

情報理工学域 メディア情報学プログラム  
プログラミング演習 最終レポート

DX2 グループ 11 Gladiator

2411593 斎藤寛仁

2411603 佐々木晃誠

2411712 Puller Stefano Daniele

2026 年 2 月 18 日

# 1 概要説明

私たちのグループは、シンプルなシューティングゲームを作成しました。画面上部から敵がランダムに降りてきて攻撃します。敵を躊躇したり、矢を放って敵を迎撃します。敵を倒すことでスコアが上昇し、一定のスコアに達するとボスが登場します。各ステージのボスを倒すことで、次のステージに移動します。ボスを倒すとそのキャラの特殊スキルを自分の能力として使うことができるようになります。プレイヤーのライフが 0になるとゲームオーバーです。

基本的なキー操作は移動と攻撃で、カーソルキーで移動し、Space キーで矢を放って攻撃します。プレイ中に一時停止することも可能です。

プレイヤーのライフは 3 が上限で、敵に当たるか敵の攻撃を受けると 1 減ります。ボスが登場したタイミングと、ボスを倒して次のステージに進んだタイミングでライフが全回復します。

プレイ画面の上部には、現在のスコアとステージ数、プレイヤーの残りライフが表示されています。特殊スキルは画面下部に順次追加されます。



図 1: プレイ画面

MVC モデルを採用し、主に Puller が M、佐々木が V、斎藤が C を担当しました。プログラムのコードは GitHub で管理しました。また、「実装予定機能リスト」を作成し、未開発の機能を赤、開発中の機能をオレンジ、開発済みの機能を緑にし、開発中の機能の最後に『(名前)』を書いて、誰が今どの機能を開発中なのかを一目で把握できるようにしました。

(文責：佐々木)

## 2 設計方針

### 2.1 基本構想

本プログラムは、Java の Swing ライブラリを用いた縦スクロール型の 2D シューティングゲームである。プレイヤーは自機を操作し、ランダムに出現する敵やボスを撃破しながらステージを進行する。開発にあたっては、拡張性と保守性を高めるため、適切なデザインパターンとデータ構造を選定した。

### 2.2 採用したアルゴリズムとデータ構造

目的の処理を実現するために、以下のデータ構造とアルゴリズムを採用した。

- **MVC モデルの採用 (アーキテクチャ)**

プログラムの責務を明確に分離するため、Model-View-Controller アーキテクチャを採用した。

– **理由:** ゲームロジック（当たり判定や敵の出現管理）と描画処理が混在すると、コードが肥大化しデバッグが困難になるためである。Model がデータを保持し、View が描画に専念することで、複数人での並行開発（ロジック担当と UI 担当の分業）を円滑に進めることができた。

- **ArrayList による動的オブジェクト管理 (データ構造)**

ゲーム内に登場する敵キャラクターや弾丸は、ゲーム進行に伴い頻繁に生成・消滅を繰り返す。そのため、固定長配列ではなく、可変長配列である `java.util.ArrayList` を採用した。

– **理由:** 画面上のオブジェクト数は予測不可能であるため、要素の追加・削除が容易なリスト構造が必要であった。メインループ内ではこのリストを走査し、一括して移動と描画を行っている。

- **Area クラスによる精密な当たり判定（アルゴリズム）**

矩形判定 (Rectangle) に加え、より精密な判定が必要な場合に `java.awt.geom.Area` クラスを用いた領域計算アルゴリズムを採用した。

- **理由:** 単純な四角形同士の判定では、円形の弾丸や不規則な形状の敵に対して「当たっていないのに当たった」という違和感が生じるため、画像の不透明部分に基づいた正確な衝突判定を実現するためである。

## 2.3 クラス構成とクラス図

効率的な実装を行うため、オブジェクト指向の継承とポリモーフィズムを積極的に活用した。主要なクラス構成を以下に示す。

- **GameObject クラス (基底クラス):** 全てのキャラクターの親クラス。座標  $(x, y)$ 、画像、および抽象メソッド `move()`, `draw()` を持つ。これにより、異なる種類のオブジェクトを同一のリストで管理可能にしている。
- **HostileEntity クラス:** `GameObject` を継承し、敵キャラクター共通の機能 (HP 管理、被ダメージ処理、スコア加算) を実装している。
- **Boss クラス / Minion クラス:** `HostileEntity` をさらに具体化し、ボス特有の振る舞いや雑魚敵の挙動を定義している。

クラス間の関係図を図 2に示す。本図に含まれるクラスは全て本プロジェクトのために新規に設計・実装したものである。また、図中の色分けは各クラスの実装担当者を表しており、MVC モデルに基づいた分担が行われていることを示している。

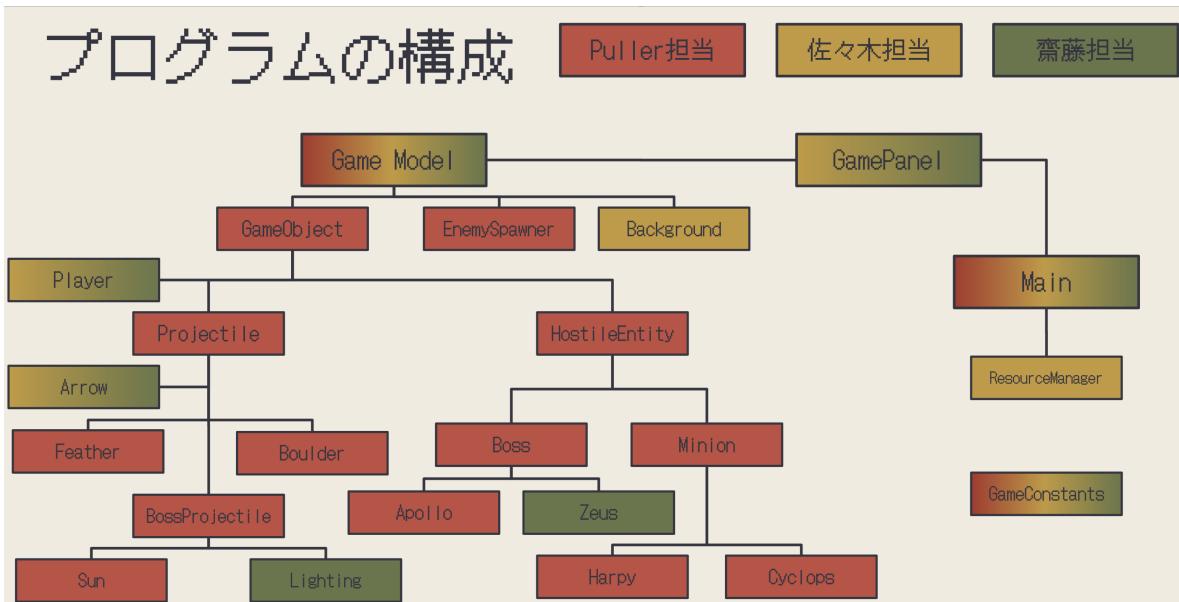


図 2: クラス構成図と開発担当

### 3 プログラムの説明

#### 3.1 齋藤担当

私が作成したのは、コントローラーと第二ボス Zeus の実装です。

##### 3.1.1 コントローラーの実装

Listing 1: GamePanel.java (一部抜粋)

```

1  private void updatePlayerVelocity() {
2      Player p = model.getPlayer();
3
4      int vx = 0;
5      int vy = 0;
6
7      if (leftPressed && !rightPressed) vx = -1;
8      if (rightPressed && !leftPressed) vx = 1;
9      if (upPressed && !downPressed) vy = -1;
10     if (downPressed && !upPressed) vy = 1;
11
12     p.setVelX(vx);
  
```

```

13     p.setVelY(vy);
14 }
15
16 // Reset keys when restarting game
17 private void resetKeyState() {
18     leftPressed = false;
19     rightPressed = false;
20     upPressed = false;
21     downPressed = false;
22
23     Player p = model.getPlayer();
24     if (p != null) {
25         p.setVelX(0);
26         p.setVelY(0);
27     }
28 }
```

キー操作は、使用するキーが押されている状態か押されていない状態かを true,false で表して処理しています。updatePlayerVelocity では、変数 vx,vy を用いて、キーが押された方向に Player が上下左右に動くことができるようとした。例えば、左のカーソルキーだけ押されている状態であれば、Player は左に移動することとなり、座標としては、負の方向に進むはずなので vx=-1 となるようにした。キー操作の入力を検知してから、vx,vy を変更するという操作をすると、方向転換の時に、止まってしまってうまく方向転換ができなかった。しかし、true,false の判別によりそれを解消することができ、より滑らかなキー操作が可能になった。

続いて、resetKeyState です。ここは、ゲームを再開したり、ゲームオーバーになってゲームを新しく始めるときなどの状態を初期化するときに、用いるプログラムです。ここは、上で説明したプログラムで変更した状態を元の状態に変更します。元の状態に変更しないと再開と同時に Player が動き始めてしまうので、方向キーの状態は、全て false にし、変数 vx,vy も 0 に戻します。

Listing 2: GamePanel.java (一部抜粋)

```

1 @Override
2 public void keyPressed(KeyEvent e) {
3     int key = e.getKeyCode();
4     GameState state = model.getState();
5 }
```

```

6      // Title Screen Input
7      if (state == GameState.TITLE) {
8          if (key == KeyEvent.VK_SPACE) {
9              model.initGame(); // Start Game
10             startTime = System.currentTimeMillis();
11             endTime = 0;
12         }
13     }
14
15     // Playing State Input
16     else if (state == GameState.PLAYING) {
17         if (key == KeyEvent.VK_LEFT) leftPressed = true;
18         if (key == KeyEvent.VK_RIGHT) rightPressed = true;
19         if (key == KeyEvent.VK_UP) upPressed = true;
20         if (key == KeyEvent.VK_DOWN) downPressed = true;
21
22         if (key == KeyEvent.VK_SPACE) {
23             model.setFiring(true);
24         }
25
26         if (key == KeyEvent.VK_P) {
27             model.setState(GameState.PAUSED);
28             resetKeyState();
29             System.out.println("Game Paused");
30         }
31
32         updatePlayerVelocity();
33
34         // Placeholder for Abilities
35         // ABILITY 1
36         if(model.getCurrentLevelIndex() > 3 && key == KeyEvent.VK_1) {
37             model.ability1();
38         }
39
40         // ABILITY 2
41         if (model.getCurrentLevelIndex() > 7 && key == KeyEvent.VK_2) {
42             model.ability2();
43         }
44
45         // ABILITY 3

```

```

46     if (key == KeyEvent.VK_3) System.out.println("Ability 3 pressed");
47 }
48
49 else if (state == GameState.PAUSED) {
50     if (key == KeyEvent.VK_P) {
51         model.setState(GameState.PLAYING);
52         resetKeyState();
53         System.out.println("Game Resumed");
54     }
55 }
56
57 else if (state == GameState.MESSAGE) {
58     if (key == KeyEvent.VK_SPACE) {
59         model.resumeGame(); // Go back to Playing
60         resetKeyState();
61     }
62 }
63
64 // Game Over State Input
65 else if (state == GameState.GAMEOVER) {
66     if (key == KeyEvent.VK_C) {
67         model.initGame(); // Retry
68         resetKeyState();
69         startTime = System.currentTimeMillis();
70         endTime = 0;
71     } else if (key == KeyEvent.VK_Q) {
72         System.exit(0); // Quit App
73     }
74 }
75 }
```

今まででは、Player の移動の操作をしました。keyPressed では、移動以外の処理を行なっています。ここでは、Game の状態に合わせて条件分岐処理を行っています。

ゲームのタイトルが出ている時は、スペースキーを押してスタートするような仕様にしているので、7,8 行目のように、ゲームの状態が TITLE の時に、スペースキーを押されたら、ゲーム開始するようにしています。

続いて、16 行目から 47 行目です。ゲームをしている時は、Player の移動だけではなく、矢を放ったり、一時停止や、特殊能力まであり、ここもこれら全て条件分岐で処理を

しています。まず、移動は、キーが押されたときの処理を行っているため、押されている方向キーのところは全て true に変更しています。そして、方向キーの押されているかの処理を終えた後、Player が動かないといけないので、updatePlayerVelocity を実行し、方向キーの変更を移動という形で表せるようにした。他の動作については、動作にあったキーを押された時に、その動作のクラスを呼び出します。

以降、他の状態については、その状態に操作できるキーが押されて時は、その動作ができるメソッドを呼び出して処理できるようにしています。

Listing 3: GamePanel.java (一部抜粋)

```
1  @Override
2  public void keyReleased(KeyEvent e) {
3      int key = e.getKeyCode();
4      if (key == KeyEvent.VK_LEFT) leftPressed = false;
5      if (key == KeyEvent.VK_RIGHT) rightPressed = false;
6      if (key == KeyEvent.VK_UP) upPressed = false;
7      if (key == KeyEvent.VK_DOWN) downPressed = false;
8
9      if (key == KeyEvent.VK_SPACE) {
10         model.setFiring(false);
11     }
12     if (model.getState() == GameState.PLAYING) {
13         updatePlayerVelocity();
14     }
15 }
```

このクラスは、keyPressed で行われた変更を元の状態に戻します。このようにすることで、毎フレーム押された状態だけを処理することができます。

### 3.1.2 第二ボス Zeus の実装

第二ボス Zeus の実装は、Zeus 本体と Zeus が攻撃する雷 (Lighting) の二つのファイルを作成して実装しました。まず、本体についてです。

Listing 4: Zeus.java (一部抜粋)

```
1  public class Zeus extends Boss {
```

Listing 5: Zeus.java (一部抜粋)

```

1   @Override
2   public void move() {
3       super.move();
4       if (ability2Timer > 0 || ability1Phase){
5           if (ability1Timer > 0) {
6               // Update horizontal position
7               x += speedX;
8
9               // Bounce logic: If it hits the screen edges
10              if (x <= 0 ) {
11                  x = 0;
12                  speedX = -speedX; // Reverse direction
13                  ability1Timer--;
14              } else if (x >= GameConstants.WINDOW_WIDTH - width) {
15                  x = GameConstants.WINDOW_WIDTH - width;
16                  speedX = -speedX; // Reverse direction
17                  ability1Timer--;
18              }
19              if(shootTimer <= 0){
20                  shootLighting();
21                  resetShootTimer();
22              }
23              shootTimer--;
24          } else if (ability1Pause <= 0){
25              ability1();
26          } else {
27              ability1Pause--;
28          }
29      } else {
30          if (!ability2Started) {
31              if (Math.random() < 0.5) {
32                  x = 0;
33                  speedX = GameConstants.ZEUS_SPEED2;
34              } else {
35                  x = GameConstants.WINDOW_WIDTH - width;
36                  speedX = -GameConstants.ZEUS_SPEED2;
37              }
38              Random random = new Random();
39              ability2Repetitions = random.nextInt(3) + 1;
40              ability2Started = true;

```

```

41     }
42     x += speedX;
43
44     if (((x < (GameConstants.PLAYER_WIDTH - 20)) && (speedX < 0)) || (
45     x > (GameConstants.WINDOW_WIDTH - (width + GameConstants.PLAYER_WIDTH)) &&
46     speedX > 0)){
47         } else {
48             shootLighting();
49         }
50
51         if (x <= 0 ) {
52             x = 0;
53             speedX = -speedX; // Reverse direction
54             ability2Repetitions--;
55         } else if (x >= GameConstants.WINDOW_WIDTH - width) {
56             x = GameConstants.WINDOW_WIDTH - width;
57             speedX = -speedX; // Reverse direction
58             ability2Repetitions--;
59         }
60         if (ability2Repetitions == 0){
61             ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
62             ability2Started = false;
63             resetShootTimer();
64         }
65
66         if (secondPhase && ability2Timer > 0){
67             ability2Timer--;
68         }
69     }

```

この Zeus.java は、Boss を継承したクラスです。通常攻撃としては、2 種類あります。それを ability1,ability2 として表しています。そして、このボスは、第一段階と第二段階(secondPhase) の二つの状態を持っています。move メソッドでは、ability2 を発動していない時は、画面の端に行ったら折り返しながら、攻撃をするという動きをしています。そして、ability2 は、画面の端から端まで素早く動きながら攻撃し続けます。この端から恥まで動く動きを random を用いてランダムに決めています。この move メソッドの工夫点としては、変数 isSecondPhase と ability2Started の用意である。ability2 が発動でき

る状態になったら、ability2Started を true にし、そうでなければ false にしておくことで、簡単に Zeus が ability1 を使うのか、ability2 を使うのかを判別することができる。また、同様に、isSecondPhase により、Zeus の第二段階にいるかをこれによって判別することができることで、それぞれの段階の攻撃を用意することが容易にできた。

次に、雷 (Lighting) についてです。

Listing 6: Lighting.java (一部抜粋)

```
1 public class Lighting extends BossProjectile {
2
3     private int vely;
4
5     public Lighting(int x, int y, int ZeusSpeedX, boolean isSecondPhase,
6                     boolean friendly, boolean ability2Started) {
7         super(0,
8               0,
9               GameConstants.LIGHTING_WIDTH,
10              GameConstants.LIGHTING_HEIGHT,
11              isSecondPhase ? ResourceManager.lightingImg2 : ResourceManager
12              .lightingImg,
13              friendly ? Alignment.PLAYER : Alignment.ENEMY,
14              3,
15              GameConstants.LIGHTING_DAMAGE);
16
17     if (isSecondPhase) {
18         vely = GameConstants.LIGHTING_SPEED2;
19     } else {
20         vely = GameConstants.LIGHTING_SPEED1;
21     }
22
23     if(isPlayerProjectile){
24         vely = -vely;
25         maxHP = GameConstants.LIGHTING_HP;
26         currentHP = maxHP;
27         // set position for player
28         this.x = x + (GameConstants.PLAYER_WIDTH - width) / 2;
29         this.y = y - GameConstants.PLAYER_HEIGHT;
30     } else {
```

```

31         if (!ability2Started){
32
33             // set position for Zeus
34             this.x = (ZeusSpeedX > 0) ? x + GameConstants.ZEUS_WIDTH - width :
35             x + width;
36             } else {
37                 this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
38             width;
39             }
40         }
41
42     @Override
43     public void move() {
44         y += vely;
45
46         if (y > GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT || y <
47             GameConstants.HUD_HEIGHT - height) {
48             isDead = true;
49         }
50     }

```

Lighting は、Zeus の攻撃手段であり、Zeus を倒すと、Player の特殊能力にもなります。そのため、friendly という変数を用意することで、同じ Lighting でも Player の攻撃として用いたり、Boss の攻撃としても用いることができます。この変数を追加することで、Lighting を味方用と、ボス用と二つファイルを作らずに、一つのファイルで完結させることができました。また、変数 isSecondPhase により、第二段階に入ったことが判別できる。第二段階に入ったときに、画像を変えることで Lighting でもより強く見せることができました。このような変化を加えることで、ゲーム性が増しました。

(文責：齋藤)

### 3.2 佐々木担当

私は主に View 層における UI の実装と、Model 層におけるプレイヤーのライフ・ダメージ処理を担当しました。

#### 1、ゲーム画面および HUD の実装

ユーザーが直感的に状況を把握できるよう、ゲームの状態に応じた画面遷移を実装しました。具体的には、ゲーム起動時の「タイトル画面」、操作説明を含む「スタート画面」、そして「ゲームオーバー画面」を作成しました。ゲームオーバー時には最終スコアを表示し、キー入力（C で継続、Q で終了）によって次のアクションを選択できる機能を設けています。また、プレイ画面上部の HUD では、現在のスコア、進行中のステージ数、およびプレイヤーの残りライフをリアルタイムで表示するようにしました。ライフは数字ではなくハートのアイコンを用いることで、視認性を高めています。

Listing 7: HUD の描画 (GamePanel.java)

```
1  public class GamePanel extends JPanel implements KeyListener {
2      private void drawTopHUD(Graphics g) {
3          // Draw dark background bar
4          g.setColor(new Color(50, 50, 80));
5          g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
6              HUD_HEIGHT);
7
8          // Draw white separator line
9          g.setColor(Color.WHITE);
10         g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH
11             , GameConstants.HUD_HEIGHT);
12
13         // Font settings
14         setPixelFont(g, 18f);
15         int textY = 35;
16
17         // A. Score
18         g.drawString("SCORE:" + model.getScore(), 10, textY);
19
20         // B. Stage (Centered)
21         String stageText = model.getStageText();
22         int stageX = (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
23             stringWidth(stageText)) / 2;
24         g.drawString(stageText, stageX, textY);
25
26         // C. Hearts / Lives (Right aligned)
27         int maxLives = GameConstants.PLAYER_MAX_LIVES;
28         int currentLives = model.getLives();
29         int heartSize = 32;
30         int spacing = 8;
```

```

28         int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (
29             heartSize + spacing));
30         int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
31
32         for (int i = 0; i < maxLives; i++) {
33             // Determine which icon to draw (Full or Empty)
34             BufferedImage icon = (i < currentLives) ? ResourceManager.
35             heartFullImg : ResourceManager.heartEmptyImg;
36
37             if (icon != null) {
38                 g.drawImage(icon, startX + (i * (heartSize + spacing)),
39                 heartY, heartSize, heartSize, null);
40             } else {
41                 // Fallback drawing if images are missing
42                 g.setColor(i < currentLives ? Color.RED : Color.GRAY);
43                 g.fillOval(startX + (i * (heartSize + spacing)), heartY,
44                 heartSize, heartSize);
45             }
46         }
47     }
48 }
```

## 2、ライフ制度と無敵時間の導入

ゲームの難易度調整とプレイヤー体験の向上のため、プレイヤーに3つのライフを付与するシステムを構築しました。初期の実装では、敵や弾に接触した際、当たり判定が連続して発生しライフが一瞬で0になってしまい「多段ヒット」の問題が発生していました。これを解消するために、ダメージを受けた直後に一定時間の「無敵時間(クールタイム)」を導入しました。ダメージ発生時に damageTimer をセットし、このタイマーが作動している間は新たなダメージを受け付けないように処理を分岐させています。さらに、無敵時間中はプレイヤーキャラクターを点滅させることで、視覚的にもダメージを受けたことと無敵状態であることをユーザーが理解しやすいようにしました。

Listing 8: ライフとタイマーの定義・初期化 (GameModel.java)

```

1  public class GameModel {
2      // 追加ライフ機能用変数:
3      private int lives; // 初期ライフ
4      private int damageTimer; // ダメージを受けた後の無敵時間 (フレーム数)
5 }
```

```

6     public void initGame() {
7         // 追加ライフ初期化:
8         lives = GameConstants.PLAYER_MAX_LIVES;
9         damageTimer = 0;
10    }
11 }

```

Listing 9: 無敵時間のカウントダウン処理 (GameModel.java)

```

1  public class GameModel {
2      public void update() {
3          // 無敵時間の更新
4          if (damageTimer > 0) {
5              damageTimer--;
6          }
7      }
8
9      // ダメージ処理メソッド
10     private void playerTakesDamage() {
11         if (damageTimer == 0) { // 無敵時間中でなければダメージ
12             lives--;
13             damageTimer = 120; // フレーム（約秒）無敵にする 1803
14             System.out.println("Damage taken! Lives remaining: " + lives);
15
16             if (lives <= 0) {
17                 state = GameState.GAMEOVER;
18                 System.out.println("GAME OVER");
19             }
20         }
21     }
22 }

```

(文責：佐々木)

### 3.3 Puller 担当

私は、MVC モデルの中核となる Model 層の設計と実装を主導しました。ゲームの根幹となるメインループ、オブジェクト指向設計に基づくクラス階層の構築、物理演算（当たり判定）、および敵 AI のロジックを担当しました。

### 3.3.1 堅牢なオブジェクト管理システム

ゲーム内に登場するプレイヤー、敵、弾丸などの全オブジェクトを効率的に処理するため、基底クラス‘GameObject’を作成し、‘ArrayList’で一元管理する構造を採用しました。特筆すべき点として、メインループ(‘update’)内でオブジェクトリストを走査中に新たなオブジェクト(発射された弾丸など)を追加すると‘ConcurrentModificationException’が発生する問題を防ぐため、一時的なバッファリスト‘newObjectsBuffer’を導入しました。これにより、ループ終了後に安全にリストを統合する堅牢な設計を実現しました。

Listing 10: 安全なオブジェクト更新ロジック (GameModel.java)

```
1 // ループ中のエラー回避のバッファ
2 private ArrayList<GameObject> newObjectsBuffer;
3
4 public void update() {
5     // ... 省略() ...
6
7     // スポナーや入力処理によって生成されたオブジェクトをバッファに追加
8
9     // 安全なタイミングでメインリストに統合
10    objects.addAll(newObjectsBuffer);
11    newObjectsBuffer.clear();
12
13    // 全オブジェクトの移動と描画を一括処理(ポリモーフィズム)
14    for (GameObject obj : objects) {
15        obj.move();
16    }
17    // ... 省略() ...
18 }
```

### 3.3.2 精密な当たり判定と領域計算

シューティングゲームにおいて最も重要な当たり判定には、単純な矩形交差判定(‘intersects’)だけでなく、‘java.awt.geom.Area’クラスを用いた領域計算アルゴリズムを採用しました。単純な四角形同士の判定では、画像が透明な部分でも「当たった」と判定されてしまう問題があります。そこで、‘Area’クラスの‘intersect’メソッドを使用することで、オブジェクトの形状に基づいた論理積を計算し、ピクセル単位に近い精密な衝突判定を実現しました。

Listing 11: Area クラスを用いた精密判定 (GameModel.java)

```
1  private boolean checkIntersection(GameObject obj1, GameObject obj2) {
2      // 1. 形状 () を取得 Shape
3      Shape s1 = obj1.getShape();
4      Shape s2 = obj2.getShape();
5
6      // 2. 高速化のため、まずは境界矩形で簡易判定
7      if (!s1.getBounds2D().intersects(s2.getBounds2D())) {
8          return false;
9      }
10
11     // 3. クラスによる精密判定 (形状の重なりを計算) Area
12     Area area1 = new Area(s1);
13     Area area2 = new Area(s2);
14     area1.intersect(area2);
15
16     return !area1.isEmpty(); // 重なり部分が存在すれば衝突
17 }
```

### 3.3.3 弾丸の優先度システム (Power Level)

単なる「当たったら消える」だけでなく、弾丸に「強度 (Power Level)」の概念を導入しました。‘Projectile’クラスに‘powerLevel’フィールドを持たせ、弾丸同士が衝突した際、強度の高い方が生き残るロジックを実装しました（例：岩 (Lv2) は矢 (Lv1) を破壊して貫通するが、太陽 (Lv3) には破壊される）。これにより、戦略的な攻撃の選択が可能になりました。

Listing 12: 弾丸同士の相殺ロジック (GameModel.java)

```
1  if (p1.getPowerLevel() > p2.getPowerLevel()) {
2      p2.setDead(); // が強く、が破壊される p1p2
3  } else if (p2.getPowerLevel() > p1.getPowerLevel()) {
4      p1.setDead(); // が強く、が破壊される p2p1
5  } else {
6      p1.setDead(); // 同等の場合は相殺
7      p2.setDead();
8  }
```

### 3.3.4 ジェネリクスを用いた敵生成

敵の出現管理を行う ‘EnemySpawner’ クラスでは、Java のジェネリクス（‘Class<? extends Minion>’）を活用しました。これにより、スパナーのインスタンス生成時に「ハーピー」や「サイクロプス」といったクラスタイプを動的に指定するだけで、異なる種類の敵を生成可能にしました。また、出現間隔に乱数による「ゆらぎ」を持たせることで、単調なゲームプレイを防いでいます。

### 3.3.5 第一ボス (Apollo) の AI 実装

ステージ 1 のボスである ‘Apollo’ クラスを実装しました。継承元の ‘Boss’ クラスの機能を活用しつつ、画面端での反射移動や、HP 半減時に画像と移動速度が変化する「発狂モード（第二形態）」への状態遷移ロジックを組み込みました。

(文責 : Puller)

## 4 実行例

コードを実行すると、左の画面が出てくる。スペースキーを押すと右の画面に移る。



図 3: タイトル画面



図 4: スタート画面

再びスペースキーを押すと、右上の画面から左下の画面に移る。この画面になったら、方向キーとスペースキーを使って敵を倒す。一定の敵を倒すと、右の画面に移り、第一ボ

スのアポロンが登場する。

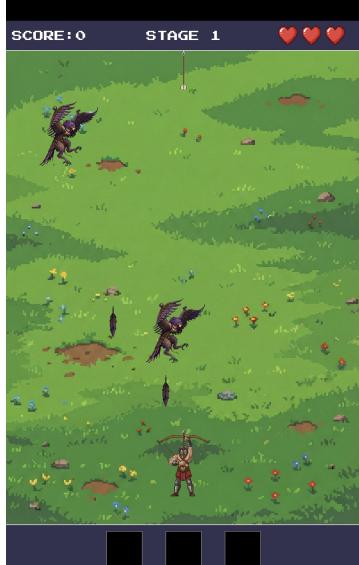


図 5: 第一ステージ



図 6: 第一ボス (アポロン)  
登場

右上の画面から、スペースキーを押すと、左下にある第一フェーズの第一ボスアポロンと戦う。攻撃し続けると、右下の画面のように変化する。



図 7: 第一ボス (アポロン)  
第一フェーズ



図 8: 第一ボス (アポロン)  
第二フェーズ

第一ボスアポロンを倒すと、左下の画面に移る。その画面で、スペースキーを押すと第二ステージが始まる。



図 9: 第一ステージ クリア



図 10: 第二ボス (ゼウス) 登場

左下のように、第一ボスを倒して、得られた太陽を使いながら第二ステージを攻略する。ステージが進むと、右下のように第二ボスゼウスが登場する。



図 11: 第二ステージ 特殊能力 1(太陽)

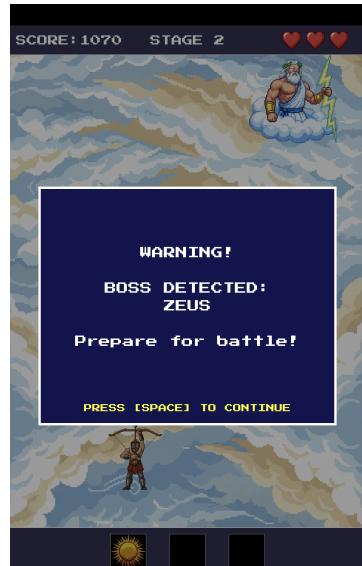


図 12: 第二ステージ

右上の画面でスペースキーを押すと、左下のように、第二ボスと戦う。ダメージを与え続けると、右下のように変化する。



図 13: 第二ボス (ゼウス)  
第一フェーズ



図 14: 第二ボス (ゼウス)  
第二フェーズ

第二ボスを倒すと、左下の画面に移る。スペースキーを押すと第三ステージが始まる。



図 15: 第二ステージ クリア



図 16: 第三ステージ 特殊能力 1(太陽)

右上、左下のようにボスを倒して得られた特殊能力を使ってスコアを稼ぐ。残りライフが 0になると、右下のようにゲームオーバーになる。



図 17: 第三ステージ 特殊能力 2(雷)



図 18: ゲームオーバー画面

(文責：齋藤)

## 5 考察

## 6 感想

### 6.1 斎藤

この授業で、初めて Java について学んだ。Java は、基本的な構文は、C 言語に似ていたがオブジェクト指向型プログラミングを初めて習ったので難しく感じた。しかし、グループ内課題でプログラミングをやっていると、オブジェクト指向の便利さに気づいた。例えば、継承である。継承は、親クラスの内容を引き継ぐ子クラスに行うものであり、それが今回のシューティングゲームを開発している時に、重要となった。なぜなら、このゲーム開発では、player, Boss, Arrow, 雑魚敵など様々なキャラクターやオブジェクトが登場してきて、それらを全て一から書くのが大変だから、GameObject と全体に共通するものを書いて大きなグループを作り、そこから派生して、Boss、BossProjectile など細かいグループを作つてコードを作成できることが、整理しやすくそれが作成のしやすさに繋がっていたからである。

私は、MVC モデルの C と M の一部を担当しました。C では、Player の上下左右の移動や、矢を放ったり、特殊能力を放つなどのさまざま操作ができるようにしました。Player の移動の際に、1 フレーム毎に縦、横の 4 方向しか移動が出来ないので、スムーズに斜めに移動することができるようになるなど、移動方向を増やしてみたいと思った。

この授業を通して、Java の基本的なことから実践まで学べるようになっていることが、他の授業には良い点であると思った。実践的な練習することで、プログラミングの理解が広がるだけではなく、周りとコミュニケーションをとり、進捗を確認し合いながら進めるることは、この授業で初めて行ったことであり、今後に活かすことができる経験であった。

(文責：斎藤)

### 6.2 佐々木

本授業を通じて、初めての Java の学習から始まり、最終的にはグループでのゲーム開発という実践的な演習に取り組みました。Java 特有のオブジェクト指向における「継承」の概念は、初めは難しかったものの、開発が進むにつれてコードの再利用性を高めるために不可欠な技術であることを実感しました。

開発手法としては MVC モデルを採用し、私は主に V と M の一部を担当しました。具

体的には、ゲームの入り口となるタイトル画面やスタート画面、そしてプレイ画面やゲームオーバー画面といった UI 全般の実装に注力しました。単にグラフィックを表示するだけでなく、ゲームの状態に応じて動的に表示を切り替えるロジック部分の構築も経験し、UI と内部処理がどう連携すべきかを深く意識することができました。

また、GitHub を用いた共同開発も初めての経験でした。ブランチの管理やコンフリクトの解消など、個人開発では味わえない困難もありましたが、チームで一つのソースコードを育て上げる過程でバージョン管理の重要性を学びました。技術的な習得だけでなく、メンバーと協力して一つのプロジェクトを完遂する難しさと喜びを知ることができ、非常に実りある経験となりました。

(文責：佐々木)

### 6.3 Puller

本演習を通して、Java によるオブジェクト指向プログラミングと MVC アーキテクチャの実践的な理解を深めることができました。特に、Model 層の実装において「継承」と「ポリモーフィズム」を活用することで、数百個のオブジェクト（敵や弾丸）をわずか数行のループで一括管理できる拡張性の高さに感動しました。

グループワークの側面では、Git/GitHub を用いたバージョン管理の重要性を痛感しました。開発終盤において、各メンバーの変更（UI の調整やロジックの追加）を統合（マージ）する際にコンフリクトが発生することもありましたが、それを解消する過程で、他者のコードを読み解く力や、変更の影響範囲を予測する力が養われたと感じています。

今後の課題としては、アルゴリズムの最適化が挙げられます。現在の当たり判定は総当たり方式 ( $O(N^2)$ ) で実装されていますが、オブジェクト数が極端に増えると処理落ちする可能性があります。今後は「四分木 (Quadtree)」などの空間分割手法を学習し、より大規模な弾幕ゲームにも対応できる効率的な衝突判定を実装してみたいと思いました。

「プログラミング演習」の授業全体を通して、単なる構文の学習にとどまらず、設計・実装・テスト・統合というソフトウェア開発の一連のライフサイクルを体験できたことは、非常に有意義でした。

(文責：Puller)

## 7 付録 1：操作法マニュアル

ゲームを起動すると、タイトル画面が出てきます。Space キーを押してスタート画面に移り、再度 Space キーを押すことでゲームが始まります。



図 19: タイトル画面



図 20: スタート画面

次に、プレイ中のキー操作について説明します。

表 1: プレイ中のキー操作

移動	カーソルキー	画面全体を縦横無尽に移動。
攻撃	Space キー	長押しで連射が可能。
一時停止、再開	P	プレイ中に押すことでゲームを一時停止、再度押すことでゲームを再開。
特殊スキル	1,2,3	各ステージのボスを倒すことで手に入る特殊スキルは、それぞれ 1,2,3 を押すことで発動。

ゲームオーバー時には、C を押すとゲームをもう一度プレイ (Continue) でき、Q を押

すとゲームをやめる (Quit) ことができます。



図 21: ゲームオーバー画面

(文責: 佐々木)

## 8 付録2：プログラムリスト

Listing 13: Main.java

```
1 package main;
2
3 import view.GamePanel;
4 import view.ResourceManager;
5
6 import javax.swing.*;
7
8 public class Main {
9
10    public static void main(String[] args) {
11        // 1. Load resources BEFORE creating the window
12        ResourceManager.loadImages();
13
14        // 2. Setup the game window
15        JFrame frame = new JFrame("Shooting Game MVC");
16        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17        frame.add(new GamePanel());
18        frame.pack();
19        frame.setLocationRelativeTo(null); // ウィンドウを画面中央に
20        frame.setVisible(true);
21    }
22}
```

Listing 14: GameModel.java

```
1 package model;
2
3 import view.ResourceManager;
4
5 import java.awt.*;
6 import java.util.ArrayList;
7 import java.util.List;
8 import java.util.Random;
9 import java.awt.geom.Area;
```

```

11 // --- GameModel (M) ---
12 public class GameModel {
13     private ArrayList<GameObject> objects;
14     private Player player;
15     private boolean isGameOver = false;
16     private Random rand = new Random();
17     private ArrayList<GameObject> newObjectsBuffer; // 弾を追加するための予約リスト（ループ中のエラー回避用）
18     private GameState state; // 現在のゲーム状態
19     private boolean isFiring; // スペースキーが押されているか
20     private int shotTimer; // 連射間隔を制御するタイマー
21     private int arrowDamage;
22     private int arrowInterval;
23
24     // Score progression
25     private static int score = 0; //スコアの導入
26     private int nextTargetScore;
27     private int currentLevelIndex = 0;
28     private boolean isBossActive = false; //ボスのフェーズの確認
29
30     // 追加ライフ機能用変数:
31     private int lives;// 初期ライフ
32     private int damageTimer; // ダメージを受けた後の無敵時間（フレーム数）
33
34     private int ability1Timer;
35     private int ability2Timer;
36     private int ability3Timer;
37
38     //background
39     private Background background;
40
41     // for writing the stage number in GamePanel
42     private int currentStage;
43
44     private String[] currentMessageLines;
45
46     private List<EnemySpawner> activeSpawners;
47
48     public GameModel() {
49         objects = new ArrayList<>();

```

```

50     newObjectsBuffer = new ArrayList<>();
51     activeSpawners = new ArrayList<>();
52     state = GameState.TITLE; // 最初はタイトル画面から
53 }
54
55 public void initGame() {
56     objects.clear();
57     newObjectsBuffer.clear();
58     activeSpawners.clear();
59     player = new Player((GameConstants.WINDOW_WIDTH - GameConstants.
60     PLAYER_WIDTH)/2, GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT -
61     GameConstants.PLAYER_HEIGHT);
62     objects.add(player);
63
64     // Initialize Background
65     background = new Background();
66
67     isFiring = false;
68     shotTimer = 0;
69     arrowDamage = GameConstants.ARROW_DAMAGE;
70     arrowInterval = GameConstants.ARROW_INTERVAL;
71     state = GameState.PLAYING;
72     score = 0; //スコアをリセット
73
74     // 追加ライフ初期化:
75     lives = GameConstants.PLAYER_MAX_LIVES;
76     damageTimer = 0;
77
78     //Score progression resets
79     this.currentLevelIndex = 0;
80     this.nextTargetScore = GameConstants.LEVEL_MILESTONES[
81     currentLevelIndex];
82     isBossActive = false;
83
84     // STAGE 1 SETUP: Add Harpy Spawner
85     activeSpawners.add(new EnemySpawner(Harpy.class, GameConstants.
HARPY_SPAWN_INTERVAL, GameConstants.HARPY_SPAWN_VARIANCE));
     this.currentStage = 1;
86
87     String tutorial = "WELCOME GLADIATOR!\n\n" +

```

```

86         "Controls:\n" +
87         "[KEY-ARROWS] Move\n" +
88         "[SPACE] Shoot\n" +
89         "[P] Pause\n\n" +
90         "Defeat enemies\n" +
91         "and survive.\n" +
92         "Good Luck!";  

93     showMessage(tutorial);
94 }
95
96 public static void addScore(int points) {
97     score += points;
98 }
99
100 public void setFiring(boolean firing) {
101     this.isFiring = firing;
102 }
103
104 public GameState getState() {
105     return state;
106 }
107
108 public void setState(GameState s) {
109     this.state = s;
110 }
111
112 public void update() {
113     if (state != GameState.PLAYING) return;
114
115     // Update background scrolling
116     if(background!= null) {
117         background.update();
118     }
119     checkLevelProgression();
120
121     // --- 連射ロジック ---
122     if (isFiring) {
123         if (shotTimer == 0) {
124             playerShoot();
125             shotTimer = GameConstants.ARROW_INTERVAL;

```

```

126         }
127     }
128     if (shotTimer > 0) {
129         shotTimer--;
130     }
131
132     // 無敵時間の更新
133     if (damageTimer > 0) {
134         damageTimer--;
135     }
136
137     if (ability1Timer > 0) {
138         ability1Timer--;
139     }
140
141     if (ability2Timer > 0) {
142         ability2Timer--;
143     }
144
145     if (ability3Timer > 0) {
146         ability3Timer--;
147     }
148
149     // --- NEW SPAWN LOGIC ---
150     // Iterate through all active spawners
151     if (!isBossActive) {
152         for (EnemySpawner spawner : activeSpawners) {
153             // If spawner says "True", create that enemy
154             if (spawner.update()) {
155                 spawnMinion(spawner.getEnemyType());
156             }
157         }
158     }
159
160     // オブジェクト追加
161     objects.addAll(newObjectsBuffer);
162     newObjectsBuffer.clear();
163
164     // 移動
165     for (GameObject obj : objects) {

```

```

166         obj.move();
167     }
168
169     //当たり判定
170     checkCollisions();
171
172     //削除
173     objects.removeIf(obj -> obj.isDead());
174 }
175
176 /**
177 * Helper method to instantiate the correct enemy based on Class type.
178 */
179 private void spawnMinion(Class<? extends Minion> type) {
180     int x,y;
181     if (type == Harpy.class) {
182         x = rand.nextInt(GameConstants.WINDOW_WIDTH - GameConstants.
HARPY_WIDTH); // Simple random X
183         y = GameConstants.HUD_HEIGHT - GameConstants.HARPY_HEIGHT; // Start at top
184         Harpy h = new Harpy(x, y, this);
185         newObjectsBuffer.add(h);
186     }
187     else if (type == Cyclops.class) {
188         x = rand.nextInt(GameConstants.WINDOW_WIDTH - GameConstants.
CYCLOPS_WIDTH); // Simple random X
189         y = GameConstants.HUD_HEIGHT - GameConstants.CYCLOPS_HEIGHT;
190         Cyclops g = new Cyclops(x, y, this); // Pass 'this' (GameModel)
191         newObjectsBuffer.add(g);
192     }
193 }
194
195 /**
196 * Allows enemies (Minions/Bosses) to add projectiles to the game.
197 * Using a specific method is safer than exposing the whole list.
198 */
199 public void spawnEnemyProjectile(Projectile p) {
200     if (p != null) {
201         newObjectsBuffer.add(p);
202     }

```

```

203     }
204
205     private void checkLevelProgression() {
206         // ボスがいたら、何もしない
207         if (isBossActive) return;
208         //次のフェーズがなかつたら return
209         if (currentLevelIndex >= GameConstants.LEVEL_MILESTONES.length) return
210         ;
211
212         if (score >= nextTargetScore) {
213             //apply the effect we decided in the applyLevelEffects function
214             applyLevelEffects(currentLevelIndex);
215             currentLevelIndex++;
216             System.out.println("The current level index is: " +
217             currentLevelIndex);
218             nextTargetScore = GameConstants.LEVEL_MILESTONES[currentLevelIndex
219             ];
220         }
221     }
222
223     private void applyLevelEffects(int levelIndex) {
224         switch (levelIndex) {
225             case 0: //START OF THE GAME
226                 System.out.println("Start of the Game");
227                 break;
228
229             case 1:
230                 System.out.println("Difficulty UP!");
231                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
232                 (0.8);
233                 break;
234
235             case 2:
236                 System.out.println("Difficulty UP!");
237                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
238                 (0.8);
239                 break;
240
241             case 3:
242                 showMessage("WARNING!\n\nBOSS DETECTED:\nAPOLLO\n\nPrepare for

```

```

battle!");
238         isBossActive = true;
239         clearEverything();
240         spawnApollo();
241         healPlayer();
242         break;
243     case 4:
244         showMessage("STAGE 1 CLEARED!\n\nEntering the Heavens...\n\n
nArrow Damage doubled!\n\n Press [1] to use\nABILITY 1:\nAPOLLO'S SUN");
245         if (background != null) {
246             clearEverything();
247             healPlayer();
248             background.setImage(ResourceManager.stage2Img);
249             background.setSpeed(GameConstants.SCREEN_SPEED);
250             ability1Timer = 0;
251             this.currentStage = 2;
252
253             // DOUBLE ARROW DAMAGE
254             arrowDamage*=2;
255
256             // Clear old spawners (remove Stage 1 config)
257             activeSpawners.clear();
258             // Harpy Spawner
259             activeSpawners.add(new EnemySpawner(Harpy.class, 100, 50))
260 ;
261             // Cyclops Spawner
262             activeSpawners.add(new EnemySpawner(Cyclops.class,
263                             GameConstants.CYCLOPS_SPAWN_INTERVAL,
264                             GameConstants.CYCLOPS_SPAWN_VARIANCE));
265
266         }
267         break;
268     case 5:
269         System.out.println("Difficulty UP!");
270         for (EnemySpawner s : activeSpawners) s.increaseDifficulty
(0.9);
271         break;
272     case 6:
273         System.out.println("Difficulty UP!");
274         for (EnemySpawner s : activeSpawners) s.increaseDifficulty

```

```

(0.9);

274         break;
275     case 7:
276         showMessage("WARNING!\n\nBOSS DETECTED:\nZEUS\nPrepare for
battle!");
277         isBossActive = true;
278         clearEverything();
279         spawnZeus();
280         healPlayer();
281         break;
282     case 8:
283         showMessage("STAGE 2 CLEARED!\n\nEntering the INFERNO...\n\n
Shooting speed increased!\n\nPress [2] to use\nABILITY 2:\nZEUS'S
LIGHTING");
284         if (background != null) {
285             clearEverything();
286             background.setImage(ResourceManager.stage3Img);
287             background.setSpeed(0);
288             ability2Timer = 0;
289             arrowInterval = GameConstants.ARROW_INTERVAL2;
290             this.currentStage = 3;
291             activeSpawners.clear();
292             activeSpawners.add(new EnemySpawner(Harpy.class, 100, 50))
293 ;
294             // Cyclops Spawner
295             activeSpawners.add(new EnemySpawner(Cyclops.class,
296                 GameConstants.CYCLOPS_SPAWN_INTERVAL,
297                 GameConstants.CYCLOPS_SPAWN_VARIANCE));
298             break;
299         case 9:
300     }
301 }

302
303 public void showMessage(String text) {
304     // Split the text by newline character to handle multiple lines
305     this.currentMessageLines = text.split("\n");
306     this.state = GameState.MESSAGE;
307 }
308

```

```

309     public void bossDefeated() {
310         System.out.println("BOSS DEFEATED! Stage clear.");
311         this.isBossActive = false;
312         healPlayer();
313     }
314
315     // 全ての敵を消すメソッド
316     private void clearEverything() {
317         // 敵または羽の場合、消す"""
318         objects.removeIf(obj -> obj instanceof HostileEntity || obj instanceof
319             Projectile);
320         newObjectsBuffer.removeIf(obj -> obj instanceof HostileEntity || obj
321             instanceof Projectile);
322     }
323
324     // プレイヤーが撃つ（から呼ばれる）Controller
325     public void playerShoot() {
326         if (!isGameOver) {
327             // プレイヤーの中央上から発射
328             Arrow a = new Arrow(player.getX() + (GameConstants.PLAYER_WIDTH-
329                 GameConstants.ARROW_WIDTH)/2, player.getY() - GameConstants.ARROW_HEIGHT,
330                 arrowDamage);
331             newObjectsBuffer.add(a);
332         }
333     }
334
335     public void ability1() {
336         if (ability1Timer > 0) return;
337         ability1Timer = GameConstants.ABILITY1TIMER;
338         Sun sun = new Sun(
339             player.getX() + GameConstants.PLAYER_WIDTH / 2,
340             player.getY(),
341             0, false, true
342         );           newObjectsBuffer.add(sun);
343         System.out.println("Player shoots sun");
344     }
345
346     public void ability2() {
347         if (ability2Timer > 0) return;
348         ability2Timer = GameConstants.ABILITY2TIMER;

```

```

345     Lighting l = new Lighting(
346         player.getX(),
347         player.getY(),
348         0, false, true, false
349     );           newObjectsBuffer.add(l);
350     System.out.println("Player shoots Lighting");
351 }
352
353 private void spawnApollo() {
354     Apollo apollo = new Apollo(this);
355     objects.add(apollo);
356     System.out.println("APOLLO HAS DESCENDED!");
357 }
358
359 public void spawnZeus() {
360     Zeus zeus = new Zeus(this);
361     objects.add(zeus);
362     System.out.println("ZEUS HAS DESCENDED!");
363 }
364
365 // ダメージ処理メソッド
366 private void playerTakesDamage() {
367     if (damageTimer == 0) { // 無敵時間中でなければダメージ
368         lives--;
369         damageTimer = 120; // フレーム（約秒）無敵にする 1803
370         System.out.println("Damage taken! Lives remaining: " + lives);

371         if (lives <= 0) {
372             state = GameState.GAMEOVER;
373             System.out.println("GAME OVER");
374         }
375     }
376 }
377 }

378
379 private void healPlayer() {
380     lives = GameConstants.PLAYER_MAX_LIVES;
381 }
382
383
384 private boolean checkIntersection(GameObject obj1, GameObject obj2) {

```

```

385     // 1. take the precise shapes
386     Shape s1 = obj1.getShape();
387     Shape s2 = obj2.getShape();
388
389     // 2. Quick check: if the outer rectangles don't touch we skip the
390     // complicated calculations
390     if (!s1.getBounds2D().intersects(s2.getBounds2D())) {
391         return false;
392     }
393
394     // 3. Precise Calculation
395     Area area1 = new Area(s1);
396     Area area2 = new Area(s2);
397
398     area1.intersect(area2);
399
400     // if the area is not empty it means they are touching
401     return !area1.isEmpty();
402 }
403
404 /**
405 * Centralized Collision Logic.
406 * Iterates through all objects to check for intersections.
407 */
408 private void checkCollisions() {
409     for (int i = 0; i < objects.size(); i++) {
410         GameObject objA = objects.get(i);
411         if (objA.isDead()) continue;
412
413         for (int j = i + 1; j < objects.size(); j++) {
414             GameObject objB = objects.get(j);
415             if (objB.isDead()) continue;
416
417             if (checkIntersection(objA, objB)) {
418                 handleCollision(objA, objB);
419             }
420         }
421     }
422 }
423

```

```

424 /**
425 * Handles the specific logic when two objects collide.
426 * Uses Projectile Power Levels and Alignment to determine the outcome.
427 */
428 private void handleCollision(GameObject a, GameObject b) {
429
430     // --- CASE 1: PROJECTILE vs PROJECTILE ---
431     if (a instanceof Projectile && b instanceof Projectile) {
432         Projectile p1 = (Projectile) a;
433         Projectile p2 = (Projectile) b;
434
435         // Same team projectiles do not destroy each other
436         if (p1.getAlignment() == p2.getAlignment()) return;
437
438         // Compare Power Levels to see who survives
439         if (p1.getPowerLevel() > p2.getPowerLevel()) {
440             p2.setDead(); // p1 dominates
441
442             // SUN CASE: p1 is a Sun, it takes damage equal to p2's damage
443             if (p1 instanceof BossProjectile) {
444                 ((BossProjectile) p1).reduceHealth(p2.getDamage());
445             }
446
447         } else if (p2.getPowerLevel() > p1.getPowerLevel()) {
448             p1.setDead(); // p2 dominates
449
450             // SUN CASE: If p2 is a Sun, it takes damage equal to p1's
451             // damage
452             if (p2 instanceof BossProjectile) {
453                 ((BossProjectile) p2).reduceHealth(p1.getDamage());
454             }
455
456         } else {
457             // Equal power (e.g., Arrow vs Feather) -> Both destroyed
458             p1.setDead();
459             p2.setDead();
460         }
461     }
462 }
```

```

463 // --- CASE 2: PROJECTILE vs LIVING ENTITY (Player or Enemy) ---
464 Projectile proj = null;
465 GameObject entity = null;
466
467 // Identify which is which
468 if (a instanceof Projectile) { proj = (Projectile) a; entity = b; }
469 else if (b instanceof Projectile) { proj = (Projectile) b; entity = a;
}
470
471 if (proj != null) {
    // Sub-case A: Projectile hits Player
    if (entity instanceof Player) {
        if (proj.getAlignment() == Alignment.ENEMY) {
            playerTakesDamage();
            if (!proj.isPenetrating()) proj.setDead();
        }
    }
    // Sub-case B: Projectile hits Enemy (Harpy, Apollo, Golem, etc.)
    else if (entity instanceof HostileEntity) {
        HostileEntity enemy = (HostileEntity) entity;
        // Only damage if the projectile belongs to the Player
        if (proj.getAlignment() == Alignment.PLAYER) {
            enemy.takeDamage(proj.getDamage());
        }
        // SUN CASE: If the projectile is a Sun, it also loses HP
        upon contact
        if (proj instanceof BossProjectile) {
            ((BossProjectile) proj).reduceHealth(1);
        }
        else if (!proj.isPenetrating()) {
            proj.setDead();
        }
    }
    return;
}
499
500 // --- CASE 3: PHYSICAL COLLISION (Player vs Enemy Body) ---

```

```

501     if ((a instanceof Player && b instanceof HostileEntity) ||
502         (b instanceof Player && a instanceof HostileEntity)) {
503         playerTakesDamage();
504     }
505 }
506
507 // 無敵時間中かどうか
508 public boolean isInvincible() {
509     return damageTimer > 0;
510 }
511
512 public int getAbilityNthTimer(int n) {
513     switch (n){
514         case 1:
515             return ability1Timer;
516         case 2:
517             return ability2Timer;
518         case 3:
519             return ability3Timer;
520         default:
521             System.out.println("getability ERROR");
522     }
523     return ability1Timer;
524 }
525
526 public boolean isAbilityUnclocked(int abilityIndex) {
527     // Logic for Ability 1 (Sun)
528     if (abilityIndex == 1) {
529         // Unlocks after defeating the first boss (Apollo)
530         // Apollo is Level Index 4. So > 4 means Stage 2 started.
531         return this.currentLevelIndex > 4;
532     }
533     // Logic for Ability 2 (Lighting)
534     if (abilityIndex == 2) {
535         // Unlocks after defeating the second boss (Zeus)
536         // Zeus is Level Index 8. So > 8 means Stage 3 started.
537         return this.currentLevelIndex > 8;
538     }
539     // Future logic for Ability 2 and 3
540     return false;

```

```
541     }
542
543     public void resumeGame() {
544         this.state = GameState.PLAYING;
545     }
546
547     //SETTERS & GETTERS
548     public ArrayList<GameObject> getObjects() {
549         return objects;
550     }
551
552     public Player getPlayer() {
553         return player;
554     }
555
556     public boolean isGameOver() {
557         return isGameOver;
558     }
559
560     public int getScore() {
561         return score;
562     }
563
564     public int getLives() {
565         return lives;
566     }
567
568     public Background getBackground() {
569         return background;
570     }
571
572     public String getStageText(){
573         if (currentStage > 3) {
574             return "EXTRA STAGE";
575         }
576         return "STAGE " + currentStage;
577     }
578
579     public int getCurrentLevelIndex(){
580         return this.currentLevelIndex;
```

```
581     }
582
583     public String[] getCurrentMessageLines() {
584         return currentMessageLines;
585     }
586 }
```

Listing 15: GameConstants.java

```
1 package model;
2
3 public final class GameConstants {
4     // 画面のサイズ
5     public static final int WINDOW_WIDTH = 600;
6
7     // HEIGHTS
8     public static final int HUD_HEIGHT = 50;
9     public static final int FIELD_HEIGHT = 800;
10    public static final int BOTTOM_HUD_HEIGHT = 100;
11    public static final int WINDOW_HEIGHT = HUD_HEIGHT + FIELD_HEIGHT +
12        BOTTOM_HUD_HEIGHT;
13
14    // ゲームの設定
15    public static final int FPS = 60;
16
17    // 背景の設定
18    public static final double SCREEN_SPEED = 1.0;
19
20    // の設定 PLAYER
21    public static final int PLAYER_WIDTH = 90;
22    public static final int PLAYER_HEIGHT = PLAYER_WIDTH*923/721; // set to
the image height width ration
23    public static final int PLAYER_SPEED = 8;
24    public static final int PLAYER_MAX_LIVES = 3;
25
26    // ABILITY TIMERS
27    public static final int ABILITY1TIMER = FPS * 12; // 12 seconds
28    public static final int ABILITY2TIMER = FPS * 8; // 8 seconds
29    public static final int ABILITY3TIMER = FPS * 10; // 10 seconds
```

```
30 // の設定 ARROW
31 public static final int ARROW_WIDTH = 10;
32 public static final int ARROW_HEIGHT = 70; // set to the image height
width ration
33 public static final int ARROW_SPEED = 30;
34 public static final int ARROW_INTERVAL = 20;
35 public static final int ARROW_INTERVAL2 = 15;
36 public static final int ARROW_DAMAGE = 1;
37
38 // ****
39 // の設定 MINIONS
40 // ****
41
42 // の設定 HARPY
43 public static final int HARPY_WIDTH = 100;
44 public static final int HARPY_HEIGHT = HARPY_WIDTH *1911/1708; // set to
the image height width ration
45 public static final int HARPY_XSPEED = 4;
46 public static final int HARPY_YSPEED = 2;
47 public static final int HARPY_HP = 2;
48 public static final int HARPY_SCORE_POINTS = 10;
49 public static final int HARPY_SPAWN_INTERVAL = 120;
50 public static final int HARPY_SPAWN_VARIANCE = HARPY_SPAWN_INTERVAL / 2;
51
52 // の設定 FEATHER
53 public static final int FEATHER_WIDTH = 12;
54 public static final int FEATHER_HEIGHT = FEATHER_WIDTH * 1698/378; // set
to the image height width ration
55 public static final int FEATHER_SPEED = 7;
56 public static final int FEATHER_FIRE_INTERVAL = 90;
57 public static final int FEATHER_FIRE_VARIANCE = FEATHER_FIRE_INTERVAL / 2;
58 public static final int FEATHER_DAMAGE = 1;
59
60 // の設定 CYCLOPS
61 public static final int CYCLOPS_WIDTH = 150;
62 public static final int CYCLOPS_HEIGHT = CYCLOPS_WIDTH;
63 public static final double CYCLOPS_YSPEED = 2;
64 public static final int CYCLOPS_HP = 10;
65 public static final int CYCLOPS_SCORE_POINTS = 50;
66 public static final int CYCLOPS_SPAWN_INTERVAL = 400;
```

```
67     public static final int CYCLOPS_SPAWN_VARIANCE = CYCLOPS_SPAWN_INTERVAL /  
68         2;  
69     public static final int CYCLOPS_MOVEMENT_TIMER = 20;  
70     public static final int CYCLOPS_ATTACK_TIMER = 60;  
71  
72     // の設定 BOULDER  
73     public static final int BOULDER_WIDTH = 100;  
74     public static final int BOULDER_HEIGHT = BOULDER_WIDTH;  
75     public static final double BOULDER_INITIAL_SPEED = 0;  
76     public static final double BOULDER_GRAVITY = 0.3;  
77     public static final int BOULDER_DAMAGE = 5;  
78  
79     // *****  
80     // の設定 BOSS  
81     // *****  
82  
83     // の設定 APOLLO  
84     public static final int APOLLO_WIDTH = 200;  
85     public static final int APOLLO_HEIGHT = APOLLO_WIDTH * 1556 / 2463;  
86     public static final int APOLLO_SPEED1 = 4;  
87     public static final int APOLLO_SPEED2 = APOLLO_SPEED1 * 3 / 2;  
88     public static final int APOLLO_HP = 50; //50  
89     public static final int APOLLO_SCORE_POINTS = 1000;  
90  
91     // の設定 SUN  
92     public static final int SUN_WIDTH = 150;  
93     public static final int SUN_HEIGHT = SUN_WIDTH;  
94     public static final double SUN_SPEED1 = 6;  
95     public static final double SUN_SPEED2 = SUN_SPEED1 * 3 / 2;  
96     public static final int SUN_DAMAGE = 1;  
97     public static final int SUN_HP = 20;  
98  
99     // の設定 ZEUS  
100    public static final int ZEUS_SPEED = 6;  
101    public static final int ZEUS_SPEED2 = 8;  
102    public static final int ZEUS_WIDTH = 150;  
103    public static final int ZEUS_HEIGHT = 150;  
104    public static final int ZEUS_HP = 150; // 150  
105    public static final int ZEUS_SCORE_POINTS = 1500;  
106    public static final int ZEUS_SHOOT_TIMER = 45;
```

```

106     public static final int ZEUS_SHOOT_TIMER2 = 30;
107     public static final int ZEUS_ABILITY1_PAUSE = 30;
108     public static final int ZEUS_ABILITY1_PAUSE2 = 20;
109     public static final int ZEUS_ABILITY2_TIMER = FPS * 5; // 5 seconds
110
111     //の設定 LIGHTING
112     public static final int LIGHTING_WIDTH = 25;
113     public static final int LIGHTING_HEIGHT = 150;
114     public static final int LIGHTING_SPEED1 = 10;
115     public static final int LIGHTING_SPEED2 = 15;
116     public static final int LIGHTING_DAMAGE = 1;
117     public static final int LIGHTING_HP = 10;
118
119     //の設定 SCORE
120     // STAGE 1
121     public static final int SCORE_STAGE1_PHASE1 = 0;
122     public static final int SCORE_STAGE1_PHASE2 = HARPY_SCORE_POINTS * 10; // 100
123     public static final int SCORE_STAGE1_PHASE3 = HARPY_SCORE_POINTS * 30; // 300
124     public static final int SCORE_FOR_BOSS_1 = HARPY_SCORE_POINTS * 50; // 500
125     // STAGE 2
126     public static final int SCORE_STAGE2_PHASE1 = SCORE_FOR_BOSS_1 +
127         APOLLO_SCORE_POINTS; //1500
128     public static final int SCORE_STAGE2_PHASE2 = SCORE_STAGE2_PHASE1 + 500; // 2000
129     public static final int SCORE_STAGE2_PHASE3 = SCORE_STAGE2_PHASE2 + 750; // 2750
130     public static final int SCORE_FOR_BOSS_2 = SCORE_STAGE2_PHASE3 + 750; // 3500
131     // STAGE 3
132     public static final int SCORE_STAGE3_PHASE1 = SCORE_FOR_BOSS_2 +
133         ZEUS_SCORE_POINTS; // 5000
134     public static final int SCORE_STAGE3_PHASE2 = 100000; // 6000
135     public static final int SCORE_STAGE3_PHASE3 = 300000; // 7000
136     public static final int SCORE_FOR_BOSS_3 = 500000; // 8000
137     // EXTRA STAGE
138     public static final int SCORE_EXTRA_STAGE = 10000; // 10000
139
140     // We put them all in an Array

```

```

139     public static final int[] LEVEL_MILESTONES = {
140         SCORE_STAGE1_PHASE1, SCORE_STAGE1_PHASE2, SCORE_STAGE1_PHASE3,
141         SCORE_FOR_BOSS_1,
142         SCORE_STAGE2_PHASE1, SCORE_STAGE2_PHASE2, SCORE_STAGE2_PHASE3,
143         SCORE_FOR_BOSS_2,
144         SCORE_STAGE3_PHASE1, SCORE_STAGE3_PHASE2, SCORE_STAGE3_PHASE3,
145         SCORE_FOR_BOSS_3,
146         SCORE_EXTRA_STAGE
147     };
148
149     //他の設定
150     public static final int FLASH_TIMER = 5;
151     private GameConstants(){} //オブジェクトを作らないようにする private
152 }
```

Listing 16: GameObject.java

```

1 package model;
2
3 import java.awt.*;
4 import java.awt.geom.Rectangle2D;
5 import java.awt.image.BufferedImage;
6
7 // --- 1. キャラクターの親クラス ---
8 public abstract class GameObject {
9     protected int x, y;
10    protected int width, height;
11    protected boolean isDead = false; // になったら消える true
12    protected BufferedImage image;
13
14    public GameObject(int x, int y, int w, int h, BufferedImage image) {
15        this.x = x;
16        this.y = y;
17        this.width = w;
18        this.height = h;
19        this.image = image;
20    }
21
22    public abstract void move();
23 }
```

```

24     public abstract void draw(Graphics g);
25
26     public Rectangle getBounds() {
27         return new Rectangle(x, y, width, height);
28     }
29
30     public Shape getShape() {
31         return new Rectangle2D.Float(x, y, width, height);
32     }
33
34     public boolean isDead() {
35         return isDead;
36     }
37
38     public void setDead() {
39         this.isDead = true;
40     }
41
42     public int getY() {
43         return y;
44     }
45
46     public int getX() {
47         return x;
48     }
49 }
```

Listing 17: GameState.java

```

1 package model;
2
3 // ゲームの状態を定義
4 public enum GameState {
5     TITLE, PLAYING, PAUSED, MESSAGE, GAMEOVER
6 }
```

Listing 18: Alignment.java

```

1 package model;
2
```

```

3  /**
4   * Alignment Enum
5   * Defines which "team" a game object belongs to.
6   * Used to prevent friendly fire and determine collision logic.
7   */
8  public enum Alignment {
9      PLAYER, // Belongs to the Player (Targets Enemies)
10     ENEMY // Belongs to Enemies (Targets Player)
11 }

```

Listing 19: Background.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 /**
8  * Background Class
9  * Handles the infinite scrolling background using a "Mirroring" technique.
10 * It draws the original image and a vertically flipped copy above it
11 * to create a seamless loop without needing a specific seamless texture.
12 */
13 public class Background {
14     private double y; // Vertical position (double for smooth movement)
15     private double speed; // Scrolling speed
16     private BufferedImage image;
17
18     // Screen dimensions
19     private final int WIDTH = GameConstants.WINDOW_WIDTH;
20     private final int HEIGHT = GameConstants.FIELD_HEIGHT / 2;
21
22     public Background() {
23         this.image = ResourceManager.stage1Img;
24         this.speed = 0;
25         this.y = GameConstants.HUD_HEIGHT;
26     }
27
28     public void setImage(BufferedImage newImage){

```

```

29     this.image = newImage;
30 }
31
32 public void setSpeed(double newSpeed) {
33     this.speed = newSpeed;
34 }
35
36 public void update() {
37     // Move the background downwards
38     y += speed;
39
40     // Reset position when a full cycle is completed to prevent overflow
41     if (y >= HEIGHT * 2 + GameConstants.HUD_HEIGHT) {
42         y = GameConstants.HUD_HEIGHT;
43     }
44 }
45
46 public void draw(Graphics g) {
47     if (image == null) return;
48
49     int currentY = (int) y;
50
51     if (isStage2Img()){
52         // DRAWING STRATEGY:
53         // The screen is 800px tall. Our "Tile" is 400px.
54         // We need to cover the screen from Y=0 to Y=800.
55         // Since 'currentY' moves down, we need to draw tiles above and
below it.
56
57         // We assume the pattern is: [Normal] [Flipped] [Normal] [Flipped]
58         ...
59
60         // 1. Draw Normal Tile at current Y
61         // Covers: y to y+400  x
62         drawForceSize(g, currentY, false);
63
64         // 2. Draw Flipped Tile BELOW
65         // Covers: y+400 to y+800
66         drawForceSize(g, currentY + HEIGHT, true);

```

```

67         // 3. Draw Flipped Tile ABOVE
68         // Covers: y-400 to y. (Crucial for when y starts at 0 or is small
69     )
70         drawForceSize(g, currentY - HEIGHT, true);
71
72         // 4. Draw Normal Tile ABOVE that
73         // Covers: y-800 to y-400. (Crucial for the loop wrap-around)
74         drawForceSize(g, currentY - (HEIGHT * 2), false);
75     } else {
76         g.drawImage(image, 0, GameConstants.HUD_HEIGHT, WIDTH, HEIGHT * 2,
77         null);
78     }
79 }
80
81 /**
82  * Helper method to draw a tile either normally or vertically flipped.
83  * @param g Graphics context
84  * @param yPos The Y position to draw at
85  * @param isFlipped If true, draws the image upside down (mirrored)
86  */
87 private void drawForceSize(Graphics g, int yPos, boolean isFlipped) {
88     if (!isFlipped) {
89         // NORMAL: Force width 600, height 400
90         g.drawImage(image, 0, yPos, WIDTH, HEIGHT, null);
91     } else {
92         // FLIPPED: Force width 600, height 400 (but drawn upwards)
93         // Destination Y starts at bottom (yPos + HEIGHT)
94         // Height is negative (-HEIGHT) to flip it
95         g.drawImage(image, 0, yPos + HEIGHT, WIDTH, -HEIGHT, null);
96     }
97 }
98
99 private boolean isStage2Img(){
100     return (this.image == ResourceManager.stage2Img);
101 }
```

Listing 20: Projectile.java

```
1 package model;
```

```

2
3 import view.ResourceManager;
4
5 import java.awt.*;
6 import java.awt.image.BufferedImage;
7
8 /**
9  * Projectile Class
10 * Abstract base class for all flying objects (Arrows, Feathers, Suns,
11 * Boulders).
12 * It introduces the concept of "Power Level" to handle projectile-vs-
13 * projectile collisions.
14 */
15 public abstract class Projectile extends GameObject {
16
17     protected Alignment alignment;
18
19     // Power Level determines priority when two projectiles collide:
20     // 0 = Ephemeral (Destroyed by anything)
21     // 1 = Light (Arrow, Feather)
22     // 2 = Heavy (Boulder - Destroys Light)
23     // 3 = Ultimate (Sun - Destroys Heavy and Light)
24     protected int powerLevel;
25
26     protected int damage;
27
28     // If true, the projectile does not vanish after hitting a target (e.g.,
29     // The Sun)
30     protected boolean isPenetrating;
31
32     public Projectile(int x, int y, int w, int h, BufferedImage image,
33     Alignment alignment, int powerLevel, int damage) {
34         super(x, y, w, h, image);
35         this.alignment = alignment;
36         this.powerLevel = powerLevel;
37         this.damage = damage;
38         this.isPenetrating = false; // Default: destroys itself on impact
39     }
40
41     // Getters

```

```

38     public Alignment getAlignment() { return alignment; }
39     public int getPowerLevel() { return powerLevel; }
40     public int getDamage() { return damage; }
41     public boolean isPenetrating() { return isPenetrating; }
42 }
```

Listing 21: Player.java

```

1 package model;
2
3 import view.ResourceManager; // Import the manager
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6 import java.awt.geom.Ellipse2D;
7
8 // --- クラス Player 自分 () ---
9 public class Player extends GameObject {
10     private int velX = 0; // 横の移動速度
11     private int velY = 0; // 縦の移動速度
12     private int speed = GameConstants.PLAYER_SPEED;
13     private int level = 1;
14
15     public Player(int x, int y) {
16         super(x, y, GameConstants.PLAYER_WIDTH, GameConstants.PLAYER_HEIGHT,
17               ResourceManager.playerImg); // 40の四角
18         x40
19     }
20
21     @Override
22     public void move() {
23         x += velX;
24         y += velY;
25
26         // 画面からはみ出さないように制限
27         if (x < 0) x = 0;
28         if (x > GameConstants.WINDOW_WIDTH - width) x = GameConstants.
29             WINDOW_WIDTH - width;
30         if (y < GameConstants.HUD_HEIGHT) y = GameConstants.HUD_HEIGHT;
31         if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height)
32             y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height;
33     }
34 }
```

```

30
31     @Override
32     public void draw(Graphics g) {
33         if(image != null){
34             g.drawImage(image, x, y, width, height, null);
35         }
36         else{
37             // Fallback if image failed to load
38             g.setColor(Color.BLUE);
39             g.fillRect(x, y, width, height);
40         }
41     }
42
43     @Override
44     public Shape getShape() {
45         // we define the margin
46         float paddingX = width * 0.3f; // remove 20 % width
47         float paddingY = height * 0.2f; // remove 10 % height
48
49         // the smaller hitbox get centered compared to the original immage
50         return new Ellipse2D.Float(
51             x + paddingX / 2,           // move right
52             y + paddingY / 2,           // move down
53             width - paddingX,          // reduce width
54             height - paddingY);       // reduce height
55     }
56
57
58     // から呼ばれるメソッド Controller
59     public void setVelX(int vx) {
60         this.velX = vx * speed;
61     }
62
63     public void setVelY(int vy) {
64         this.velY = vy * speed;
65     }
66
67     public int getLevel(){
68         return level;
69     }

```

70 }

Listing 22: Arrow.java

```
1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 public class Arrow extends Projectile {
8
9     public Arrow(int x, int y, int arrowDamage) {
10         // Alignment: PLAYER
11         // Power Level: 1 (Light) -> Can be destroyed by Boulders (Lvl 2) or
12         // Sun (Lvl 3)
13         // Damage: Standard Arrow Damage
14         super(x, y, GameConstants.ARROW_WIDTH, GameConstants.ARROW_HEIGHT,
15               ResourceManager.arrowImg,
16                     Alignment.PLAYER, 1, arrowDamage);
17     }
18
19     @Override
20     public void move() {
21         y -= GameConstants.ARROW_SPEED;
22         if (y < -height) {
23             isDead = true;
24         }
25     }
26
27     @Override
28     public void draw(Graphics g) {
29         if (ResourceManager.arrowImg != null) {
30             g.drawImage(image, x, y, width, height, null);
31         } else {
32             g.setColor(Color.YELLOW);
33             g.fillRect(x, y, width, height);
34         }
35     }
36 }
```

Listing 23: BossProjectile.java

```
1 package model;
2
3 import java.awt.image.BufferedImage;
4
5 public abstract class BossProjectile extends Projectile {
6
7     protected boolean isPlayerProjectile;
8     protected int maxHP;
9     protected int currentHP;
10
11    public BossProjectile(int x, int y, int w, int h, BufferedImage image,
12                          Alignment alignment, int powerLevel, int damage){
13        super(x, y, w, h, image, alignment, powerLevel, damage);
14    }
15
16    public void reduceHealth(int damage) {
17        // Only Player's Projectiles can die
18        if (!isPlayerProjectile) return;
19        this.currentHP -= damage;
20        if (this.currentHP <= 0) {
21            this.isDead = true;
22        }
23    }
}
```

Listing 24: Boss.java

```
1 package model;
2
3 import java.awt.*;
4 import java.awt.image.BufferedImage;
5
6 public abstract class Boss extends HostileEntity {
7
8    public Boss(int x, int y, int w, int h, BufferedImage image, int hp, int
9               scorePoints, GameModel model) {
10        super(x, y, w, h, image, hp, scorePoints, model);
11    }
}
```

```

12     @Override
13     public void takeDamage(int dmg) {
14         super.takeDamage(dmg);
15
16         if (this.isDead){
17             model.bossDefeated();
18         }
19     }
20 }
```

Listing 25: Apollo.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 /**
8 * Apollo Class
9 * Represents the first Boss of the game.
10 * It moves horizontally, bounces off walls, and shoots "Sun" projectiles.
11 * It has two phases: Normal and Enraged (Red & Fast).
12 */
13 public class Apollo extends Boss {
14
15     private int speedX = GameConstants.APOLLO_SPEED1;
16     private boolean secondPhase = false; // Flag to track if the boss is in "
17     Rage Mode"
18
19     public Apollo(GameModel model) {
20         // Call the parent constructor (Boss -> HostileEntity)
21         // Parameters: x, y, width, height, HP, Score Points
22         super(
23             (GameConstants.WINDOW_WIDTH - GameConstants.APOLLO_WIDTH) / 2,
24             // Start in the middle
25             GameConstants.HUD_HEIGHT, // Start below HUD
26             GameConstants.APOLLO_WIDTH,
27             GameConstants.APOLLO_HEIGHT,
28             ResourceManager.apolloImg,
```

```

27             GameConstants.APOLLO_HP,
28             GameConstants.APOLLO_SCORE_POINTS, // Score awarded when
29     defeated
30             model
31         );
32     this.image = ResourceManager.apolloImg;
33 }
34
35 @Override
36 public void move() {
37     super.move();
38     // Update horizontal position
39     x += speedX;
40
41     // Bounce logic: If it hits the screen edges
42     if (x <= 0 || x >= GameConstants.WINDOW_WIDTH - width) {
43         speedX = -speedX; // Reverse direction
44
45         // Trigger shooting mechanism via GameModel
46         // We pass the current phase status to decide if the Sun should be
47         // Red/Fast
48         shootSun();
49     }
50 }
51
52 private void shootSun(){
53     Sun s = new Sun(x, y, speedX, secondPhase, false);
54     model.spawnEnemyProjectile(s);
55 }
56
57 @Override
58 public void takeDamage(int dmg) {
59     // 1. Apply damage using the parent class logic (reduces HP, checks
60     // death, adds score)
61     super.takeDamage(dmg);
62
63     // 2. Specific Logic for Apollo: Check for Second Phase (Rage Mode)
64     // If HP drops below 50% and we are not yet in the second phase...
65     if (hp <= maxHp / 2 && !secondPhase) {
66         image = ResourceManager.apolloImg2; // Change sprite to Red Apollo

```

```

64         speedX = (speedX > 0) ? GameConstants.Apollo_SPEED2 : -
65         GameConstants.Apollo_SPEED2; // Double the movement speed
66         secondPhase = true; // Activate the flag
67         System.out.println("Apollo entering Phase 2!");
68     }
69 }
70
71 @Override
72 public void draw(Graphics g) {
73     BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
74     apolloHitImg : image;
75
76     if (image != null) {
77         // Directional Flipping Logic
78         if (speedX > 0) {
79             // Moving RIGHT: Draw normally
80             g.drawImage(imgToDraw, x, y, width, height, null);
81         } else {
82             // Moving LEFT: Flip the image horizontally
83             // We draw starting at (x + width) and use a negative width to
84             flip
85             g.drawImage(imgToDraw, x + width, y, -width, height, null);
86         }
87     } else {
88         // Fallback: Draw an Orange rectangle if images fail to load
89         g.setColor(Color.ORANGE);
90         g.fillRect(x, y, width, height);
91     }
92 }
93 }
```

Listing 26: Sun.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.geom.Ellipse2D;
6 import java.awt.image.BufferedImage;
7
```

```

8  public class Sun extends BossProjectile {
9
10
11     private double preciseX, preciseY;
12     private double velX, velY;
13
14     private double initialSize;
15
16     public Sun(int summonerX, int summonerY, int ApolloSpeedX, boolean
17     isSecondPhase, boolean friendly) {
18         // Call parent constructor
19         // HP = 1 (doesn't matter), Score = 0
20         super(0,
21             0,
22             GameConstants.SUN_WIDTH,
23             GameConstants.SUN_HEIGHT,
24             isSecondPhase ? ResourceManager.sunImg2 : ResourceManager.
25             sunImg,
26             friendly ? Alignment.PLAYER : Alignment.ENEMY,
27             3,
28             GameConstants.SUN_DAMAGE);
29
30         this.isPlayerProjectile = friendly;
31         this.isPenetrating = true;
32
33         if (isPlayerProjectile) {
34             maxHP = GameConstants.SUN_HP;
35             currentHP = maxHP;
36         }
37
38         this.initialSize = GameConstants.SUN_WIDTH;
39
40         velX = 1;
41         velY = 1;
42         double currentSpeed;
43         double angleRadians;
44
45         if (isSecondPhase) {
46             currentSpeed = GameConstants.SUN_SPEED2;
47         } else {
48             currentSpeed = GameConstants.SUN_SPEED1;

```

```

46 }
47
48     if (!friendly){
49         // sun comes from apollo
50         preciseX = (ApolloSpeedX > 0) ? (summonerX + GameConstants.
51 APOLLO_WIDTH + width / 2) : (summonerX - width / 2);
52         preciseY = summonerY + height;
53         x = (int) preciseX;
54         y = (int) preciseY;
55         // --- Phase Logic ---
56         // --- Trajectory Calculation ---
57         double minAngle = 20.0;
58         double maxAngle = 70;
59         angleRadians = Math.toRadians(minAngle + Math.random() * (maxAngle
60 - minAngle));
61     } else {
62         // sun comes from ability
63         preciseX = summonerX;
64         preciseY = summonerY - height / 2;
65
66         // 2. right boarder check
67         if (preciseX > GameConstants.WINDOW_WIDTH - width / 2) {
68             preciseX = GameConstants.WINDOW_WIDTH - width / 2;
69         }
70
71         // 3. left boarder check
72         if (preciseX < width / 2) {
73             preciseX = width / 2;
74         }
75
76         x = (int) preciseX;
77         y = (int) preciseY;
78         angleRadians = Math.toRadians(360-30);
79         velX = (Math.random() < 0.5) ? 1 : -1;
80     }
81
82     velX *= currentSpeed * Math.cos(angleRadians);
83     velY *= currentSpeed * Math.sin(angleRadians);
84
85     if (ApolloSpeedX < 0) {

```

```

84         velX = -velX;
85     }
86 }
87
88 @Override
89 public void move() {
90     preciseX += velX;
91     preciseY += velY;
92
93     x = (int) preciseX;
94     y = (int) preciseY;
95
96     double currentRadius = getCurrentSize() / 2.0;
97
98     // Wall Bounce
99     if (x - currentRadius < 0) {
100         x = (int)currentRadius;
101         preciseX = x;
102         velX = -velX;
103     }
104     if (x + currentRadius > GameConstants.WINDOW_WIDTH) {
105         x = (int)(GameConstants.WINDOW_WIDTH - currentRadius);
106         preciseX = x;
107         velX = -velX;
108     }
109
110     // Despawn
111     if (y - currentRadius > GameConstants.FIELD_HEIGHT + GameConstants.
112     HUD_HEIGHT ||
113             y + currentRadius < GameConstants.HUD_HEIGHT) {
114         isDead = true;
115     }
116
117     // Helper to calculate size based on HP
118     private double getCurrentSize() {
119         if (!isPlayerProjectile) return initialSize;
120
121         // Calculate ratio: currentHP / maxHP
122         double ratio = (double) currentHP / maxHP;

```

```

123     return initialSize * ratio;
124 }
125
126 @Override
127 public void draw(Graphics g) {
128     double size = getCurrentSize();
129     double radius = size / 2.0;
130
131     int drawX = (int) (x - radius);
132     int drawY = (int) (y - radius);
133     int drawSize = (int) size;
134
135     if (image != null) {
136         g.drawImage(image, drawX, drawY, drawSize, drawSize, null);
137     } else {
138         g.setColor(Color.YELLOW);
139         g.fillOval(x, y, width, height);
140     }
141 }
142
143 @Override
144 public Shape getShape() {
145     double size = getCurrentSize();
146     double radius = size / 2.0;
147     return new Ellipse2D.Float((float)(x - radius), (float)(y - radius), (float)size, (float)size);
148 }
149
150 }
```

Listing 27: Zeus.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6 import java.util.Random;
7
8 public class Zeus extends Boss {
```

```

9   private int speedX = GameConstants.ZEUS_SPEED;
10  private boolean secondPhase = false; // Flag to track if the boss is in "
11    Rage Mode"
12  private int shootTimer;
13  private int maxShootTimer;
14  private int ability1Timer;
15  private int ability2Timer;
16  private int ability1Pause;
17  private int maxAbility1Pause;
18  private int ability1Position;
19  private BufferedImage hitImg;
20  boolean ability1Phase;
21  boolean ability2Started;
22  private int ability2Repetitions;

23
24  public Zeus (GameModel model) {
25      super ( (GameConstants.WINDOW_HEIGHT - GameConstants.ZEUS_WIDTH) / 2,
26             GameConstants.HUD_HEIGHT,
27             GameConstants.ZEUS_WIDTH,
28             GameConstants.ZEUS_HEIGHT,
29             ResourceManager.zeusImg,
30             GameConstants.ZEUS_HP,
31             GameConstants.ZEUS_SCORE_POINTS,
32             model
33     );
34     maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER;
35     resetShootTimer();
36     maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE;
37     setAbility1Timer();
38     ability1Position = 1;
39     hitImg = ResourceManager.zeusHitImg;
40     ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
41     ability2Started = false;
42 }
43
44 @Override
45 public void move() {
46     super.move();
47     if (ability2Timer > 0 || ability1Phase){
48         if (ability1Timer > 0) {

```

```

48         // Update horizontal position
49         x += speedX;
50
51         // Bounce logic: If it hits the screen edges
52         if (x <= 0 ) {
53             x = 0;
54             speedX = -speedX; // Reverse direction
55             ability1Timer--;
56         } else if (x >= GameConstants.WINDOW_WIDTH - width) {
57             x = GameConstants.WINDOW_WIDTH - width;
58             speedX = -speedX; // Reverse direction
59             ability1Timer--;
60         }
61         if(shootTimer <= 0){
62             shootLighting();
63             resetShootTimer();
64         }
65         shootTimer--;
66     } else if (ability1Pause <= 0){
67         ability1();
68     } else {
69         ability1Pause--;
70     }
71 } else {
72     if (!ability2Started) {
73         if (Math.random() < 0.5) {
74             x = 0;
75             speedX = GameConstants.ZEUS_SPEED2;
76         } else {
77             x = GameConstants.WINDOW_WIDTH - width;
78             speedX = -GameConstants.ZEUS_SPEED2;
79         }
80         Random random = new Random();
81         ability2Repetitions = random.nextInt(3) + 1;
82         ability2Started = true;
83     }
84     x += speedX;
85
86     if (((x < (GameConstants.PLAYER_WIDTH - 20)) && (speedX < 0)) || (

```

```

speedX > 0)){
87     } else {
88         shootLighting();
89     }
90
91     if (x <= 0 ) {
92         x = 0;
93         speedX = -speedX; // Reverse direction
94         ability2Repetitions--;
95     } else if (x >= GameConstants.WINDOW_WIDTH - width) {
96         x = GameConstants.WINDOW_WIDTH - width;
97         speedX = -speedX; // Reverse direction
98         ability2Repetitions--;
99     }
100    if (ability2Repetitions == 0){
101        ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
102        ability2Started = false;
103        resetShootTimer();
104    }
105
106}
107
108    if (secondPhase && ability2Timer > 0){
109        ability2Timer--;
110    }
111}
112
113 private void ability1() {
114     if (ability1Position == 1){
115         ability1Phase = true;
116     }
117
118     if (ability1Position == 5) {
119         setAbility1Timer();
120         ability1Position = 1;
121         resetShootTimer();
122         return;
123     }
124
125     if(speedX > 0) {

```

```

126         x = ability1Position * (GameConstants.WINDOW_WIDTH-width)/4;
127     } else {
128         x = (4 - ability1Position) * (GameConstants.WINDOW_WIDTH-width)/4;
129     }
130     if (ability1Position % 2 == 0){
131         y = GameConstants.HUD_HEIGHT;
132     } else {
133         y = GameConstants.HUD_HEIGHT + height / 3;
134     }
135
136     shootLighting();
137     ability1Pause = maxAbility1Pause;
138     ability1Position++;
139 }
140
141 private void shootLighting(){
142     Lighting l = new Lighting(x, y, speedX, secondPhase, false,
143     ability2Started);
144     model.spawnEnemyProjectile(l);
145 }
146
147 @Override
148 public void takeDamage(int dmg) {
149     super.takeDamage(dmg);
150
151     if (hp <= maxHp / 2 && !secondPhase) {
152         image = ResourceManager.zeusImg2;
153         hitImg = ResourceManager.zeusHitImg2;
154         speedX = (speedX > 0) ? GameConstants.ZEUS_SPEED2 : -GameConstants
155         .ZEUS_SPEED2; // Double the movement speed
156         secondPhase = true;
157         maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER2;
158         maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE2;
159     }
160 }
161
162 private void setAbility1Timer(){
163     Random random = new Random();
164     ability1Timer = random.nextInt(4) + 1;
165     ability1Phase = false;

```

```

164 }
165
166     private void resetShootTimer(){
167         shootTimer = maxShootTimer;
168     }
169
170     @Override
171     public void draw(Graphics g) {
172         BufferedImage imgToDraw = (flashTimer > 0) ? hitImg : image;
173
174         if (image != null) {
175             if (speedX > 0) {
176                 g.drawImage(imgToDraw, x, y, width, height, null);
177             } else {
178                 g.drawImage(imgToDraw, x + width, y, -width, height, null);
179             }
180         } else {
181             g.setColor(Color.ORANGE);
182             g.fillRect(x, y, width, height);
183         }
184     }
185 }
```

Listing 28: Lighting.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.geom.Ellipse2D;
6
7 public class Lighting extends BossProjectile {
8
9     private int velY;
10    public Lighting(int x, int y, int ZeusSpeedX, boolean isSecondPhase,
11        boolean friendly, boolean ability2Started) {
12        super(0,
13              0,
14              GameConstants.LIGHTING_WIDTH,
15              GameConstants.LIGHTING_HEIGHT,
```

```

15         isSecondPhase ? ResourceManager.lightingImg2 : ResourceManager
16         .lightingImg,
17             friendly ? Alignment.PLAYER : Alignment.ENEMY,
18             3,
19             GameConstants.LIGHTING_DAMAGE);
20
21     this.isPlayerProjectile = friendly;
22     this.isPenetrating = true;
23
24     if (isSecondPhase) {
25         vely = GameConstants.LIGHTING_SPEED2;
26     } else {
27         vely = GameConstants.LIGHTING_SPEED1;
28     }
29
30     if(isPlayerProjectile){
31         vely = -vely;
32         maxHP = GameConstants.LIGHTING_HP;
33         currentHP = maxHP;
34         // set position for player
35         this.x = x + (GameConstants.PLAYER_WIDTH - width) / 2;
36         this.y = y - GameConstants.PLAYER_HEIGHT;
37     } else {
38         if (!ability2Started){
39
40             // set position for Zeus
41             this.x = (ZeusSpeedX > 0) ? x + GameConstants.ZEUS_WIDTH - width :
42             x + width;
43             } else {
44                 this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
45             width;
46             }
47             this.y = y + height;
48     }
49 }
50
51
52     @Override
53     public void move() {
54         y += vely;
55     }

```

```

52     if (y > GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT || y <
53         GameConstants.HUD_HEIGHT - height) {
54         isDead = true;
55     }
56 }
57
58 @Override
59 public void draw(Graphics g) {
60     if (image != null) {
61         g.drawImage(image, x, y, width, height, null);
62     } else {
63         g.setColor(Color.YELLOW);
64         g.fillOval(x, y, width, height);
65     }
66 }
67
68 @Override
69 public Shape getShape() {
70     return new Ellipse2D.Float(x, y, width, height);
71 }

```

Listing 29: HostileEntity.java

```

1 package model;
2
3 import java.awt.image.BufferedImage;
4
5 /**
6  * HostileEntity
7  * Abstract class representing any entity that is hostile to the player.
8  * Handles HP, damage logic, invincibility, and score points.
9  */
10 public abstract class HostileEntity extends GameObject {
11
12     protected int hp;
13     protected int maxHp;
14     protected int flashTimer = 0; // For hit effect
15     protected boolean isInvincible = false; // If true, entity takes no damage
16     protected int scorePoints; // Points awarded when destroyed

```

```

17     protected GameModel model;
18
19     public HostileEntity(int x, int y, int w, int h, BufferedImage image, int
20         hp, int scorePoints, GameModel model) {
21         super(x, y, w, h, image);
22         this.hp = hp;
23         this.maxHp = hp;
24         this.scorePoints = scorePoints;
25         this.model = model;
26     }
27
28     /**
29      * Handles taking damage from player projectiles.
30      * @param dmg Amount of damage to take
31      */
32     public void takeDamage(int dmg) {
33         // If dead or invincible, do nothing
34         if (isDead || isInvincible) return;
35
36         this.hp -= dmg;
37         this.flashTimer = GameConstants.FLASH_TIMER; // Set flash effect
38
39         // Check for death
40         if (this.hp <= 0) {
41             this.isDead = true;
42             GameModel.addScore(this.scorePoints); // Award points
43         }
44
45     @Override
46     public void move() {
47         if (flashTimer > 0) {
48             flashTimer--;
49         }
50     }
51
52     public int getFlashTimer() {
53         return flashTimer;
54     }
55 }
```

Listing 30: EnemySpawner.java

```
1 package model;
2
3 import java.util.Random;
4
5 /**
6  * EnemySpawner Class
7  * Manages the spawn timing for a specific type of enemy.
8  * Allows independent spawn rates for different enemies (e.g., Harpies vs
9  * Golems).
10 */
11
12 public class EnemySpawner {
13
14     private Class<? extends Minion> enemyType; // The class of the enemy to
15     spawn
16     private int baseInterval; // Frames between spawns (average)
17     private int variance; // Random variation in frames
18     private int timer; // Current countdown timer
19     private Random rand = new Random();
20
21     public EnemySpawner(Class<? extends Minion> enemyType, int baseInterval,
22     int variance) {
23         this.enemyType = enemyType;
24         this.baseInterval = baseInterval;
25         this.variance = variance;
26         resetTimer(); // Start the timer immediately
27     }
28
29     /**
30      * Updates the timer.
31      * @return true if it's time to spawn an enemy, false otherwise.
32      */
33     public boolean update() {
34         timer--;
35         if (timer <= 0) {
36             resetTimer();
37             return true;
38         }
39         return false;
40     }
41 }
```

```

37
38     private void resetTimer() {
39         // Calculate next spawn time: base +/- random variance
40         int var = (variance > 0) ? rand.nextInt(variance * 2 + 1) - variance :
41             0;
42         this.timer = baseInterval + var;
43         if (this.timer < 30) this.timer = 30; // Minimum safety limit (0.5 sec
44     }
45
46     /**
47      * Increases difficulty by reducing the spawn interval.
48      * @param multiplier Factor to multiply interval (e.g., 0.8 for 20% faster
49      * .
50      */
51
52     public void increaseDifficulty(double multiplier) {
53         this.baseInterval = (int)(this.baseInterval * multiplier);
54         this.variance = (int)(this.variance * multiplier);
55         // Prevent it from becoming too fast (e.g., limit to 60 frames)
56         if (this.baseInterval < 60) this.baseInterval = 60;
57     }
58
59 }
```

Listing 31: Boulder.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.geom.Ellipse2D;
6 import java.awt.image.BufferedImage;
7
8 /**
9  * Boulder Class
10 * A heavy projectile dropped by the StoneGolem.
11 * It has Power Level 2, meaning it destroys Arrows/Feathers but is destroyed
```

```

    by the Sun.

12  /*
13  public class Boulder extends Projectile {
14
15      private double preciseY;
16      private double velY;
17      private double gravity = GameConstants.Boulder_GRAVITY; // Accelerates
downwards
18
19      public Boulder(int x, int y) {
20          super(x, y,
21                  GameConstants.Boulder_WIDTH,
22                  GameConstants.Boulder_HEIGHT,
23                  ResourceManager.boulderImg,
24                  Alignment.ENEMY,
25                  2,
26                  GameConstants.Boulder_DAMAGE
27      );
28
29      this.preciseY = y;
30      this.velY = GameConstants.Boulder_INITIAL_SPEED; // Initial throw
speed
31  }
32
33  @Override
34  public void move() {
35      // Apply Gravity
36      velY += gravity;
37      preciseY += velY;
38      y = (int) preciseY;
39
40      // Despawn if off-screen
41      if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT) {
42          isDead = true;
43      }
44  }
45
46  @Override
47  public void draw(Graphics g) {
48      if (image != null) {

```

```

49         g.drawImage(image, x, y, width, height, null);
50     } else {
51         // Placeholder: Gray Rock
52         g.setColor(Color.GRAY);
53         g.fillOval(x, y, width, height);
54     }
55 }
56
57 @Override
58 public Shape getShape() {
59     // set hitbox to a circle
60     return new Ellipse2D.Float(x, y, width, height);
61 }
62 }
```

Listing 32: Cyclops.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 public class Cyclops extends Minion{
8
9     private double preciseY;
10    private double velY;
11    private int movementTimer;
12    private int attackTimer;
13    private Boulder myBoulder = null;
14    private boolean reachedMidScreen;
15    private boolean wingsClosed = false;
16
17    public Cyclops (int x, int y, GameModel model){
18        super(x, y,
19              GameConstants.CYCLOPS_WIDTH,
20              GameConstants.CYCLOPS_HEIGHT,
21              ResourceManager.cyclopsImg,
22              GameConstants.CYCLOPS_HP,
23              GameConstants.CYCLOPS_SCORE_POINTS,
```

```

24         model);
25     this.preciseY = y;
26     this.velY = GameConstants.CYCLOPS_YSPEED;
27     this.movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
28     this.attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
29     this.reachedMidScreen = false;
30 }
31
32 @Override
33 public void move() {
34     super.move();
35     if (movementTimer <= 0){
36         if (velY > 0){
37             // Go up
38             velY = GameConstants.CYCLOPS_YSPEED / -4;
39             movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 / 2;
40             wingsClosed = true; // STATE: closed wings
41         }
42         else {
43             // go down
44             wingsClosed = false; // STATE: opened wings
45
46         if (!reachedMidScreen) {
47             velY = GameConstants.CYCLOPS_YSPEED;
48             movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
49         } else {
50             velY = GameConstants.CYCLOPS_YSPEED / 4;
51             movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 /
52                 2;
53         }
54     }
55     this.image = wingsClosed ? ResourceManager.cyclopsImg2 :
56     ResourceManager.cyclopsImg;
57 }
58
59     preciseY += velY;
60     y = (int) preciseY;
61
62     if(y > (GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT/2 -
63 height)){
64         reachedMidScreen = true;

```

```

61    }
62
63    // CHECK IF THERE IS BOULDER OR IF IT IS DEAD
64    if (myBoulder == null || myBoulder.isDead()) {
65        // REDUCE TIMER
66        if(attackTimer > 0){
67            attackTimer--;
68        }
69        // ONCE THE TIMER IS 0 SPAWN BOULDER AND RESET TIMER
70        if (attackTimer == 0){
71            throwBoulder();
72            attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
73        }
74    }
75    movementTimer--;
76}
77
78 @Override
79 public void draw(Graphics g) {
80     BufferedImage imgToDraw;
81
82     if (image == ResourceManager.cyclopsImg){
83         imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg :
84         image;
85     } else {
86         imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg2 :
87         image;
88     }
89
90     if (image != null) {
91         g.drawImage(imgToDraw, x, y, width, height, null);
92     } else {
93         g.setColor(Color.RED);
94         g.fillRect(x, y, width, height);
95     }
96 }
97
98 private void throwBoulder(){
99     Boulder b = new Boulder (x + (width - GameConstants.Boulder_WIDTH)/2,
y + height);

```

```

98     this.myBoulder = b;
99     model.spawnEnemyProjectile(b);
100 }
101 }
```

Listing 33: Feather.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5
6 public class Feather extends Projectile {
7
8     public Feather(int x, int y) {
9         // Alignment: ENEMY
10        // Power Level: 1 (Light) -> Clashes equally with Arrows
11        // Damage: Standard Feather Damage
12        super(x, y, GameConstants.FEATHER_WIDTH, GameConstants.FEATHER_HEIGHT,
13              ResourceManager.featherImg,
14                  Alignment.ENEMY, 1, GameConstants.FEATHER_DAMAGE);
15    }
16
17    @Override
18    public void move() {
19        y += GameConstants.FEATHER_SPEED;
20        // Despawn logic
21        if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT) {
22            isDead = true;
23        }
24    }
25
26    @Override
27    public void draw(Graphics g) {
28        if (ResourceManager.featherImg != null) {
29            g.drawImage(image, x, y, width, height, null);
30        } else {
31            g.setColor(Color.MAGENTA);
32            g.fillRect(x, y, width, height);
33        }
34    }
35}
```

```
33     }
34 }
```

Listing 34: Harpy.java

```
1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 public class Harpy extends Minion {
8
9     private int velX;
10    private int velY;
11    private int fireTimer;
12    private boolean isInScreen;
13
14    public Harpy(int x, int y, GameModel model) {
15        // Pass params to parent: x, y, width, height, HP, Score Points
16        super(x, y,
17                GameConstants.HARPY_WIDTH,
18                GameConstants.HARPY_HEIGHT,
19                ResourceManager.harpyImg,
20                GameConstants.HARPY_HP,
21                GameConstants.HARPY_SCORE_POINTS,
22                model);
23
24        // --- MOVEMENT SETUP ---
25        this.velY = GameConstants.HARPY_YSPEED;
26        this.velX = GameConstants.HARPY_XSPEED;
27        this.isInScreen = false;
28
29        // Randomize direction
30        if (Math.random() < 0.5) {
31            this.velX = -this.velX;
32        }
33        resetFireTimer();
34    }
35}
```

```

36     @Override
37     public void move() {
38         super.move();
39         x += velX;
40         y += velY;
41
42         // Bounce Logic
43         if (x < 0) {
44             x = 0;
45             velX = -velX;
46         }
47         if (x > GameConstants.WINDOW_WIDTH - width) {
48             x = GameConstants.WINDOW_WIDTH - width;
49             velX = -velX;
50         }
51         if (y < GameConstants.HUD_HEIGHT && isInScreen) {
52             y = GameConstants.HUD_HEIGHT;
53             velY = -velY;
54         }
55         if (y > GameConstants.HUD_HEIGHT) {
56             isInScreen = true;
57         }
58         if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
59 ) {
60             y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
61 ;
62             velY = -velY;
63         }
64
65         if(fireTimer > 0) {
66             fireTimer--;
67         }
68         if (fireTimer <= 0) {
69             throwFeather();
70             resetFireTimer();
71         }
72     }
73
74     private void throwFeather() {
75         Feather f = new Feather (x + (width - GameConstants.FEATHER_WIDTH)/2,

```

```

y + height);
model.spawnEnemyProjectile(f);
}

public void resetFireTimer() {
    int base = GameConstants.FEATHER_FIRE_INTERVAL;
    int variance = GameConstants.FEATHER_FIRE_VARIANCE;

    int randomVariation = (int)(Math.random() * (variance * 2)) - variance
;

    this.fireTimer = base + randomVariation;
}

@Override
public void draw(Graphics g) {
    BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
harpyHitImg : image;

    if (image != null) {
        if (velX < 0) {
            g.drawImage(imgToDraw, x, y, width, height, null);
        } else {
            g.drawImage(imgToDraw, x + width, y, -width, height, null);
        }
    } else {
        g.setColor(Color.RED);
        g.fillRect(x, y, width, height);
    }
}
}

```

Listing 35: Minion.java

```

1 package model;
2
3 import java.awt.image.BufferedImage;
4
5 /**
6  * Minion Class

```

```

7  * Abstract class representing basic non-boss enemies.
8  * Used to group Harpies, Golems, etc., and manage shared logic like scoring.
9  */
10 public abstract class Minion extends HostileEntity {
11     public Minion(int x, int y, int w, int h, BufferedImage image, int hp, int
12         scorePoints, GameModel model) {
13         super(x, y, w, h, image, hp, scorePoints, model);
14     }
}

```

Listing 36: GamePanel.java

```

1 package view;
2
3 import model.*;
4 import java.awt.*;
5 import java.awt.event.KeyEvent;
6 import java.awt.event.KeyListener;
7 import javax.swing.*;
8 import java.awt.image.BufferedImage;
9
10 // --- Main Class (View and Controller simplified) ---
11 public class GamePanel extends JPanel implements KeyListener {
12     private GameModel model;
13     private Timer timer;
14
15     // Variables to track button states
16     private boolean leftPressed = false;
17     private boolean rightPressed = false;
18     private boolean upPressed = false;
19     private boolean downPressed = false;
20
21     private long startTime; // Game start time
22     private long endTime; // Game end time
23
24     public GamePanel() {
25         model = new GameModel(); // Initial state is TITLE
26
27         // Update preferred size to include the new BOTTOM_HUD_HEIGHT
28         this.setPreferredSize(new Dimension(GameConstants.WINDOW_WIDTH,

```

```

        GameConstants.WINDOW_HEIGHT));
29         this.setBackground(Color.BLACK);
30         this.setFocusable(true);
31         this.addKeyListener(this);
32
33         // Game Loop Timer
34         timer = new Timer(1000/GameConstants.FPS, e -> {
35             model.update();
36             repaint();
37         });
38         timer.start();
39     }
40
41     @Override
42     protected void paintComponent(Graphics g) {
43         super.paintComponent(g);
44
45         // Switch drawing based on game state
46         GameState state = model.getState();
47
48         if (state == GameState.TITLE) {
49             drawTitleScreen(g);
50         } else if (state == GameState.PLAYING || state == GameState.PAUSED ||
51         state == GameState.MESSAGE) {
52             drawGameScreen(g);
53             if (state == GameState.PAUSED){
54                 drawPauseScreen(g);
55             }
56             else if (state == GameState.MESSAGE) {
57                 drawMessageScreen(g);
58             }
59
60         } else if (state == GameState.GAMEOVER) {
61             drawGameScreen(g); // Draw game screen in background
62             drawGameOverScreen(g);
63         }
64     }
65
66     // Helper method to set font easily
67     private void setPixelFont(Graphics g, float size) {

```

```

67     if (ResourceManager.pixelFont != null) {
68         g.setFont(ResourceManager.pixelFont.deriveFont(size));
69     } else {
70         g.setFont(new Font("Arial", Font.BOLD, (int)size));
71     }
72 }
73
74 // Main Game Drawing Method
75 private void drawGameScreen(Graphics g) {
76
77     // 1. Draw Background
78     if (model.getBackground() != null) {
79         model.getBackground().draw(g);
80     } else {
81         // Fallback: Black background if image is missing
82         g.setColor(Color.BLACK);
83         // Fill only the game field area
84         g.fillRect(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH
85         , GameConstants.FIELD_HEIGHT);
86     }
87
88     // 2. Draw Game Objects (Player, Enemies, Projectiles)
89     for (GameObject obj : model.getObjects()) {
90         // Invincibility flashing logic for Player
91         if (obj instanceof Player && model.isInvincible()) {
92             // Toggle visibility every 100ms
93             if (System.currentTimeMillis() % 200 < 100) {
94                 continue;
95             }
96             obj.draw(g);
97         }
98
99     // 3. Draw Top HUD (Score, Stage, Lives)
100    drawTopHUD(g);
101
102    // 4. Draw Bottom HUD (Ability Slots) -> NEW!
103    drawBottomHUD(g);
104 }
105

```

```

106     // Helper method to draw the Top HUD
107     private void drawTopHUD(Graphics g) {
108         // Draw dark background bar
109         g.setColor(new Color(50, 50, 80));
110         g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.HUD_HEIGHT)
111 ;
112
113         // Draw white separator line
114         g.setColor(Color.WHITE);
115         g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH,
116 GameConstants.HUD_HEIGHT);
117
118         // Font settings
119         setPixelFont(g, 18f);
120         int textY = 35;
121
122
123         // A. Score
124         g.drawString("SCORE:" + model.getScore(), 10, textY);
125
126
127         // B. Stage (Centered)
128         String stageText = model.getStageText();
129         int stageX = (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
130 stringWidth(stageText)) / 2;
131         g.drawString(stageText, stageX, textY);
132
133
134         // C. Hearts / Lives (Right aligned)
135         int maxLives = GameConstants.PLAYER_MAX_LIVES;
136         int currentLives = model.getLives();
137         int heartSize = 32;
138         int spacing = 8;
139         int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (heartSize
+ spacing));
140         int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
141
142
143         for (int i = 0; i < maxLives; i++) {
144             // Determine which icon to draw (Full or Empty)
145             BufferedImage icon = (i < currentLives) ? ResourceManager.
146 heartFullImg : ResourceManager.heartEmptyImg;
147
148             if (icon != null) {

```

```

141         g.drawImage(icon, startX + (i * (heartSize + spacing)), heartY
142 , heartSize, heartSize, null);
143     } else {
144         // Fallback drawing if images are missing
145         g.setColor(i < currentLives ? Color.RED : Color.GRAY);
146         g.fillOval(startX + (i * (heartSize + spacing)), heartY,
147 heartSize, heartSize);
148     }
149 }
150
151 // Helper method to draw the Bottom HUD (Ability Slots)
152 // Helper method to draw the Bottom HUD (Ability Slots)
153 private void drawBottomHUD(Graphics g) {
154     // Calculate start Y position (below the game field)
155     int startY = GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT;
156     int height = GameConstants.BOTTOM_HUD_HEIGHT;
157
158     // 1. Background Bar
159     g.setColor(new Color(50, 50, 80));
160     g.fillRect(0, startY, GameConstants.WINDOW_WIDTH, height);
161
162     // 2. Separator Line
163     g.setColor(Color.WHITE);
164     g.drawLine(0, startY, GameConstants.WINDOW_WIDTH, startY);
165
166     // 3. Draw Slots
167     int slotSize = 60;
168     int gap = 40;
169     int totalWidth = (3 * slotSize) + (2 * gap);
170     int startX = (GameConstants.WINDOW_WIDTH - totalWidth) / 2;
171     int slotY = startY + (height - slotSize) / 2 - 9;
172
173     setPixelFont(g, 14f);
174
175     for (int i = 0; i < 3; i++) {
176         int x = startX + (i * (slotSize + gap));
177
178         // A. Draw Slot Background (Black)
179         g.setColor(Color.BLACK);

```

```

179     g.fillRect(x, slotY, slotSize, slotSize);

180

181     // Check if this is the first slot (Index 0) AND if Ability 1 is
182     // unlocked
183     if (i == 0 && model.isAbilityUnclocked(1)) {

184         // 1. Draw the Icon (The Sun)
185         if (ResourceManager.sunImg != null) {
186             g.drawImage(ResourceManager.sunImg, x, slotY, slotSize,
187             slotSize, null);
188         }

189         // 2. Draw Cooldown Overlay (The fading effect)
190         int timer = model.getAbilityNthTimer(1);
191         int maxTime = GameConstants.ABILITY1TIMER; // Make sure this
is set correctly in Constants!

192         if (timer > 0) {
193             // Calculate percentage of time remaining (0.0 to 1.0)
194             float ratio = (float) timer / maxTime;

195             // Calculate height of the dark overlay based on the ratio
196             // If ratio is 1.0 (just used), height is full (60).
197             // If ratio is 0.5, height is half (30), covering the top
198             // half.
199             // This creates the effect of the color "filling up from
bottom".
200             int overlayHeight = (int) (slotSize * ratio);

201             // Set color to semi-transparent black
202             g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (
transparency)

203             // Draw the overlay from the top of the slot downwards
204             g.fillRect(x, slotY, slotSize, overlayHeight);

205             // Optional: Draw the text timer on top if you want
206             g.setColor(Color.WHITE);
207             String keyNum = String.valueOf(timer/60 + 1);
208             int numWidth = g.getFontMetrics().stringWidth(keyNum);

```

```

213             g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY
214             + 37);
215         }
216     }
217
218     // Check if this is the first slot (Index 0) AND if Ability 1 is
219     // unlocked
220     if (i == 1 && model.isAbilityUnclocked(2)) {
221
222         // 1. Draw the Icon (The Sun)
223         if (ResourceManager.lightingImg != null) {
224             g.drawImage(ResourceManager.lightingImg, x, slotY,
225             slotSize, slotSize, null);
226         }
227
228         // 2. Draw Cooldown Overlay (The fading effect)
229         int timer = model.getAbilityNthTimer(2);
230         int maxTime = GameConstants.ABILITY2TIMER; // Make sure this
231         is set correctly in Constants!
232
233         if (timer > 0) {
234             // Calculate percentage of time remaining (0.0 to 1.0)
235             float ratio = (float) timer / maxTime;
236
237             // Calculate height of the dark overlay based on the ratio
238             // If ratio is 1.0 (just used), height is full (60).
239             // If ratio is 0.5, height is half (30), covering the top
240             // half.
241             // This creates the effect of the color "filling up from
242             // bottom".
243             int overlayHeight = (int) (slotSize * ratio);
244
245             // Set color to semi-transparent black
246             g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (
247             transparency)
248
249             // Draw the overlay from the top of the slot downwards
250             g.fillRect(x, slotY, slotSize, overlayHeight);
251
252             // Optional: Draw the text timer on top if you want

```

```

246         g.setColor(Color.WHITE);
247         String keyNum = String.valueOf(timer/60 + 1);
248         int numWidth = g.getFontMetrics().stringWidth(keyNum);
249         g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY
250             + 37);
251     }
252 }
// -----
253
254 // B. Draw Slot Border
255 g.setColor(Color.GRAY);
256 g.drawRect(x, slotY, slotSize, slotSize);
257
258 // C. Draw Key Number (1, 2, 3)
259 g.setColor(Color.WHITE);
260 String keyNum = String.valueOf(i + 1);
261 int numWidth = g.getFontMetrics().stringWidth(keyNum);
262 g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY +
263 slotSize + 25);
264 }
265
266 // Draw Title Screen
267 private void drawTitleScreen(Graphics g) {
268     g.setColor(Color.WHITE);
269
270     // Title: Big Pixel Font
271     setPixelFont(g, 28f);
272     String title = "GLADIATOR GAME"; // Cambia il nome se vuoi
273     int titleWidth = g.getFontMetrics().stringWidth(title);
274     g.drawString(title, (GameConstants.WINDOW_WIDTH - titleWidth)/2, 250);
275
276     // Subtitle: Smaller
277     setPixelFont(g, 15f);
278     String msg = "Press SPACE to Start";
279     int msgWidth = g.getFontMetrics().stringWidth(msg);
280     g.drawString(msg, (GameConstants.WINDOW_WIDTH - msgWidth)/2, 350);
281 }
282
283 private void drawPauseScreen(Graphics g) {

```

```

284     // 1. Semi-transparent black overlay
285     g.setColor(new Color(0, 0, 0, 150)); // 150 = Alpha (Transparency)
286     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
287     WINDOW_HEIGHT);
288
289     // 2. "PAUSE" Text
290     g.setColor(Color.WHITE);
291     setPixelFont(g, 40f); // Large font
292     String pauseText = "PAUSE";
293     int pauseWidth = g.getFontMetrics().stringWidth(pauseText);
294     // Center text
295     g.drawString(pauseText, (GameConstants.WINDOW_WIDTH - pauseWidth) / 2,
296     GameConstants.WINDOW_HEIGHT / 2 - 100);
297
298     // 3. Instruction Text
299     setPixelFont(g, 20f); // Smaller font
300     String resumeText = "Press [P] to Resume";
301     int resumeWidth = g.getFontMetrics().stringWidth(resumeText);
302     g.drawString(resumeText, (GameConstants.WINDOW_WIDTH - resumeWidth) /
303     2, GameConstants.WINDOW_HEIGHT / 2 - 50);
304 }
305
306 private void drawMessageScreen(Graphics g) {
307     // 1. Semi-transparent black background for the whole screen (dimming)
308     g.setColor(new Color(0, 0, 0, 100));
309     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
310     WINDOW_HEIGHT);
311
312     // 2. The Message Box Dimensions
313     int boxWidth = 500;
314     int boxHeight = 400;
315     int boxX = (GameConstants.WINDOW_WIDTH - boxWidth) / 2;
316     int boxY = (GameConstants.WINDOW_HEIGHT - boxHeight) / 2;
317
318     // 3. Draw the Box Background (Dark Blue)
319     g.setColor(new Color(20, 20, 80));
320     g.fillRect(boxX, boxY, boxWidth, boxHeight);
321
322     // 4. Draw the Box Border (White)
323     g.setColor(Color.WHITE);

```

```

320     Graphics2D g2 = (Graphics2D) g;
321     g2.setStroke(new BasicStroke(4)); // Thicker border
322     g2.drawRect(boxX, boxY, boxWidth, boxHeight);
323
324     // 5. Draw the Text
325     String[] lines = model.getCurrentMessageLines();
326     if (lines != null) {
327         setPixelFont(g, 20f); // Size for text
328         g.setColor(Color.WHITE);
329
330         int lineHeight = 30;
331         // Calculate starting Y to center the block of text vertically
332         int totalTextHeight = lines.length * lineHeight;
333         int startTextY = boxY + (boxHeight - totalTextHeight) / 2 + 10; // +10 adjustment
334
335         for (int i = 0; i < lines.length; i++) {
336             String line = lines[i];
337             // Center align each line horizontally
338             int lineWidth = g.getFontMetrics().stringWidth(line);
339             int lineX = (GameConstants.WINDOW_WIDTH - lineWidth) / 2;
340
341             g.drawString(line, lineX, startTextY + (i * lineHeight));
342         }
343     }
344
345     // 6. Draw "Press Space" prompt at the bottom of the box
346     setPixelFont(g, 14f);
347     g.setColor(Color.YELLOW);
348     String prompt = "PRESS [SPACE] TO CONTINUE";
349     int promptWidth = g.getFontMetrics().stringWidth(prompt);
350     g.drawString(prompt, (GameConstants.WINDOW_WIDTH - promptWidth) / 2,
351     boxY + boxHeight - 20);
352
353     // Draw Game Over Screen
354     private void drawGameOverScreen(Graphics g) {
355         // Semi-transparent overlay
356         g.setColor(new Color(0, 0, 0, 150));
357         g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.

```

```

        WINDOW_HEIGHT);

358
    // Game Over Text
359    g.setColor(Color.RED);
360    setPixelFont(g, 35f); // Big Red Text
361    String GO = "GAME OVER";
362    int goWidth = g.getFontMetrics().stringWidth(GO);
363    g.drawString(GO, (GameConstants.WINDOW_WIDTH - goWidth)/2, 250);

365
366    g.setColor(Color.WHITE);
367    setPixelFont(g, 20f);

368
369    String scoreMsg = "Final Score: " + model.getScore();
370    int scoreWidth = g.getFontMetrics().stringWidth(scoreMsg);
371    g.drawString(scoreMsg, (GameConstants.WINDOW_WIDTH - scoreWidth)/2,
372 320);

373    // ... Time e Quit/Continue (usa la stessa logica per centrare) ...
374    String cont = "[C] Continue";
375    String quit = "[Q] Quit";
376
377    g.drawString(cont, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
378 stringWidth(cont))/2, 400);
379    g.drawString(quit, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
380 stringWidth(quit))/2, 440);
381 }

382 // Update Player Velocity based on key states
383 private void updatePlayerVelocity() {
384     Player p = model.getPlayer();
385
386     int vx = 0;
387     int vy = 0;
388
389     if (leftPressed && !rightPressed) vx = -1;
390     if (rightPressed && !leftPressed) vx = 1;
391     if (upPressed && !downPressed) vy = -1;
392     if (downPressed && !upPressed) vy = 1;
393
394     p.setVelX(vx);

```

```

394     p.setVelY(vy);
395 }
396
397 // Reset keys when restarting game
398 private void resetKeyState() {
399     leftPressed = false;
400     rightPressed = false;
401     upPressed = false;
402     downPressed = false;
403
404     Player p = model.getPlayer();
405     if (p != null) {
406         p.setVelX(0);
407         p.setVelY(0);
408     }
409 }
410
411 @Override
412 public void keyPressed(KeyEvent e) {
413     int key = e.getKeyCode();
414     GameState state = model.getState();
415
416     // Title Screen Input
417     if (state == GameState.TITLE) {
418         if (key == KeyEvent.VK_SPACE) {
419             model.initGame(); // Start Game
420             startTime = System.currentTimeMillis();
421             endTime = 0;
422         }
423     }
424
425     // Playing State Input
426     else if (state == GameState.PLAYING) {
427         if (key == KeyEvent.VK_LEFT) leftPressed = true;
428         if (key == KeyEvent.VK_RIGHT) rightPressed = true;
429         if (key == KeyEvent.VK_UP) upPressed = true;
430         if (key == KeyEvent.VK_DOWN) downPressed = true;
431
432         if (key == KeyEvent.VK_SPACE) {
433             model.setFiring(true);

```

```

434     }
435
436     if (key == KeyEvent.VK_P) {
437         model.setState(GameState.PAUSED);
438         resetKeyState();
439         System.out.println("Game Paused");
440     }
441
442     updatePlayerVelocity();
443
444     // Placeholder for Abilities
445     // ABILITY 1
446     if(model.getCurrentLevelIndex() > 3 && key == KeyEvent.VK_1) {
447         model.ability1();
448     }
449
450     // ABILITY 2
451     if (model.getCurrentLevelIndex() > 7 && key == KeyEvent.VK_2) {
452         model.ability2();
453     }
454
455     // ABILITY 3
456     if (key == KeyEvent.VK_3) System.out.println("Ability 3 pressed");
457 }
458
459     else if (state == GameState.PAUSED) {
460         if (key == KeyEvent.VK_P) {
461             model.setState(GameState.PLAYING);
462             resetKeyState();
463             System.out.println("Game Resumed");
464         }
465     }
466
467     else if (state == GameState.MESSAGE) {
468         if (key == KeyEvent.VK_SPACE) {
469             model.resumeGame(); // Go back to Playing
470             resetKeyState();
471         }
472     }
473 }
```

```

474     // Game Over State Input
475     else if (state == GameState.GAMEOVER) {
476         if (key == KeyEvent.VK_C) {
477             model.initGame(); // Retry
478             resetKeyState();
479             startTime = System.currentTimeMillis();
480             endTime = 0;
481         } else if (key == KeyEvent.VK_Q) {
482             System.exit(0); // Quit App
483         }
484     }
485 }
486
487 @Override
488 public void keyReleased(KeyEvent e) {
489     int key = e.getKeyCode();
490     if (key == KeyEvent.VK_LEFT) leftPressed = false;
491     if (key == KeyEvent.VK_RIGHT) rightPressed = false;
492     if (key == KeyEvent.VK_UP) upPressed = false;
493     if (key == KeyEvent.VK_DOWN) downPressed = false;
494
495     if (key == KeyEvent.VK_SPACE) {
496         model.setFiring(false);
497     }
498     if (model.getState() == GameState.PLAYING) {
499         updatePlayerVelocity();
500     }
501 }
502
503 @Override
504 public void keyTyped(KeyEvent e) {}
505 }
```

Listing 37: ResourceManager.java

```

1 package view;
2
3 import javax.imageio.ImageIO;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
```

```
6 import java.io.IOException;
7 import java.io.File;
8 import java.io.InputStream;
9
10 /**
11  * ResourceManager
12  * 起動時に一度だけ画像を読み込みメモリに保持する
13 */
14 public class ResourceManager {
15     // PLAYER
16     public static BufferedImage playerImg;
17     public static BufferedImage arrowImg;
18
19     //*****
20     // MINIONS
21     //*****
22     // HARPY
23     public static BufferedImage harpyImg;
24     public static BufferedImage harpyHitImg;
25     // HARPY's FEATHER
26     public static BufferedImage featherImg;
27     // CYCLOPS
28     public static BufferedImage cyclopsImg;
29     public static BufferedImage cyclopsImg2;
30     public static BufferedImage cyclopsHitImg;
31     public static BufferedImage cyclopsHitImg2;
32     // CYCLOPS's BOULCER
33     public static BufferedImage boulderImg;
34
35     //*****
36     // BOSSES
37     //*****
38
39     // APOLLO
40     public static BufferedImage apolloImg;
41     public static BufferedImage apolloImg2;
42     public static BufferedImage apolloHitImg;
43     // APOLLO's SUN
44     public static BufferedImage sunImg;
45     public static BufferedImage sunImg2;
```

```
46 // ZEUS
47 public static BufferedImage zeusImg;
48 public static BufferedImage zeusImg2;
49 public static BufferedImage zeusHitImg;
50 public static BufferedImage zeusHitImg2;
51 // ZEUS's LIGHTNING
52 public static BufferedImage lightingImg;
53 public static BufferedImage lightingImg2;

54
55 //*****
56 // HUD
57 //*****

58
59 // STAGES
60 public static BufferedImage stage1Img;
61 public static BufferedImage stage2Img;
62 public static BufferedImage stage3Img;
63 // HEART
64 public static BufferedImage heartFullImg;
65 public static BufferedImage heartEmptyImg;

66
67 // PIXEL FONT
68 public static Font pixelFont;

69
70 /**
71 * "res" フォルダからすべてのリソースを読み込む"
72 */
73 public static void loadImages() {
74     try {
75         System.out.println("Loading resources...");
76
77         // PLAYER
78         playerImg = loadTexture("res/player.png");
79         arrowImg = loadTexture("res/arrow.png");
80
81         //*****
82         // MINIONS
83         //*****
84         // HARPY
85         harpyImg = loadTexture("res/enemy.png");
```

```

86     harpyHitImg = createWhiteSilhouette(harpyImg);
87     // HARPY's FEATHER
88     featherImg = loadTexture("res/feather.png");
89     // CYCLOPS
90     cyclopsImg = loadTexture("res/cyclops_openedwings.png");
91     cyclopsImg2 = loadTexture("res/cyclops_closedwings.png");
92     cyclopsHitImg = createWhiteSilhouette(cyclopsImg);
93     cyclopsHitImg2 = createWhiteSilhouette(cyclopsImg2);
94     // CYCLOPS's BOULDER
95     boulderImg = loadTexture("res/boulder.png");

96
97     //*****
98     // BOSSES
99     //*****

100    // APOLLO
101    apolloImg = loadTexture("res/Apollo.png");
102    apolloImg2 = loadTexture("res/ApolloRed.png");
103    apolloHitImg = createWhiteSilhouette(apolloImg);
104    // APOLLO's SUN
105    sunImg = loadTexture("res/sun.png");
106    sunImg2 = loadTexture("res/sunRed.png");
107    // ZEUS
108    zeusImg = loadTexture("res/Zeus.png");
109    zeusImg2 = loadTexture("res/ZeusAngry.png");
110    zeusHitImg = createWhiteSilhouette(zeusImg);
111    zeusHitImg2 = createWhiteSilhouette(zeusImg2);
112    // ZEUS's LIGHTNING
113    lightingImg = loadTexture("res/lighting.png");
114    lightingImg2 = loadTexture("res/lightingAngry.png");

115
116     //*****
117     // HUD
118     //*****

119
120    // STAGES
121    stage1Img = loadTexture("res/stage1.png");
122    stage2Img = loadTexture("res/stage2.png");
123    stage3Img = loadTexture("res/stage3.png");
124    // HEART
125

```

```

126     heartFullImg = loadTexture("res/heart.png");
127     heartEmptyImg = createBlackSilhouette(heartFullImg);
128
129     // --- LOAD CUSTOM FONT ---
130     try {
131         // Load the font file from the res folder
132         InputStream is = ResourceManager.class.getClassLoader().
133             getResourceAsStream("res/PixelFont.ttf");
134
135         if (is != null) {
136             // Create the font object (default size is 1pt)
137             pixelFont = Font.createFont(Font.TRUETYPE_FONT, is);
138             System.out.println("Pixel Font loaded successfully!");
139         } else {
140             System.err.println("Error: PixelFont.ttf not found. Using
141             default font.");
142             pixelFont = new Font("Arial", Font.BOLD, 20); // Fallback
143         }
144     } catch (FontFormatException | IOException e) {
145         e.printStackTrace();
146         pixelFont = new Font("Arial", Font.BOLD, 20); // Fallback
147     }
148
149     System.out.println("All Resources loaded successfully!");
150 } catch (IOException e) {
151     System.err.println("Error: Could not load images.");
152     e.printStackTrace();
153 }
154
155 // 画像を安全に読み込むためのヘルパーメソッド
156 private static BufferedImage loadTexture(String path) throws IOException {
157     // クラスパスからリソースを探す
158     java.net.URL url = ResourceManager.class.getClassLoader().getResource(
159         path);
160
161     if (url == null) {
162         // もし getResource で見つからない場合（フォルダ構成の違いなど）、
163         // 通常のファイルパスとして読み込みを試みる（フォールバック処理）
164         try {

```

```

163         return ImageIO.read(new File(path));
164     } catch (IOException ex) {
165         throw new IOException("Image not found: " + path);
166     }
167 }
168 return ImageIO.read(url);
169 }

170 // ダメージ演出用に、透明度を維持したまま「真っ白なシルエット」を作成するメソッド
171 private static BufferedImage createWhiteSilhouette(BufferedImage original)
172 {
173     // 元の画像と同じサイズで、空の画像を作成
174     BufferedImage whiteImg = new BufferedImage(
175         original.getWidth(),
176         original.getHeight(),
177         BufferedImage.TYPE_INT_ARGB
178     );
179
180     // すべてのピクセルを走査する
181     for (int x = 0; x < original.getWidth(); x++) {
182         for (int y = 0; y < original.getHeight(); y++) {
183             int p = original.getRGB(x, y);
184
185             // アルファ値（透明度）を取得
186             int a = (p >> 24) & 0xff;
187
188             // 透明ではない部分（キャラクター部分）だけを「真っ白」に塗りつぶす
189             if (a > 0) {
190                 // ARGB: アルファ値 + R(255) + G(255) + B(255)
191                 int whiteColor = (a << 24) | (255 << 16) | (255 << 8) |
192                 255;
193                 whiteImg.setRGB(x, y, whiteColor);
194             }
195         }
196     }
197     return whiteImg;
198 }

199 private static BufferedImage createBlackSilhouette(BufferedImage original)
200 {

```

```
200     BufferedImage blackImg = new BufferedImage(original.getWidth(),
201         original.getHeight(), BufferedImage.TYPE_INT_ARGB);
202     for (int x = 0; x < original.getWidth(); x++) {
203         for (int y = 0; y < original.getHeight(); y++) {
204             int p = original.getRGB(x, y);
205             int a = (p >> 24) & 0xff; // Get Alpha
206
207             // If the pixel is not transparent, make it BLACK
208             if (a > 0) {
209                 // ARGB: Alpha + R(0) + G(0) + B(0)
210                 int blackColor = (a << 24) | (0 << 16) | (0 << 8) | 0;
211                 blackImg.setRGB(x, y, blackColor);
212             }
213         }
214     }
215 }
216 }
```

(文責：佐々木)