを呼び出し描画
66          g.drawString(":"+hp,80,30);//drawString メソッドを呼び出し文字を描画
67
68
```

```
69
70          field.draw(g);
71          for(Character f:model.getCharactors()){
72              f.draw(g, field.getOffsetX(), field.getOffsetY());
73          }
74      }
75      public void update(Observable o,Object arg){//使っていない
76          repaint();
77      }
78      //timer の update 処理
79      public void actionPerformed(ActionEvent e){
80              size = getSize();
81              model.update();
82              field.update(size);
83              this.repaint();
84              if(model.gameOver){
85                  soundManager.stop("field2");
86                  soundManager.stop("boss");
87                  timer.stop();
88                  System.out.println("goal");
89                  //this.setFocusable(false);
90                  sceneManager.setSceneNum(4);
91                  sceneManager.changeScene();
92              }else if(model.goal){
93                  soundManager.stop("field2");
94                  soundManager.stop("boss");
95                  timer.stop();
96                  System.out.println("goal");
97                  //this.setFocusable(false);
98                  if(sceneManager.getSceneNum()==1){
99                      sceneManager.setSceneNum(2);
100                 }else if(sceneManager.getSceneNum()==2){
101                     soundManager.stop("field2");
102                 soundManager.stop("boss");
103                     sceneManager.setSceneNum(4);
104                 }
105
106                 sceneManager.changeScene();
```

```
107             }else if(model.stageClear){
108                 soundManager.stop("field2");
109                 soundManager.stop("boss");
110                 timer.stop();
111                 sceneManager.setSceneNum(5);
112                 sceneManager.changeScene();
113             }
114             //System.out.println("t");
115         }
116
117         //get,set 関数
118         public int getViewWidth(){
119             return viewWidth;
120         }
121         public int getViewHeight(){
122             return viewHeight;
123         }
124     }
125
```

8.12. SceneManager.java

```
1   import javax.swing.*; //swingをインポート
2   import java.awt.*; //AWTをインポート
3   import java.awt.event.*; //イベント処理用
4   import java.util.*;
5   public class SceneManager {
6       private int sceneNum;//intのscene番号を管理
7
8       private Model model;//Model型の変数を宣言
9
10      private MainFrame view;//MainFrame型の変数を宣言
11      private SoundManager soundManager;//soundManager型の変数を宣
言
12      SceneManager(){//SceneManagerのメソッド
13          soundManager = new SoundManager();//SoundManager型のロー
カル変数を定義し,SoundManagerオブジェクトを生成
14          model = new Model(0, soundManager);//model型のローカル変数
を定義し,Modelオブジェクトを生成
```

```
15
16              sceneNum = 0;
17              view = new MainFrame();//MainFrame型のローカル変数を定義
し,MainFrameオブジェクトを生成
18              view.setVisible(true);//見えるようになる
19          }
20          public void setSceneNum(int n){//Scene番号を振るメソッド
21              sceneNum = n;
22          }
23          public int getSceneNum(){//Scene番号を取得するメソッド
24              return sceneNum;
25          }
26          public void changeScene(){//Sceneを切り替えるメソッド
27              switch(sceneNum){//場合分け
28                  case 0://0の時
29                  view.change(new
TitleView1(soundManager,this));//TitleView1オブジェクトを生成してchangeメソッド
で切り替え
30                      break;
31                  case 1:
32                  model.init(0, soundManager);//modelのinitメソッドを呼び出
す
33                  view.change(new GameView(model,
soundManager,this));//TitleView1オブジェクトを生成してchangeメソッドで切り替え
34                      break;
35                  case 2:
36                  model.init(1, soundManager);//modelのinitメソッドを呼び出
す
37                  view.change(new GameView(model,
soundManager,this));//GameViewオブジェクトを生成してchangeメソッドで切り替え
38                      break;
39                  case 3:
40                  view.change(new TitleView(soundManager,this));//TitleViewオ
ブジェクトを生成してchangeメソッドで切り替え
41                      break;
42                  case 4:
43                      view.change(new GameOverView(soundManager,this,
model));//GameOverViewオブジェクトを生成してchangeメソッドで切り替え
```

```
44                          break;
45                      case 5:
46                          view.change(new GameClearView(soundManager,this,
model));//GameClearViewオブジェクトを生成してchangeメソッドで切り替え
47                  }
48          }
49      }
```

8.13. TitleView.java

```
1       import javax.swing.*; //swingをインポート
2       import java.awt.*; //AWTをインポート
3       import java.awt.event.*; //イベント処理用
4       import java.util.*;
5       import java.awt.Font; //文字のフォント用
6       public class TitleView extends JPanel implements KeyListener{ //JPanelを
継承しKeyListenerを実装したクラス
7
8           SceneManager sceneManager; //SceneManager型の変数を宣言
9           private SoundManager soundManager;//SoundManager型の変数を宣
言
10          TitleView(SoundManager soundManager，SceneManager
sceneManager){ //TitleViewのメソッド(引数はSoundManager型，SceneManager型の
変数)
11              this.sceneManager = sceneManager; //インスタンス変数にローカ
ル変数を代入
12              this.soundManager = soundManager;//インスタンス変数にローカ
ル変数を代入
13              this.addKeyListener(this); //キーリスナーを登録
14              this.setFocusable(true); //JPanelがキーボードを受け付けるよう
にする
15
16          }
17          public void paintComponent(Graphics g) { //ライブラリ側から自動的に
呼び出されるメソッド，Graphicsオブジェクトのメソッドを使って図形を描画
18              super.paintComponent(g); //上位クラスのpaintComponentを呼
び出し
19              g.setColor(Color.BLACK); //setColorメソッドを呼び出し色指定
20              g.fillRect(0,0,500,700); //fillRectメソッドを呼び出しパネルを描
```

画
```
21                  Font font = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,30); //Font
```
型のローカル変数を定義し,Fontオブジェクトを生成しフォント指定
```
22                  g.setFont(font); //setFontメソッドを呼び出す
23                  g.setColor(Color.WHITE);//setColorメソッドを呼び出し色指定
24                  g.drawString("Dキーで進む", 50, 100);//drawStringメソッドを呼び
```
出し文字を描画，以下Font6まで同様
```
25                  Font font2 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,30);
26                  g.setFont(font2);
27                  g.setColor(Color.WHITE);
28                  g.drawString("Aキーで戻る", 50, 130);
29                  Font font3 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,30);
30                  g.setFont(font3);
31                  g.setColor(Color.WHITE);
32                  g.drawString("Sキーで攻撃", 50, 160);
33                  Font font4 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,30);
34                  g.setFont(font4);
35                  g.setColor(Color.WHITE);
36                  g.drawString("SPACEでジャンプ", 50, 190);
37                  Font font5 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,30);
38                  g.setFont(font5);
39                  g.setColor(Color.WHITE);
40                  g.drawString("敵を倒してゴールを目指せ！", 50, 220);
41                  Font font6 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,30);
42                  g.setFont(font6);
43                  g.setColor(Color.WHITE);
44                  g.drawString("SPACEで次へ", 150, 350);
45
46
47          }
48          public void changeScene(){ //画面切り替え用のchangeSceneメソッド
49              soundManager.stop("title");//メソッドを呼び出してBGM切り替え
50              soundManager.play("decide");
51              sceneManager.setSceneNum(1); //setSceneNumメソッドを呼び
```
出し，指定されてある番号の画面を表示
```
52              sceneManager.changeScene(); //changeSceneメソッドを呼び出
```
し画面切り替え
```
53          }
```

```
54          public void keyTyped(KeyEvent e){
55
56          }
57          public void keyReleased(KeyEvent e){ //keyReleaseメソッド
58
59
60              switch(e.getKeyCode()){ //場合分け
61
62                  case KeyEvent.VK_SPACE: //spaceキーが押された時
63                      changeScene();//changeSceneメソッドを呼び出し
64                      System.out.println("space");//spaceと出力
65                      break;
66                  }
67          }
68          public void keyPressed(KeyEvent e){
69          }
70      }
71      class TitleController implements KeyListener{//keyListenerを実装したクラ
ス
72          TitleView v; //TitleView型のローカル変数を宣言
73          TitleController(TitleView v){ //TitleControllerメソッド
74              this.v = v; //インスタンス変数にローカル変数を代入
75          }
76          public void keyTyped(KeyEvent e){}
77          public void keyReleased(KeyEvent e){
78              int n = e.getKeyCode();
79              switch(e.getKeyCode()){
80
81                  case KeyEvent.VK_SPACE:
82
83                      v.changeScene();
84                      System.out.println("space");
85                      break;
86                  }
87          }
88          public void keyPressed(KeyEvent e){
89          }
90      }
```

8.14. TitleView1.java

```
1   import javax.swing.*; //swingをインポート
2   import java.awt.*; //AWTをインポート
3   import java.awt.event.*; //イベント処理用
4   import java.util.*;
5   import java.awt.Color; //色指定
6   import java.awt.Font; //文字のフォント用
7   import java.awt.Graphics; //描画用
8   import java.awt.Image; //画像読み込み用
9
10  public class TitleView1 extends JPanel  implements KeyListener{ //JPanel
を継承しKeyListenerを実装したクラス
11
12
13
14      SceneManager sceneManager;//SceneManager型の変数を宣言
15      private SoundManager soundManager;//soundManager型の変数を宣
言
16      TitleView1(SoundManager soundManager,SceneManager
sceneManager){//TitleView１のメソッド(引数はSoundManager型，SceneManager型
の変数)
17          this.sceneManager = sceneManager;//インスタンス変数にローカ
ル変数を代入
18          this.soundManager = soundManager;//インスタンス変数にローカ
ル変数を代入
19          soundManager.play("title");//メソッドを呼び出しBGM切り替え
20          Thread thread = new Thread(){//Thread型のローカル変数を定義
し,Threadオブジェクトを生成．処理を並行して行う
21              @Override
22              public void run(){//runメソッドを定義(Override)
23                  while(true){//ループさせる
24                      try{
25                          sleep(50);//指定したミリ秒処理を止める
26                      }catch(InterruptedException e){//別のスレッドが現在の
スレッドに割り込んだ場合。この例外がスローされると、現在のスレッドの割り込み
ステータスはクリアされる
27
```

```
28                          }
29                          if(down){//dowmの時
30                             y+=10;//y座標を10ずらす
31                          }
32                          if(up){
33                            y-=10;
34                          }
35                          if(left){
36                            x-=10;
37                          }
38                          if(right){
39                            x+=10;
40                          }
41                          repaint();
42                        }
43                    }
44                  };
45                  thread.start();//開始
46
47
48              addKeyListener(this);//キーリスナーを登録
49
50          }
51
52          int x;
53          int y;
54          public void paintComponent(Graphics g) {
55                  super.paintComponent(g);//上位クラスのpaintComponentを呼
び出し
56                  g.setColor(Color.BLACK);//setColorメソッドを呼び出し色指定
57                  g.fillRect(0,0,500,700);//fillRectメソッドを呼び出しパネルを描
画
58                  Font font = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,70);//Font
型のローカル変数を定義し,Fontオブジェクトを生成しフォント指定
59                  g.setFont(font);//setFontメソッドを呼び出す
60                    g.setColor(//setColorメソッドを呼び出し色指定
61                      new Color(
62                  (int)(Math.random() * 256), //ランダムでRGB値を決定
```

```
63                  (int)(Math.random() * 256),
64                  (int)(Math.random() * 256)));
65                  g.drawString("横スクロール", 30, 100);//drawStringメソッドを呼び
出し文字を描画，以下Font３まで同様
66                  Font font2 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,70);
67                  g.setFont(font2);
68                    g.setColor(
69                      new Color(
70                          (int)(Math.random() * 256),
71                          (int)(Math.random() * 256),
72                          (int)(Math.random() * 256)));
73                  g.drawString("アクション", 150, 170);
74                  Font font3 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,20);
75                  g.setFont(font3);
76                    g.setColor(
77                      new Color(
78                          (int)(Math.random() * 256),
79                          (int)(Math.random() * 256),
80                          (int)(Math.random() * 256)));
81                  g.drawString("SPACEを押してスタート", 150, 350);
82
83                  Image hart;//Image型の変数を宣言
84                  hart = Toolkit.getDefaultToolkit().getImage("pictures/hart.png");   //
メソッドを呼び出し画像読み込み
85
86                  g.drawImage(hart,x,y,this);//drawImageメソッドを呼び出し描画
87          }
88          public void changeScene(){//画面切り替え用のchangeSceneメソッド
89              soundManager.play("decide");//メソッドを呼び出しBGM切り替え
90              soundManager.stop("title");
91              sceneManager.setSceneNum(3);//setSceneNumメソッドを呼び出
し，指定されてある番号の画面を表示
92              sceneManager.changeScene();//changeSceneメソッドを呼び出
し画面切り替え
93          }
94          public void keyTyped(KeyEvent e){
95
96          }
```

```
 97          boolean up; //boolean型の変数を宣言
 98          boolean down;
 99          boolean right;
100          boolean left;
101
102          public void keyReleased(KeyEvent e){
103
104
105             switch(e.getKeyCode()){
106
107                case KeyEvent.VK_SPACE: //spaceキーが押された時
108                     changeScene();//changeSceneメソッドを呼び出し
109                     System.out.println("space");//spaceと出力
110                     break;
111                case 37: //キーと対応した番号の時
112                     left = false;//falseとする
113                     break;
114                case 38:
115                     up = false;
116                     break;
117                case 39:
118                     right = false;
119                     break;
120                case 40:
121                     down = false;
122                     break;
123                }
124          }
125          public void keyPressed(KeyEvent e){
126             switch(e.getKeyCode()){
127
128                     case 37://キーと対応した番号の時
129                     left = true;//trueとする
130                     break;
131                    case 38:
132                     up = true;
133                     break;
134                    case 39:
```

```
135                     right = true;
136                     break;
137                 case 40:
138                     down = true;
139                     break;
140             }
141         }
142     }
143
144
```

8.15. GameOverView.java

```
1   import javax.swing.*; //swingをインポート
2   import java.awt.*; //AWTをインポート
3   import java.awt.event.*; //イベント処理用
4   import java.util.*;
5   import java.awt.Color; //文字の色指定
6   import java.awt.Font; //文字のフォント用
7   import java.awt.Graphics; //描画用
8   import java.awt.Image; //画像読み込み用
9
10  public class GameOverView extends JPanel  implements
KeyListener{ //JPanelを継承しKeyListenerを実装したクラス
11
12
13
14      SceneManager sceneManager; //SceneManager型の変数を宣言
15      private Model model; //Model型の変数を宣言
16      private SoundManager soundManager; //soundManager型の変数を宣
言
17
18      GameOverView(SoundManager soundManager,SceneManager
sceneManager,Model model){//TitleViewのメソッド(引数はsoundManager型.
SceneManager型,Model型の変数)
19          this.sceneManager = sceneManager;//インスタンス変数にローカ
ル変数を代入
20          this.model = model;//インスタンス変数にローカル変数を代入
21          this.soundManager = soundManager;//インスタンス変数にローカ
```

```
ル変数を代入
    22
    23                  Thread thread = new Thread(){//Thread型のローカル変数を定義し,Threadオブジェクトを生成．処理を並行して行う
    24                      @Override
    25                      public void run(){//runメソッドを定義(Override)
    26                          while(true){//ループさせる
    27                              try{
    28                                  sleep(50);//指定したミリ秒処理を止める
    29                              }catch(InterruptedException e){//別のスレッドが現在のスレッドに割り込んだ場合。この例外がスローされると、現在のスレッドの割り込みステータスはクリアされる
    30
    31                              }
    32
    33                              repaint();
    34                          }
    35                      }
    36                  };
    37                  thread.start();//開始
    38
    39
    40              addKeyListener(this);//キーリスナーを登録
    41              soundManager.stop("gameover");//メソッドを呼び出しBGM切り替え
    42              soundManager.play("gameover");
    43
    44          }
    45
    46
    47          public void paintComponent(Graphics g) {//ライブラリ側から自動的に呼び出されるメソッド，Graphicsオブジェクトのメソッドを使って図形を描画
    48              super.paintComponent(g);//上位クラスのpaintComponentを呼び出し
    49              g.setColor(Color.BLACK);//setColorメソッドを呼び出し色指定
    50              g.fillRect(0,0,500,700);//fillRectメソッドを呼び出しパネルを描画
    51              Font font = new Font("HGP創英角ﾎﾟｯﾌﾟ体
```

",Font.ITALIC,100);//Font型のローカル変数を定義し,Fontオブジェクトを生成しフォント指定

```
52                  g.setFont(font);//setFontメソッドを呼び出す
53                     g.setColor(//setColorメソッドを呼び出し色指定
54                        new Color(
55                  (int)(Math.random() * 256), 0, 0));
56                  g.drawString("GAME", 30, 130);//drawStringメソッドを呼び出し文字を描画，以下Font４まで同様
57                  Font font2 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,100);
58                  g.setFont(font2);
59                     g.setColor(
60                        new Color(
61                  (int)(Math.random() * 256), 0, 0));
62                  g.drawString("OVER", 180, 230);
63                  Font font3 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,50);
64                  g.setFont(font3);
65                     g.setColor(Color.BLUE);
66                  int score = model.getScore();
67                  g.drawString("Score:"+score,110,310);
68
69                  Font font4 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,20);
70                  g.setFont(font4);
71                     g.setColor(
72                        new Color(
73                              (int)(Math.random() * 256), //ランダムでRGB値を決定
74                              (int)(Math.random() * 256),
75                              (int)(Math.random() * 256)));
76                  g.drawString("SPACEを押してタイトルへ", 130, 350);
77
78
79            }
80            public void changeScene(){//画面切り替え用のchangeSceneメソッド
81                  soundManager.play("decide");//メソッドを呼び出しBGM切り替え
82                  sceneManager.setSceneNum(0);//setSceneNumメソッドを呼び出し，指定されてある番号の画面を表示
83                  sceneManager.changeScene();//changeSceneメソッドを呼び出し画面切り替え
84            }
```

```
85          public void keyTyped(KeyEvent e){
86
87          }
88
89
90          public void keyReleased(KeyEvent e){
91
92
93             switch(e.getKeyCode()){//場合分け
94
95                case KeyEvent.VK_SPACE://spaceキーが押された時
96                     changeScene();//changeSceneメソッドを呼び出し
97                     System.out.println("space");//spaceと出力
98                     break;
99
100               }
101         }
102         public void keyPressed(KeyEvent e){
103
104               }
105         }
106
107
108
```

8.16. GameClearView.java

```
1     import javax.swing.*; //swingをインポート
2     import java.awt.*; //AWTをインポート
3     import java.awt.event.*; //イベント処理用
4     import java.util.*;
5     import java.awt.Color; //文字の色指定
6     import java.awt.Font; //文字のフォント用
7     import java.awt.Graphics; //描画用
8     import java.awt.Image; //画像読み込み用
9
10    public class GameClearView extends JPanel   implements
KeyListener{ //JPanelを継承しKeyListenerを実装したクラス
11
```

```
12
13
14          SceneManager sceneManager;//SceneManager型の変数を宣言
15          private Model model;//Model型の変数を宣言
16          private SoundManager soundManager;//soundManager型の変数を宣言
17
18          GameClearView(SoundManager soundManager,SceneManager sceneManager,Model model){//GameClearViewのメソッド(引数はSoundManager型, SceneManager型の変数,Model型の変数)
19              this.sceneManager = sceneManager;//インスタンス変数にローカル変数を代入
20              this.model = model;//インスタンス変数にローカル変数を代入
21              this.soundManager = soundManager;//インスタンス変数にローカル変数を代入
22
23
24              Thread thread = new Thread(){//Thread型のローカル変数を定義し,Threadオブジェクトを生成．処理を並行して行う
25                  @Override
26                  public void run(){//runメソッドを定義(Override)
27                      while(true){//ループさせる
28                          try{
29                              sleep(30);//指定したミリ秒処理を止める
30                          }catch(InterruptedException e){//別のスレッドが現在のスレッドに割り込んだ場合。この例外がスローされると、現在のスレッドの割り込みステータスはクリアされる
31
32                          }
33
34                          repaint();
35                      }
36                  }
37              };
38              thread.start();//開始
39
40
41              addKeyListener(this);//キーリスナーを登録
```

```
42                soundManager.stop("gameclear");//メソッドを呼び出しBGM切り
替え
43                soundManager.play("gameclear");
44
45            }
46
47
48            public void paintComponent(Graphics g) {
49                super.paintComponent(g);//上位クラスのpaintComponentを呼
び出し
50                g.setColor(Color.BLACK);//setColorメソッドを呼び出し色指定
51                g.fillRect(0,0,500,700);//fillRectメソッドを呼び出しパネルを描
画
52                Font font = new Font("HGP創英角ﾎﾟｯﾌﾟ体
",Font.ITALIC,100);//Font型のローカル変数を定義し,Fontオブジェクトを生成しフォン
ト指定
53                g.setFont(font);//setFontメソッドを呼び出す
54                g.setColor(//setColorメソッドを呼び出し色指定
55                    new Color(
56                (int)(Math.random() * 256), 0, 0));
57                g.drawString("STAGE", 30, 130);//drawStringメソッドを呼び出し
文字を描画，以下Font４まで同様
58                Font font2 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,100);
59                g.setFont(font2);
60                g.setColor(
61                    new Color(
62                (int)(Math.random() * 256), 0, 0));
63                g.drawString("CLEAR!", 120, 230);
64                Font font3 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,50);
65                g.setFont(font3);
66                g.setColor(Color.BLUE);
67                int score = model.getScore();
68                g.drawString("Score:"+score,110,310);
69
70                Font font4 = new Font("HGP創英角ﾎﾟｯﾌﾟ体",Font.ITALIC,20);
71                g.setFont(font4);
72                g.setColor(
73                    new Color(
```

```
74                          (int)(Math.random() * 256), //ランダムでRGB値を決定
75                          (int)(Math.random() * 256),
76                          (int)(Math.random() * 256)));
77              g.drawString("SPACEを押してタイトルへ", 130, 350);
78
79
80          }
81          public void changeScene(){//画面切り替え用のchangeSceneメソッド
82
83              soundManager.play("decide");//メソッドを呼び出しBGM切り替え
84              sceneManager.setSceneNum(0);//setSceneNumメソッドを呼び出
し，指定されてある番号の画面を表示
85              sceneManager.changeScene();//changeSceneメソッドを呼び出
し画面切り替え
86          }
87          public void keyTyped(KeyEvent e){
88
89          }
90
91
92          public void keyReleased(KeyEvent e){
93
94
95            switch(e.getKeyCode()){
96
97              case KeyEvent.VK_SPACE://spaceキーが押された時
98                  changeScene();//changeSceneメソッドを呼び出し
99                  System.out.println("space");//spaceと出力
100                 break;
101
102             }
103         }
104         public void keyPressed(KeyEvent e){
105
106         }
```

8.17. CharaController.java

```
 1  import javax.swing.*;
 2  import java.awt.*;
 3  import java.awt.event.*;
 4  import java.util.*;
 5
 6  public class CharaController implements KeyListener {
 7      protected Model model;
 8      public CharaController(Model a) {
 9         model = a;
10      }
11
12      public void keyTyped(KeyEvent e){}
13      public void keyReleased(KeyEvent e){
14         switch(e.getKeyCode()){
15            case KeyEvent.VK_D:
16                  model.move(1, false);
17                  break;
18            case KeyEvent.VK_A:
19                  model.move(-1, false);
20                  break;
21            case KeyEvent.VK_S:
22                  model.shoot();
23                  break;
24            case KeyEvent.VK_SPACE:
25                  //System.out.println("jump!");
26                  model.jump();
27                  break;
28            }
29      }
30      public void keyPressed(KeyEvent e){
31         switch(e.getKeyCode()){
32            case KeyEvent.VK_D:
33                  model.move(1, true);
34                  break;
35            case KeyEvent.VK_A:
```

```
36                      model.move(-1, true);
37                      break;
38                  }
39          }
40  }
```

8.18. Enemy1.java

```
 1  import javax.swing.*;
 2  import java.awt.*;
 3  import java.awt.event.*;
 4  import java.util.*;
 5
 6  /*Ball クラスと異なるのは、軌道とキャラナンバーのみ*/
 7
 8
 9  public class EnemyBall extends Ball{
10          EnemyBall(int x, int y){
11                  super(x, y);
12                  changeCharaNum();
13                  fallcount = 0;
14                  this.dir = 1;
15                  this.gw = 16;
16                  this.gh = 16;
17          };
18
19          public void changeCharaNum(){
20                  this.characterNum = 97;
21          }
22
23          private int fallcount;
24          @Override
25          public void update(Field field){
26                  super.update(field);
27                  if(dir==0){
28                          vx = 2f;
```

```
29              }else{
30                  vx = -2f;
31              }
32              if(isGround){
33                  System.out.println(hp);
34                  vy += -2;
35                  fallcount+=1;
36                  if(fallcount>5){
37                      hp = 0;
38                  }
39              }
40          }
41
42          public void EBdir(int i){
43              this.dir = i;
44          }
45
46
47          @Override
48          public void draw(Graphics g, int offsetX, int offsetY){
49              super.draw(g, offsetX, offsetY);
50              g.setColor(Color.blue);
51              /*System.out.println("ok");
52              g.setColor(Color.blue);
53                  g.fillOval((int)x+offsetX+width/4,  (int)y+offsetY+width/4,  width/2,
height/2);*/
54          }
55      }
```

8.19. Enemy2.java

```
1   import javax.swing.*;
2   import java.awt.*;
3   import java.awt.event.*;
4   import java.util.*;
5   /*クリボーみたいな敵(強化版)*/
6
```

```
 7  public class Enemy2 extends Character{
 8      protected int wayx = -2;
 9      private ImageIcon icon1, icon2;
10      public Enemy2(int x, int y){
11          super(x, y, 32, 32, 2, 1);
12          gw = gh = 32;
13      }
14      //アップデート
15      public void update(Field field){
16          super.update(field);
17          if(isCollisionX){
18            wayx = -wayx;
19            isCollisionX = false;
20          }
21          moveX(wayx);
22      }
23      //移動
24      public void moveX(int d){
25        super.moveX(d);
26        vx = d;
27      }
28      //描画処理
29      public void draw(Graphics g, int offsetX, int offsetY){
30           icon1 = new ImageIcon(getClass().getResource("pictures/enemy2-
1.png"));
31           icon2 = new ImageIcon(getClass().getResource("pictures/enemy2-
2.png"));
32          if(dir == 1){
33            image = icon1.getImage();
34          }
35          else{
36            image = icon2.getImage();
37          }
38          g.drawImage(image, (int)x +offsetX, (int)y+offsetY, gw, gh,null);
39      }
40  }
```

8.20. Enemy3.java

```
 1  import javax.swing.*;
 2  import java.awt.*;
 3  import java.awt.event.*;
 4  import java.util.*;
 5  /*はねるパタパタみたいな敵*/
 6
 7  public class Enemy3 extends Character{
 8      protected int wayx = -1;
 9      protected int jumpcount = 0;
10      private ImageIcon icon1, icon2;
11      public Enemy3(int x, int y){
12          super(x, y, 32, 32, 1, 1);
13          gw = gh = 32;
14      }
15      //ジャンプ機能
16      public void jum(){
17          if(isGround){
18              if(jumpcount < 2){
19                  vy = -2f;
20                  isGround = false;
21                  jumpcount += 1;
22              }
23              else{
24                  vy = -3f;
25                  isGround = false;
26                  jumpcount = 0;
27              }
28          }
29      }
30      //アップデート
31      public void update(Field field){
32          super.update(field);
33          if(isCollisionX){
34              wayx = -wayx;
35              isCollisionX = false;
36          }
```

```
37              moveX(wayx);
38
39              if(isGround == true){
40                 this.jum();
41              }
42          }
43          //移動
44          public void moveX(int d){
45             super.moveX(d);
46             vx = d;
47          }
48          //描画処理
49          public void draw(Graphics g, int offsetX, int offsetY){
50                 icon1 = new ImageIcon(getClass().getResource("pictures/enemy3-
1.png"));
51                 icon2 = new ImageIcon(getClass().getResource("pictures/enemy3-
2.png"));
52             if(dir == 1){
53                image = icon1.getImage();
54             }
55             else{
56                image = icon2.getImage();
57             }
58                    g.drawImage(image, (int)x +offsetX, (int)y+offsetY-10, gw+10,
gh+10,null);
59          }
60    }
```

8.21. Enemy4.java

```
1   import javax.swing.*;
2   import java.awt.*;
3   import java.awt.event.*;
4   import java.util.*;
5   /*弾飛ばす敵*/
6   //CharacterNum は 2
7   public class Enemy4 extends Character{
```

```
8       private ImageIcon icon1, icon2;
9       protected int cooltime = 0;      //攻撃までのクールタイム(この敵のジャンプもクールタイムで管理)
10      public Enemy4(int x, int y){
11          super(x, y, 32, 32, 2, 2);
12          gw = gh = 32;
13          this.dir = 1;
14          this.attackFlag = false;
15      }
16      //ジャンプ機能
17      public void jum(){
18          if(isGround){
19              vy = -4f;
20              isGround = false;
21          }
22      }
23      //アップデート
24      public void update(Field field){
25          super.update(field);
26          if(isGround == true && cooltime == 300){
27             this.jum();
28          }
29
30          if(this.cooltime == 350){
31             this.cooltime = 0;
32             this.attackFlag = true;
33          }
34
35          this.cooltime += 1;
36      }
37
38      //描画処理
39      public void draw(Graphics g, int offsetX, int offsetY){
40          //System.out.println((int)x +offsetX);
41           icon1 = new ImageIcon(getClass().getResource("pictures/enemy4-1.png"));
42           icon2 = new ImageIcon(getClass().getResource("pictures/enemy4-2.png"));
```

```
43              if(dir == 1){
44                 image = icon1.getImage();
45              }
46              else{
47                 image = icon2.getImage();
48              }
49                    g.drawImage(image, (int)x +offsetX, (int)y+offsetY-10, gw+10,
gh+10,null);
50          }
51   }
52
```

8.22. EnemyBall.java

```
 1  import javax.swing.*;
 2  import java.awt.*;
 3  import java.awt.event.*;
 4  import java.util.*;
 5
 6  /*Ball クラスと異なるのは、軌道とキャラナンバーのみ*/
 7
 8
 9  public class EnemyBall extends Ball{
10          EnemyBall(int x, int y){
11                  super(x, y);
12                  changeCharaNum();
13                  fallcount = 0;
14                  this.dir = 1;
15                  this.gw = 16;
16                  this.gh = 16;
17          };
18
19          public void changeCharaNum(){
20                  this.characterNum = 97;
21          }
22
23          private int fallcount;
```

```
24          @Override
25          public void update(Field field){
26                super.update(field);
27                if(dir==0){
28                      vx = 2f;
29                }else{
30                      vx = -2f;
31                }
32                if(isGround){
33                      System.out.println(hp);
34                      vy += -2;
35                      fallcount+=1;
36                      if(fallcount>5){
37                            hp = 0;
38                      }
39                }
40          }
41
42          public void EBdir(int i){
43                this.dir = i;
44          }
45
46
47          @Override
48          public void draw(Graphics g, int offsetX, int offsetY){
49                super.draw(g, offsetX, offsetY);
50                /*System.out.println("ok");
51                g.setColor(Color.blue);
52                      g.fillOval((int)x+offsetX+width/4,   (int)y+offsetY+width/4,   width/2,
height/2);*/
53          }
54    }
```