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 齋藤寛仁
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。敵を躱したり、矢を放って敵を迎え撃ちます。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

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）

```
    private void updatePlayerVelocity() {
        Player p = model.getPlayer();

        int vx = 0;
        int vy = 0;

        if (leftPressed && !rightPressed) vx = -1;
        if (rightPressed && !leftPressed) vx = 1;
        if (upPressed && !downPressed) vy = -1;
        if (downPressed && !upPressed) vy = 1;

        p.setVelX(vx);
        p.setVelY(vy);
    }

    // Reset keys when restarting game
    private void resetKeyState() {
        leftPressed = false;
        rightPressed = false;
        upPressed = false;
```

```
        downPressed = false;

        Player p = model.getPlayer();
        if (p != null) {
            p.setVelX(0);
            p.setVelY(0);
        }
    }
```

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にした。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）

```
    @Override
    public void keyPressed(KeyEvent e) {
        int key = e.getKeyCode();
        GameState state = model.getState();

        // Title Screen Input
        if (state == GameState.TITLE) {
            if (key == KeyEvent.VK_SPACE) {
                model.initGame(); // Start Game
                startTime = System.currentTimeMillis();
                endTime = 0;
            }
        }
```

```
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）

```
    @Override
    public void keyReleased(KeyEvent e) {
        int key = e.getKeyCode();
        if (key == KeyEvent.VK_LEFT) leftPressed = false;
        if (key == KeyEvent.VK_RIGHT) rightPressed = false;
        if (key == KeyEvent.VK_UP) upPressed = false;
        if (key == KeyEvent.VK_DOWN) downPressed = false;

        if (key == KeyEvent.VK_SPACE) {
            model.setFiring(false);
        }
        if (model.getState() == GameState.PLAYING) {
            updatePlayerVelocity();
        }
    }
```

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）

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

Listing 5: Zeus.java (一部抜粋)

```
    @Override
    public void move() {
        super.move();
        if (ability2Timer > 0 || ability1Phase){
            if (ability1Timer > 0) {
                // Update horizontal position
                x += speedX;

                // Bounce logic: If it hits the screen edges
                if (x <= 0 ) {
```

```
                    x = 0;
                    speedX = -speedX; // Reverse direction
                    ability1Timer--;
                } else if (x >= GameConstants.WINDOW_WIDTH - width) {
                    x = GameConstants.WINDOW_WIDTH - width;
                    speedX = -speedX; // Reverse direction
                    ability1Timer--;
                }
                if(shootTimer <= 0){
                    shootLighting();
                    resetShootTimer();
                }
                shootTimer--;
            } else if (ability1Pause <= 0){
                ability1();
            } else {
                ability1Pause--;
            }
        } else {
            if (!ability2Started) {
                if (Math.random() < 0.5) {
                    x = 0;
                    speedX = GameConstants.ZEUS_SPEED2;
                } else {
                    x = GameConstants.WINDOW_WIDTH - width;
                    speedX = -GameConstants.ZEUS_SPEED2;
                }
                Random random = new Random();
                ability2Repetitions = random.nextInt(3) + 1;
                ability2Started = true;
            }
            x += speedX;

            if (((x < (GameConstants.PLAYER_WIDTH - 20)) && (speedX < 0)) || (
        x > (GameConstants.WINDOW_WIDTH - (width + GameConstants.PLAYER_WIDTH)) &&
         speedX > 0)){
            } else {
                shootLighting();
            }

```

```
49              if (x <= 0 ) {
50                  x = 0;
51                  speedX = -speedX; // Reverse direction
52                  ability2Repetitions--;
53              } else if (x >= GameConstants.WINDOW_WIDTH - width) {
54                  x = GameConstants.WINDOW_WIDTH - width;
55                  speedX = -speedX; // Reverse direction
56                  ability2Repetitions--;
57              }
58              if (ability2Repetitions == 0){
59                  ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
60                  ability2Started = false;
61                  resetShootTimer();
62              }
63
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 ablity1,ablity2 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、ablity2 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、ablity2 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 ramdom 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。ablity2 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 ablity1 を使うのか、ablity2 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       public Lighting(int x, int y, int ZeusSpeedX, boolean isSecondPhase,
        boolean friendly, boolean ability2Started) {
5           super(0,
6                   0,
7                   GameConstants.LIGHTING_WIDTH,
8                   GameConstants.LIGHTING_HEIGHT,
9                   isSecondPhase ? ResourceManager.lightingImg2 : ResourceManager
        .lightingImg,
10                  friendly ? Alignment.PLAYER : Alignment.ENEMY,
11                  3,
12                  GameConstants.LIGHTING_DAMAGE);
13
14          this.isPlayerProjectile = friendly;
15          this.isPenetrating = true;
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 :
         x + width;
35              } else {
36                  this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
         width;
37              }
38              this.y = y + height;
```

```
39          }
40      }
41
42      @Override
43      public void move() {
44          y += velY;
45
46          if (y > GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT || y <
    GameConstants.HUD_HEIGHT - height) {
47              isDead = true;
48          }
49      }
```

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)

```
public class GamePanel extends JPanel implements KeyListener {
    private void drawTopHUD(Graphics g) {
        // Draw dark background bar
        g.setColor(new Color(50, 50, 80));
        g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.HUD_HEIGHT);

        // Draw white separator line
        g.setColor(Color.WHITE);
        g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH, GameConstants.HUD_HEIGHT);

        // Font settings
        setPixelFont(g, 18f);
        int textY = 35;

        // A. Score
        g.drawString("SCORE:" + model.getScore(), 10, textY);

        // B. Stage (Centered)
        String stageText = model.getStageText();
        int stageX = (GameConstants.WINDOW_WIDTH - g.getFontMetrics().stringWidth(stageText)) / 2;
        g.drawString(stageText, stageX, textY);

        // C. Hearts / Lives (Right aligned)
        int maxLives = GameConstants.PLAYER_MAX_LIVES;
        int currentLives = model.getLives();
        int heartSize = 32;
        int spacing = 8;
        int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (heartSize + spacing));
        int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;

        for (int i = 0; i < maxLives; i++) {
            // Determine which icon to draw (Full or Empty)
            BufferedImage icon = (i < currentLives) ? ResourceManager.heartFullImg : ResourceManager.heartEmptyImg;

            if (icon != null) {
```

```
                    g.drawImage(icon, startX + (i * (heartSize + spacing)),
    heartY, heartSize, heartSize, null);
                } else {
                    // Fallback drawing if images are missing
                    g.setColor(i < currentLives ? Color.RED : Color.GRAY);
                    g.fillOval(startX + (i * (heartSize + spacing)), heartY,
    heartSize, heartSize);
                }
            }
        }
    }
```

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 になってしまう「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)

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

        public void initGame() {
            // 追加ライフ初期化:
            lives = GameConstants.PLAYER_MAX_LIVES;
            damageTimer = 0;
        }
    }
```

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

```
    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

4 実行例

5 考察

6 感想

6.1 齋藤

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

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。

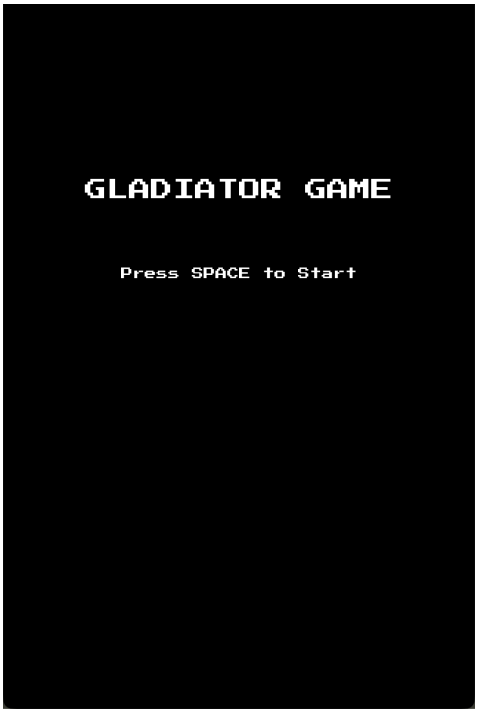

図 2: タイトル画面

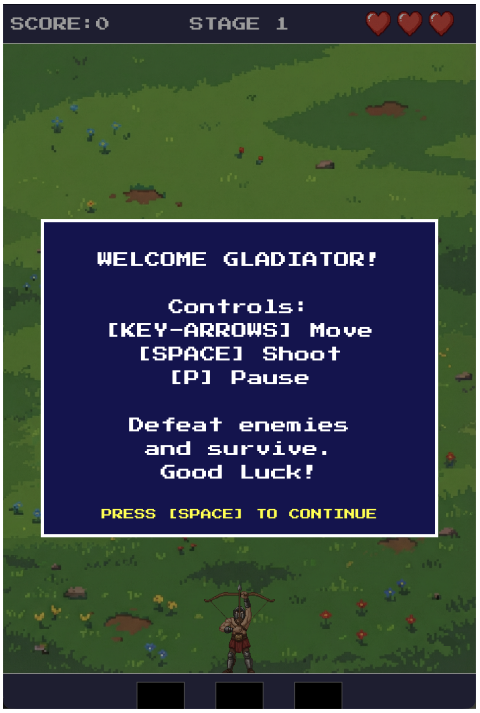

図 3: スタート画面

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 4: 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 10: 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
```

```
        // 2. Setup the game window
        JFrame frame = new JFrame("Shooting Game MVC");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new GamePanel());
        frame.pack();
        frame.setLocationRelativeTo(null); // ウィンドウを画面中央に
        frame.setVisible(true);
    }
}
```

Listing 11: GameModel.java

```
package model;

import view.ResourceManager;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.awt.geom.Area;

// --- GameModel (M) ---
public class GameModel {
    private ArrayList<GameObject> objects;
    private Player player;
    private boolean isGameOver = false;
    private Random rand = new Random();
    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）
    private GameState state; // 現在のゲーム状態
    private boolean isFiring; // スペースキーが押されているか
    private int shotTimer; // 連射間隔を制御するタイマー
    private int arrowDamage;
    private int arrowInterval;

    // Score progression
    private static int score = 0; //スコアの導入
    private int nextTargetScore;
    private int currentLevelIndex = 0;
    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.
        PLAYER_WIDTH)/2, GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT -
        GameConstants.PLAYER_HEIGHT);
60          objects.add(player);
61
62          // Initialize Background
63          background = new Background();
64
65          isFiring = false;
66          shotTimer = 0;
```

```
67          arrowDamage = GameConstants.ARROW_DAMAGE;
68          arrowInterval = GameConstants.ARROW_INTERVAL;
69          state = GameState.PLAYING;
70          score = 0; //スコアをリセット
71
72          // 追加ライフ初期化:
73          lives = GameConstants.PLAYER_MAX_LIVES;
74          damageTimer = 0;
75
76          //Score progression resets
77          this.currentLevelIndex = 0;
78          this.nextTargetScore = GameConstants.LEVEL_MILESTONES[
    currentLevelIndex];
79          isBossActive = false;
80
81          // STAGE 1 SETUP: Add Harpy Spawner
82          activeSpawners.add(new EnemySpawner(Harpy.class, GameConstants.
    HARPY_SPAWN_INTERVAL, GameConstants.HARPY_SPAWN_VARIANCE));
83          this.currentStage = 1;
84
85          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ったら return
209         if (currentLevelIndex >= GameConstants.LEVEL_MILESTONES.length) return
    ;
210
211         if (score >= nextTargetScore) {
212             //apply the effect we decided in the applyLevelEffects function
213             applyLevelEffects(currentLevelIndex);
214             currentLevelIndex++;
215             System.out.println("The current level index is: " +
    currentLevelIndex);
216             nextTargetScore = GameConstants.LEVEL_MILESTONES[currentLevelIndex
    ];
217         }
218     }
```

```
219
220     private void applyLevelEffects(int levelIndex) {
221         switch (levelIndex) {
222             case 0: //START OF THE GAME
223                 System.out.println("Start of the Game");
224                 break;
225
226             case 1:
227                 System.out.println("Difficulty UP!");
228                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
    (0.8);
229                 break;
230
231             case 2:
232                 System.out.println("Difficulty UP!");
233                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
    (0.8);
234                 break;
235
236             case 3:
237                 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\
    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                 // Cyclops Spawner
261                 activeSpawners.add(new EnemySpawner(Cyclops.class,
262                         GameConstants.CYCLOPS_SPAWN_INTERVAL,
263                         GameConstants.CYCLOPS_SPAWN_VARIANCE));
264
265             }
266             break;
267         case 5:
268             System.out.println("Difficulty UP!");
269             for (EnemySpawner s : activeSpawners) s.increaseDifficulty
    (0.9);
270             break;
271         case 6:
272             System.out.println("Difficulty UP!");
273             for (EnemySpawner s : activeSpawners) s.increaseDifficulty
    (0.9);
274             break;
275         case 7:
276             showMessage("WARNING!\n\nBOSS DETECTED:\nZEUS\n\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\
    nShooting 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                     // Cyclops Spawner
294                     activeSpawners.add(new EnemySpawner(Cyclops.class,
295                             GameConstants.CYCLOPS_SPAWN_INTERVAL,
296                             GameConstants.CYCLOPS_SPAWN_VARIANCE));
297                 }
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
     Projectile);
319         newObjectsBuffer.removeIf(obj -> obj instanceof HostileEntity || obj
    instanceof Projectile);
320     }
321
322     // プレイヤーが撃つ（から呼ばれる）Controller
323     public void playerShoot() {
324         if (!isGameOver) {
325             // プレイヤーの中央上から発射
```

```
326             Arrow a = new Arrow(player.getX() + (GameConstants.PLAYER_WIDTH-
    GameConstants.ARROW_WIDTH)/2, player.getY() - GameConstants.ARROW_HEIGHT,
    arrowDamage);
327             newObjectsBuffer.add(a);
328         }
329     }
330
331     public void ability1() {
332         if (ability1Timer > 0) return;
333         ability1Timer = GameConstants.ABILITY1TIMER;
334         Sun sun = new Sun(
335                 player.getX() + GameConstants.PLAYER_WIDTH / 2,
336                 player.getY(),
337                 0, false, true
338         );          newObjectsBuffer.add(sun);
339         System.out.println("Player shoots sun");
340     }
341
342     public void ability2() {
343         if (ability2Timer > 0) return;
344         ability2Timer = GameConstants.ABILITY2TIMER;
345         Lighting l = new Lighting(
346                 player.getX() + GameConstants.PLAYER_WIDTH / 2,
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
372             if (lives <= 0) {
373                 state = GameState.GAMEOVER;
374                 System.out.println("GAME OVER");
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
    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
    damage
451                 if (p2 instanceof BossProjectile) {
452                     ((BossProjectile) p2).reduceHealth(p1.getDamage());
453                 }
454 
455             } else {
456                 // Equal power (e.g., Arrow vs Feather) -> Both destroyed
457                 p1.setDead();
458                 p2.setDead();
459             }
460             return;
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) {
472             // Sub-case A: Projectile hits Player
473             if (entity instanceof Player) {
474                 if (proj.getAlignment() == Alignment.ENEMY) {
475                     playerTakesDamage();
476                     if (!proj.isPenetrating()) proj.setDead();
477                 }
478             }
479             // Sub-case B: Projectile hits Enemy (Harpy, Apollo, Golem, etc.)
480             else if (entity instanceof HostileEntity) {
```

```
481                 HostileEntity enemy = (HostileEntity) entity;
482
483                 // Only damage if the projectile belongs to the Player
484                 if (proj.getAlignment() == Alignment.PLAYER) {
485                     enemy.takeDamage(proj.getDamage());
486
487                     // SUN CASE: If the projectile is a Sun, it also loses HP upon contact
488                     if (proj instanceof BossProjectile) {
489                         ((BossProjectile) proj).reduceHealth(1);
490                     }
491
492                     else if (!proj.isPenetrating()) {
493                         proj.setDead();
494                     }
495                 }
496             }
497             return;
498         }
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 12: 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      //HEIGTHS
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 +
        BOTTOM_HUD_HEIGHT;
12
13      // ゲームの設定
14      public static final int FPS = 60;
15
16      // 背景の設定
17      public static final double SCREEN_SPEED = 1.0;
18
19      // の設定 PLAYER
20      public static final int PLAYER_WIDTH = 90;
21      public static final int PLAYER_HEIGHT = PLAYER_WIDTH*923/721; // set to
        the image height width ration
22      public static final int PLAYER_SPEED = 8;
23      public static final int PLAYER_MAX_LIVES = 3;
24
25      //ABILITY TIMERS
26      public static final int ABILITY1TIMER = FPS * 15; // 15 seconds
27      public static final int ABILITY2TIMER = FPS * 8; // 10 seconds
28      public static final int ABILITY3TIMER = FPS * 10; // 10 seconds
29
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 /
        2;
68      public static final int CYCLOPS_MOVEMENT_TIMER = 20;
69      public static final int CYCLOPS_ATTACK_TIMER = 60;
70
71      // の設定 BOULDER
72      public static final int BOULDER_WIDTH = 100;
73      public static final int BOULDER_HEIGHT = BOULDER_WIDTH;
74      public static final double BOULDER_INITIAL_SPEED = 0;
75      public static final double BOULDER_GRAVITY = 0.3;
76      public static final int BOULDER_DAMAGE = 5;
77
78      // ************************************
79      // の設定 BOSS
80      // ************************************
81
82      // の設定 APOLLO
83      public static final int APOLLO_WIDTH = 200;
84      public static final int APOLLO_HEIGHT = APOLLO_WIDTH * 1556 / 2463;
```

```
85      public static final int APOLLO_SPEED1 = 4;
86      public static final int APOLLO_SPEED2 = APOLLO_SPEED1 * 3 / 2;
87      public static final int APOLLO_HP = 2; //50
88      public static final int APOLLO_SCORE_POINTS = 1000;
89
90      // の設定 SUN
91      public static final int SUN_WIDTH = 150;
92      public static final int SUN_HEIGHT = SUN_WIDTH;
93      public static final double SUN_SPEED1 = 6;
94      public static final double SUN_SPEED2 = SUN_SPEED1 * 3 / 2;
95      public static final int SUN_DAMAGE = 1;
96      public static final int SUN_HP = 20;
97
98      // の設定 ZEUS
99      public static final int ZEUS_SPEED = 4;
100     public static final int ZEUS_SPEED2 = 8;
101     public static final int ZEUS_WIDTH = 150;
102     public static final int ZUES_HEIGHT = 150;
103     public static final int ZEUS_HP = 10; // 100
104     public static final int ZEUS_SCORE_POINTS = 1500;
105     public static final int ZEUS_SHOOT_TIMER = 60;
106     public static final int ZEUS_SHOOT_TIMER2 = 40;
107     public static final int ZEUS_ABILITY1_PAUSE = 40;
108     public static final int ZEUS_ABILITY1_PAUSE2 = 25;
109     public static final int ZEUS_ABILITY2_TIMER = FPS * 6; // 10 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 * 1; //
        100
123     public static final int SCORE_STAGE1_PHASE3 = HARPY_SCORE_POINTS * 3; //
```

```
    300
124     public static final int SCORE_FOR_BOSS_1 = HARPY_SCORE_POINTS * 5; // 500
125     // STAGE 2
126     public static final int SCORE_STAGE2_PHASE1 = SCORE_FOR_BOSS_1 +
        APOLLO_SCORE_POINTS; //1500
127     public static final int SCORE_STAGE2_PHASE2 = SCORE_STAGE2_PHASE1 + 5; //
        2000
128     public static final int SCORE_STAGE2_PHASE3 = SCORE_STAGE2_PHASE2 + 7; //
        2750
129     public static final int SCORE_FOR_BOSS_2 = SCORE_STAGE2_PHASE3 + 7; //
        3500
130     // STAGE 3
131     public static final int SCORE_STAGE3_PHASE1 = SCORE_FOR_BOSS_2 +
        ZEUS_SCORE_POINTS; // 5000
132     public static final int SCORE_STAGE3_PHASE2 = 100000; // 6000
133     public static final int SCORE_STAGE3_PHASE3 = 300000; // 7000
134     public static final int SCORE_FOR_BOSS_3 = 500000; // 8000
135     // EXTRA STAGE
136     public static final int SCORE_EXTRA_STAGE = 10000; // 10000
137
138     // We put them all in an Array
139     public static final int[] LEVEL_MILESTONES = {
140             SCORE_STAGE1_PHASE1, SCORE_STAGE1_PHASE2, SCORE_STAGE1_PHASE3,
        SCORE_FOR_BOSS_1,
141             SCORE_STAGE2_PHASE1, SCORE_STAGE2_PHASE2, SCORE_STAGE2_PHASE3,
        SCORE_FOR_BOSS_2,
142             SCORE_STAGE3_PHASE1, SCORE_STAGE3_PHASE2, SCORE_STAGE3_PHASE3,
        SCORE_FOR_BOSS_3,
143             SCORE_EXTRA_STAGE
144     };
145
146     //他の設定
147     public static final int FLASH_TIMER = 5;
148     private GameConstants(){} //オブジェクトを作らないようににする private
149 }
```

Listing 13: 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 14: 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 15: 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 16: 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
```

```
 * Handles the infinite scrolling background using a "Mirroring" technique.
 * It draws the original image and a vertically flipped copy above it
 * to create a seamless loop without needing a specific seamless texture.
 */
public class Background {
    private double y; // Vertical position (double for smooth movement)
    private double speed; // Scrolling speed
    private BufferedImage image;

    // Screen dimensions
    private final int WIDTH = GameConstants.WINDOW_WIDTH;
    private final int HEIGHT = GameConstants.FIELD_HEIGHT / 2;

    public Background() {
        this.image = ResourceManager.stage1Img;
        this.speed = 0;
        this.y = GameConstants.HUD_HEIGHT;
    }

    public void setImage(BufferedImage newImage){
        this.image = newImage;
    }

    public void setSpeed(double newSpeed) {
        this.speed = newSpeed;
    }

    public void update() {
        // Move the background downwards
        y += speed;

        // Reset position when a full cycle is completed to prevent overflow
        if (y >= HEIGHT * 2 + GameConstants.HUD_HEIGHT) {
            y = GameConstants.HUD_HEIGHT;
        }
    }

    public void draw(Graphics g) {
        if (image == null) return;

```

```
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              // 1. Draw Normal Tile at current Y
60              // Covers: y to y+400  x
61              drawForceSize(g, currentY, false);
62
63              // 2. Draw Flipped Tile BELOW
64              // Covers: y+400 to y+800
65              drawForceSize(g, currentY + HEIGHT, true);
66
67              // 3. Draw Flipped Tile ABOVE
68              // Covers: y-400 to y. (Crucial for when y starts at 0 or is small
    )
69              drawForceSize(g, currentY - HEIGHT, true);
70
71              // 4. Draw Normal Tile ABOVE that
72              // Covers: y-800 to y-400. (Crucial for the loop wrap-around)
73              drawForceSize(g, currentY - (HEIGHT * 2), false);
74          } else {
75              g.drawImage(image, 0, GameConstants.HUD_HEIGHT, WIDTH, HEIGHT * 2,
     null);
76          }
77      }
78
79      /**
80       * Helper method to draw a tile either normally or vertically flipped.
81       * @param g Graphics context
82       * @param yPos The Y position to draw at
83       * @param isFlipped If true, draws the image upside down (mirrored)
84       */
```

```
85      private void drawForceSize(Graphics g, int yPos, boolean isFlipped) {
86          if (!isFlipped) {
87              // NORMAL: Force width 600, height 400
88              g.drawImage(image, 0, yPos, WIDTH, HEIGHT, null);
89          } else {
90              // FLIPPED: Force width 600, height 400 (but drawn upwards)
91              // Destination Y starts at bottom (yPos + HEIGHT)
92              // Height is negative (-HEIGHT) to flip it
93              g.drawImage(image, 0, yPos + HEIGHT, WIDTH, -HEIGHT, null);
94          }
95      }
96
97      private boolean isStage2Img(){
98          return (this.image == ResourceManager.stage2Img);
99      }
100 }
```

Listing 17: 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,
         Boulders).
11   * It introduces the concept of "Power Level" to handle projectile-vs-
         projectile collisions.
12   */
13  public abstract class Projectile extends GameObject {
14
15      protected Alignment alignment;
16
17      // Power Level determines priority when two projectiles collide:
18      // 0 = Ephemeral (Destroyed by anything)
19      // 1 = Light (Arrow, Feather)
```

```
    // 2 = Heavy (Boulder - Destroys Light)
    // 3 = Ultimate (Sun - Destroys Heavy and Light)
    protected int powerLevel;

    protected int damage;

    // If true, the projectile does not vanish after hitting a target (e.g., The Sun)
    protected boolean isPenetrating;

    public Projectile(int x, int y, int w, int h, BufferedImage image, Alignment alignment, int powerLevel, int damage) {
        super(x, y, w, h, image);
        this.alignment = alignment;
        this.powerLevel = powerLevel;
        this.damage = damage;
        this.isPenetrating = false; // Default: destroys itself on impact
    }

    // Getters
    public Alignment getAlignment() { return alignment; }
    public int getPowerLevel() { return powerLevel; }
    public int getDamage() { return damage; }
    public boolean isPenetrating() { return isPenetrating; }
}
```

Listing 18: Player.java

```
package model;

import view.ResourceManager; // Import the manager
import java.awt.*;
import java.awt.image.BufferedImage;
import java.awt.geom.Ellipse2D;

// --- クラス Player 自分 () ---
public class Player extends GameObject {
    private int velX = 0; // 横の移動速度
    private int velY = 0; // 縦の移動速度
    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,
        ResourceManager.playerImg); // 40の四角
        x40
17      }
18
19      @Override
20      public void move() {
21          x += velX;
22          y += velY;
23
24          // 画面からはみ出さないように制限
25          if (x < 0) x = 0;
26          if (x > GameConstants.WINDOW_WIDTH - width) x = GameConstants.
        WINDOW_WIDTH - width;
27          if (y < GameConstants.HUD_HEIGHT) y = GameConstants.HUD_HEIGHT;
28          if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
        ) y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height;
29      }
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
```

```
        // the smaller hitbox get centered compared to the original immage
        return new Ellipse2D.Float(
                x + paddingX / 2,     // move right
                y + paddingY / 2,     // move down
                width - paddingX,     // reduce width
                height - paddingY     // reduce height
        );
    }

    // から呼ばれるメソッド Controller
    public void setVelX(int vx) {
        this.velX = vx * speed;
    }

    public void setVelY(int vy) {
        this.velY = vy * speed;
    }

    public int getLevel(){
        return level;
    }
}
```

Listing 19: Arrow.java

```
package model;

import view.ResourceManager;
import java.awt.*;
import java.awt.image.BufferedImage;

public class Arrow extends Projectile {

    public Arrow(int x, int y, int arrowDamage) {
        // Alignment: PLAYER
        // Power Level: 1 (Light) -> Can be destroyed by Boulders (Lvl 2) or Sun (Lvl 3)
        // Damage: Standard Arrow Damage
        super(x, y, GameConstants.ARROW_WIDTH, GameConstants.ARROW_HEIGHT, ResourceManager.arrowImg,
```

```
                Alignment.PLAYER, 1, arrowDamage);
    }

    @Override
    public void move() {
        y -= GameConstants.ARROW_SPEED;
        if (y < -height) {
            isDead = true;
        }
    }

    @Override
    public void draw(Graphics g) {
        if (ResourceManager.arrowImg != null) {
            g.drawImage(image, x, y, width, height, null);
        } else {
            g.setColor(Color.YELLOW);
            g.fillRect(x, y, width, height);
        }
    }
}
```

Listing 20: BossProjectile.java

```
package model;

import java.awt.image.BufferedImage;

public abstract class BossProjectile extends Projectile {

    protected boolean isPlayerProjectile;
    protected int maxHP;
    protected int currentHP;

    public BossProjectile(int x, int y, int w, int h, BufferedImage image,
    Alignment alignment, int powerLevel, int damage){
        super(x, y, w, h, image, alignment, powerLevel, damage);
    }

    public void reduceHealth(int damage) {
```

```
        // Only Player's Projectiles can die
        if (!isPlayerProjectile) return;
        this.currentHP -= damage;
        if (this.currentHP <= 0) {
            this.isDead = true;
        }
    }
}
```

Listing 21: Boss.java

```
package model;

import java.awt.*;
import java.awt.image.BufferedImage;

public abstract class Boss extends HostileEntity {

    public Boss(int x, int y, int w, int h, BufferedImage image, int hp, int
    scorePoints, GameModel model) {
        super(x, y, w, h, image, hp, scorePoints, model);
    }

    @Override
    public void takeDamage(int dmg) {
        super.takeDamage(dmg);

        if (this.isDead){
            model.bossDefeated();
        }
    }
}
```

Listing 22: Apollo.java

```
package model;

import view.ResourceManager;
import java.awt.*;
import java.awt.image.BufferedImage;
```

```java

/**
 * Apollo Class
 * Represents the first Boss of the game.
 * It moves horizontally, bounces off walls, and shoots "Sun" projectiles.
 * It has two phases: Normal and Enraged (Red & Fast).
 */
public class Apollo extends Boss {

    private int speedX = GameConstants.APOLLO_SPEED1;
    private boolean secondPhase = false; // Flag to track if the boss is in "Rage Mode"

    public Apollo(GameModel model) {
        // Call the parent constructor (Boss -> HostileEntity)
        // Parameters: x, y, width, height, HP, Score Points
        super(
                (GameConstants.WINDOW_WIDTH - GameConstants.APOLLO_WIDTH) / 2, // Start in the middle
                GameConstants.HUD_HEIGHT, // Start below HUD
                GameConstants.APOLLO_WIDTH,
                GameConstants.APOLLO_HEIGHT,
                ResourceManager.apolloImg,
                GameConstants.APOLLO_HP,
                GameConstants.APOLLO_SCORE_POINTS, // Score awarded when defeated
                model
        );
        this.image = ResourceManager.apolloImg;
    }

    @Override
    public void move() {
         super.move();
        // Update horizontal position
        x += speedX;

        // Bounce logic: If it hits the screen edges
        if (x <= 0 || x >= GameConstants.WINDOW_WIDTH - width) {
            speedX = -speedX; // Reverse direction
```

```
43
44              // Trigger shooting mechanism via GameModel
45              // We pass the current phase status to decide if the Sun should be
     Red/Fast
46              shootSun();
47          }
48      }
49
50      private void shootSun(){
51          Sun s = new Sun(x, y, speedX, secondPhase, false);
52          model.spawnEnemyProjectile(s);
53      }
54
55      @Override
56      public void takeDamage(int dmg) {
57          // 1. Apply damage using the parent class logic (reduces HP, checks
        death, adds score)
58          super.takeDamage(dmg);
59
60          // 2. Specific Logic for Apollo: Check for Second Phase (Rage Mode)
61          // If HP drops below 50% and we are not yet in the second phase...
62          if (hp <= maxHp / 2 && !secondPhase) {
63              image = ResourceManager.apolloImg2; // Change sprite to Red Apollo
64              speedX = (speedX > 0) ? GameConstants.APOLLO_SPEED2 : -
        GameConstants.APOLLO_SPEED2; // Double the movement speed
65              secondPhase = true; // Activate the flag
66              System.out.println("Apollo entering Phase 2!");
67          }
68      }
69
70      @Override
71      public void draw(Graphics g) {
72          BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
        apolloHitImg : image;
73
74          if (image != null) {
75              // Directional Flipping Logic
76              if (speedX > 0) {
77                  // Moving RIGHT: Draw normally
78                  g.drawImage(imgToDraw, x, y, width, height, null);
```

```
            } else {
                // Moving LEFT: Flip the image horizontally
                // We draw starting at (x + width) and use a negative width to
     flip
                g.drawImage(imgToDraw, x + width, y, -width, height, null);
            }
        } else {
            // Fallback: Draw an Orange rectangle if images fail to load
            g.setColor(Color.ORANGE);
            g.fillRect(x, y, width, height);
        }
    }
}
```

Listing 23: Sun.java

```
package model;

import view.ResourceManager;
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.image.BufferedImage;

public class Sun extends BossProjectile {

    private double preciseX, preciseY;
    private double velX, velY;

    private double initialSize;

    public Sun(int summonerX, int summonerY, int ApolloSpeedX, boolean
    isSecondPhase, boolean friendly) {
        // Call parent constructor
        // HP = 1 (doesn't matter), Score = 0
        super(0,
                0,
                GameConstants.SUN_WIDTH,
                GameConstants.SUN_HEIGHT,
                isSecondPhase ? ResourceManager.sunImg2 : ResourceManager.
    sunImg,
```

```
                friendly ? Alignment.PLAYER : Alignment.ENEMY,
                3,
                GameConstants.SUN_DAMAGE);

        this.isPlayerProjectile = friendly;
        this.isPenetrating = true;

        if (isPlayerProjectile) {
            maxHP = GameConstants.SUN_HP;
            currentHP = maxHP;
        }

        this.initialSize = GameConstants.SUN_WIDTH;

        velX = 1;
        velY = 1;
        double currentSpeed;
        double angleRadians;

        if (isSecondPhase) {
            currentSpeed = GameConstants.SUN_SPEED2;
        } else {
            currentSpeed = GameConstants.SUN_SPEED1;
        }

        if (!friendly){
            // sun comes from apollo
            preciseX = (ApolloSpeedX > 0) ? (summonerX + GameConstants.
    APOLLO_WIDTH + width / 2) : (summonerX - width / 2);
            preciseY = summonerY + height;
            x = (int) preciseX;
            y = (int) preciseY;
            // --- Phase Logic ---
            // --- Trajectory Calculation ---
            double minAngle = 20.0;
            double maxAngle = 70;
            angleRadians = Math.toRadians(minAngle + Math.random() * (maxAngle
     - minAngle));
        } else {
            // sun comes from ability
```

```
            preciseX = summonerX;
            preciseY = summonerY - height / 2;

            // 2. right boarder check
            if (preciseX > GameConstants.WINDOW_WIDTH - width / 2) {
                preciseX = GameConstants.WINDOW_WIDTH - width / 2;
            }

            // 3. left boarder check
            if (preciseX < width / 2) {
                preciseX = width / 2;
            }

            x = (int) preciseX;
            y = (int) preciseY;
            angleRadians = Math.toRadians(360-30);
            velX = (Math.random() < 0.5) ? 1 : -1;
        }

        velX *= currentSpeed * Math.cos(angleRadians);
        velY *= currentSpeed * Math.sin(angleRadians);

        if (ApolloSpeedX < 0) {
            velX = -velX;
        }
    }

    @Override
    public void move() {
        preciseX += velX;
        preciseY += velY;

        x = (int) preciseX;
        y = (int) preciseY;

        double currentRadius = getCurrentSize() / 2.0;

        // Wall Bounce
        if (x - currentRadius < 0) {
            x = (int)currentRadius;
```

```
            preciseX = x;
            velX = -velX;
        }
        if (x + currentRadius > GameConstants.WINDOW_WIDTH) {
            x = (int)(GameConstants.WINDOW_WIDTH - currentRadius);
            preciseX = x;
            velX = -velX;
        }

        // Despawn
        if (y - currentRadius > GameConstants.FIELD_HEIGHT + GameConstants.
    HUD_HEIGHT ||
                y + currentRadius < GameConstants.HUD_HEIGHT) {
            isDead = true;
        }
    }

    // Helper to calculate size based on HP
    private double getCurrentSize() {
        if (!isPlayerProjectile) return initialSize;

        // Calculate ratio: currentHP / maxHP
        double ratio = (double) currentHP / maxHP;
        return initialSize * ratio;
    }

    @Override
    public void draw(Graphics g) {
        double size = getCurrentSize();
        double radius = size / 2.0;

        int drawX = (int) (x - radius);
        int drawY = (int) (y - radius);
        int drawSize = (int) size;

        if (image != null) {
            g.drawImage(image, drawX, drawY, drawSize, drawSize, null);
        } else {
            g.setColor(Color.YELLOW);
            g.fillOval(x, y, width, height);
```

```
        }
    }

    @Override
    public Shape getShape() {
        double size = getCurrentSize();
        double radius = size / 2.0;
        return new Ellipse2D.Float((float)(x - radius), (float)(y - radius), (
    float)size, (float)size);
    }

}
```

Listing 24: Zeus.java

```
package model;

import view.ResourceManager;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.Random;

public class Zeus extends Boss {
    private int speedX = GameConstants.ZEUS_SPEED;
    private boolean secondPhase = false; // Flag to track if the boss is in "
    Rage Mode"
    private int shootTimer;
    private int maxShootTimer;
    private int ability1Timer;
    private int ability2Timer;
    private int ability1Pause;
    private int maxAbility1Pause;
    private int ability1Position;
    private BufferedImage hitImg;
    boolean ability1Phase;
    boolean ability2Started;
    private int ability2Repetitions;

    public Zeus (GameModel model) {
        super ( (GameConstants.WINDOW_HEIGHT - GameConstants.ZEUS_WIDTH) / 2,
```

```
                GameConstants.HUD_HEIGHT,
                GameConstants.ZEUS_WIDTH,
                GameConstants.ZUES_HEIGHT,
                ResourceManager.zeusImg,
                GameConstants.ZEUS_HP,
                GameConstants.ZEUS_SCORE_POINTS,
                model
        );
        maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER;
        resetShootTimer();
        maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE;
        setAbility1Timer();
        ability1Position = 1;
        hitImg = ResourceManager.zeusHitImg;
        ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
        ability2Started = false;
    }

    @Override
    public void move() {
        super.move();
        if (ability2Timer > 0 || ability1Phase){
            if (ability1Timer > 0) {
                // Update horizontal position
                x += speedX;

                // Bounce logic: If it hits the screen edges
                if (x <= 0 ) {
                    x = 0;
                    speedX = -speedX; // Reverse direction
                    ability1Timer--;
                } else if (x >= GameConstants.WINDOW_WIDTH - width) {
                    x = GameConstants.WINDOW_WIDTH - width;
                    speedX = -speedX; // Reverse direction
                    ability1Timer--;
                }
                if(shootTimer <= 0){
                    shootLighting();
                    resetShootTimer();
                }
```

```
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)) || (
    x > (GameConstants.WINDOW_WIDTH - (width + GameConstants.PLAYER_WIDTH)) &&
     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;
```

```
                resetShootTimer();
            }

        }

        if (secondPhase && ability2Timer > 0){
            ability2Timer--;
        }
    }

    private void ability1() {
        if (ability1Position == 1){
            ability1Phase = true;
        }

        if (ability1Position == 5) {
            setAbility1Timer();
            ability1Position = 1;
            resetShootTimer();
            return;
        }

        if(speedX > 0) {
            x = ability1Position * (GameConstants.WINDOW_WIDTH-width)/4;
        } else {
            x = (4 - ability1Position) * (GameConstants.WINDOW_WIDTH-width)/4;
        }
        if (ability1Position % 2 == 0){
            y = GameConstants.HUD_HEIGHT;
        } else {
            y = GameConstants.HUD_HEIGHT + height / 3;
        }

        shootLighting();
        ability1Pause = maxAbility1Pause;
        ability1Position++;
    }

    private void shootLighting(){
        Lighting l = new Lighting(x, y, speedX, secondPhase, false,
```

```
    ability2Started);
143         model.spawnEnemyProjectile(l);
144     }
145 
146     @Override
147     public void takeDamage(int dmg) {
148         super.takeDamage(dmg);
149 
150         if (hp <= maxHp / 2 && !secondPhase) {
151             image = ResourceManager.zeusImg2;
152             hitImg = ResourceManager.zeusHitImg2;
153             speedX = (speedX > 0) ? GameConstants.ZEUS_SPEED2 : -GameConstants
    .ZEUS_SPEED2; // Double the movement speed
154             secondPhase = true;
155             maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER2;
156             maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE2;
157         }
158     }
159 
160     private void setAbility1Timer(){
161         Random random = new Random();
162         ability1Timer = random.nextInt(4) + 1;;
163         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 {
```

```
            g.setColor(Color.ORANGE);
            g.fillRect(x, y, width, height);
        }
    }
}
```

Listing 25: Lighting.java

```
package model;

import view.ResourceManager;
import java.awt.*;
import java.awt.geom.Ellipse2D;

public class Lighting extends BossProjectile {

    private int velY;
    public Lighting(int x, int y, int ZeusSpeedX, boolean isSecondPhase,
    boolean friendly, boolean ability2Started) {
        super(0,
                0,
                GameConstants.LIGHTING_WIDTH,
                GameConstants.LIGHTING_HEIGHT,
                isSecondPhase ? ResourceManager.lightingImg2 : ResourceManager
    .lightingImg,
                friendly ? Alignment.PLAYER : Alignment.ENEMY,
                3,
                GameConstants.LIGHTING_DAMAGE);

        this.isPlayerProjectile = friendly;
        this.isPenetrating = true;

        if (isSecondPhase) {
            velY = GameConstants.LIGHTING_SPEED2;
        } else {
            velY = GameConstants.LIGHTING_SPEED1;
        }

        if(isPlayerProjectile){
            velY = -velY;
```

```
            maxHP = GameConstants.LIGHTING_HP;
            currentHP = maxHP;
            // set position for player
            this.x = x + (GameConstants.PLAYER_WIDTH - width) / 2;
            this.y = y - GameConstants.PLAYER_HEIGHT;
        } else {
            if (!ability2Started){

            // set position for Zeus
            this.x = (ZeusSpeedX > 0) ? x + GameConstants.ZEUS_WIDTH - width :
 x + width;
            } else {
                this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
 width;
            }
            this.y = y + height;
        }
    }

    @Override
    public void move() {
        y += velY;

        if (y > GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT || y <
GameConstants.HUD_HEIGHT - height) {
            isDead = true;
        }
    }

    @Override
    public void draw(Graphics g) {
        if (image != null) {
            g.drawImage(image, x, y, width, height, null);
        } else {
            g.setColor(Color.YELLOW);
            g.fillOval(x, y, width, height);
        }
    }

    @Override
```

```
68      public Shape getShape() {
69          return new Ellipse2D.Float(x, y, width, height);
70      }
71  }
```

Listing 26: 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
        hp, int scorePoints, GameModel model) {
20          super(x, y, w, h, image);
21          this.hp = hp;
22          this.maxHp = hp;
23          this.scorePoints = scorePoints;
24          this.model = model;
25      }
26
27      /**
28       * Handles taking damage from player projectiles.
29       * @param dmg Amount of damage to take
30       */
31      public void takeDamage(int dmg) {
32          // If dead or invincible, do nothing
```

```
33          if (isDead || isInvincible) return;
34
35          this.hp -= dmg;
36          this.flashTimer = GameConstants.FLASH_TIMER; // Set flash effect
37
38          // Check for death
39          if (this.hp <= 0) {
40              this.isDead = true;
41              GameModel.addScore(this.scorePoints); // Award points
42          }
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 27: 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
         Golems).
9    */
10  public class EnemySpawner {
11
12      private Class<? extends Minion> enemyType; // The class of the enemy to
         spawn
```

```
13      private int baseInterval;   // Frames between spawns (average)
14      private int variance;       // Random variation in frames
15      private int timer;          // Current countdown timer
16      private Random rand = new Random();
17
18      public EnemySpawner(Class<? extends Minion> enemyType, int baseInterval,
        int variance) {
19          this.enemyType = enemyType;
20          this.baseInterval = baseInterval;
21          this.variance = variance;
22          resetTimer(); // Start the timer immediately
23      }
24
25      /**
26       * Updates the timer.
27       * @return true if it's time to spawn an enemy, false otherwise.
28       */
29      public boolean update() {
30          timer--;
31          if (timer <= 0) {
32              resetTimer();
33              return true;
34          }
35          return false;
36      }
37
38      private void resetTimer() {
39          // Calculate next spawn time: base +/- random variance
40          int var = (variance > 0) ? rand.nextInt(variance * 2 + 1) - variance :
         0;
41          this.timer = baseInterval + var;
42          if (this.timer < 30) this.timer = 30; // Minimum safety limit (0.5 sec
        )
43      }
44
45      /**
46       * Increases difficulty by reducing the spawn interval.
47       * @param multiplier Factor to multiply interval (e.g., 0.8 for 20% faster
        ).
48       */
```

```
49      public void increaseDifficulty(double multiplier) {
50          this.baseInterval = (int)(this.baseInterval * multiplier);
51          this.variance = (int)(this.variance * multiplier);
52          // Prevent it from becoming too fast (e.g., limit to 60 frames)
53          if (this.baseInterval < 60) this.baseInterval = 60;
54      }
55
56      public Class<? extends Minion> getEnemyType() {
57          return enemyType;
58      }
59  }
```

Listing 28: 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,
```

```
                2,
                GameConstants.BOULDER_DAMAGE
        );

        this.preciseY = y;
        this.velY = GameConstants.BOULDER_INITIAL_SPEED; // Initial throw
    speed
    }

    @Override
    public void move() {
        // Apply Gravity
        velY += gravity;
        preciseY += velY;
        y = (int) preciseY;

        // Despawn if off-screen
        if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT) {
            isDead = true;
        }
    }

    @Override
    public void draw(Graphics g) {
        if (image != null) {
            g.drawImage(image, x, y, width, height, null);
        } else {
            // Placeholder: Gray Rock
            g.setColor(Color.GRAY);
            g.fillOval(x, y, width, height);
        }
    }

    @Override
    public Shape getShape() {
        // set hitbox to a circle
        return new Ellipse2D.Float(x, y, width, height);
    }
}
```

Listing 29: Cyclops.java

```
package model;

import view.ResourceManager;
import java.awt.*;
import java.awt.image.BufferedImage;

public class Cyclops extends Minion{

    private double preciseY;
    private double velY;
    private int movementTimer;
    private int attackTimer;
    private Boulder myBoulder = null;
    private boolean reachedMidScreen;
    private boolean wingsClosed = false;

    public Cyclops (int x, int y, GameModel model){
        super(x, y,
            GameConstants.CYCLOPS_WIDTH,
            GameConstants.CYCLOPS_HEIGHT,
            ResourceManager.cyclopsImg,
            GameConstants.CYCLOPS_HP,
            GameConstants.CYCLOPS_SCORE_POINTS,
            model);
        this.preciseY = y;
        this.velY = GameConstants.CYCLOPS_YSPEED;
        this.movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
        this.attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
        this.reachedMidScreen = false;
    }

    @Override
    public void move() {
        super.move();
        if (movementTimer <= 0){
            if (velY > 0){
                // Go up
                velY = GameConstants.CYCLOPS_YSPEED / -4;
                movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 / 2;
```

```
                wingsClosed = true; // STATE: closed wings
            }
            else {
                // go down
                wingsClosed = false; // STATE: opened wings

                if (!reachedMidScreen) {
                    velY = GameConstants.CYCLOPS_YSPEED;
                    movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
                } else {
                    velY = GameConstants.CYCLOPS_YSPEED / 4;
                    movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 /
 2;
                }
            }
            this.image = wingsClosed ? ResourceManager.cyclopsImg2 :
ResourceManager.cyclopsImg;
        }
        preciseY += velY;
        y = (int) preciseY;

        if(y > (GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT/2 -
height)){
            reachedMidScreen = true;
        }

        // CHECK IF THERE IS BOULDER OR IF IT IS DEAD
        if (myBoulder == null || myBoulder.isDead()) {
            // REDUCE TIMER
            if(attackTimer > 0){
                attackTimer--;
            }
            // ONCE THE TIMER IS 0 SPAWN BOULDER AND RESET TIMER
            if (attackTimer == 0){
                throwBoulder();
                attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
            }
        }
        movementTimer--;
    }
```

```
77
78      @Override
79      public void draw(Graphics g) {
80          BufferedImage imgToDraw;
81
82          if (image == ResourceManager.cyclopsImg){
83              imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg :
    image;
84          } else {
85              imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg2 :
    image;
86          }
87
88          if (image != null) {
89              g.drawImage(imgToDraw, x, y, width, height, null);
90          } else {
91              g.setColor(Color.RED);
92              g.fillRect(x, y, width, height);
93          }
94      }
95
96      private void throwBoulder(){
97          Boulder b = new Boulder (x + (width - GameConstants.BOULDER_WIDTH)/2,
    y + height);
98          this.myBoulder = b;
99          model.spawnEnemyProjectile(b);
100     }
101 }
```

Listing 30: 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
```

```
        // Power Level: 1 (Light) -> Clashes equally with Arrows
        // Damage: Standard Feather Damage
        super(x, y, GameConstants.FEATHER_WIDTH, GameConstants.FEATHER_HEIGHT,
     ResourceManager.featherImg,
                Alignment.ENEMY, 1, GameConstants.FEATHER_DAMAGE);
    }

    @Override
    public void move() {
        y += GameConstants.FEATHER_SPEED;
        // Despawn logic
        if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT) {
            isDead = true;
        }
    }

    @Override
    public void draw(Graphics g) {
        if (ResourceManager.featherImg != null) {
            g.drawImage(image, x, y, width, height, null);
        } else {
            g.setColor(Color.MAGENTA);
            g.fillRect(x, y, width, height);
        }
    }
}
```

Listing 31: Harpy.java

```
package model;

import view.ResourceManager;
import java.awt.*;
import java.awt.image.BufferedImage;

public class Harpy extends Minion {

    private int velX;
    private int velY;
    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              y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
    ;
60              velY = -velY;
61          }
62
63          if(fireTimer > 0) {
64              fireTimer--;
65          }
66          if (fireTimer <= 0) {
67              throwFeather();
68              resetFireTimer();
69          }
70      }
71
72      private void throwFeather() {
73          Feather f = new Feather (x + (width - GameConstants.FEATHER_WIDTH)/2,
    y + height);
74          model.spawnEnemyProjectile(f);
75      }
76
77      public void resetFireTimer() {
78          int base = GameConstants.FEATHER_FIRE_INTERVAL;
79          int variance = GameConstants.FEATHER_FIRE_VARIANCE;
80
81          int randomVariation = (int)(Math.random() * (variance * 2)) - variance
    ;
82
83          this.fireTimer = base + randomVariation;
84      }
85
86      @Override
87      public void draw(Graphics g) {
```

```
88          BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
        harpyHitImg : image;
89
90          if (image != null) {
91              if (velX < 0) {
92                  g.drawImage(imgToDraw, x, y, width, height, null);
93              } else {
94                  g.drawImage(imgToDraw, x + width, y, -width, height, null);
95              }
96          } else {
97              g.setColor(Color.RED);
98              g.fillRect(x, y, width, height);
99          }
100     }
101 }
```

Listing 32: 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
        scorePoints, GameModel model) {
12         super(x, y, w, h, image, hp, scorePoints, model);
13     }
14 }
```

Listing 33: 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 ||
    state == GameState.MESSAGE) {
51              drawGameScreen(g);
52              if (state == GameState.PAUSED){
53                  drawPauseScreen(g);
54              }
55              else if (state == GameState.MESSAGE) {
56                  drawMessageScreen(g);
57              }
58
59          } else if (state == GameState.GAMEOVER) {
60              drawGameScreen(g); // Draw game screen in background
61              drawGameOverScreen(g);
62          }
63      }
64
65      // Helper method to set font easily
66      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);
```

```
            // Fill only the game field area
            g.fillRect(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH
    , GameConstants.FIELD_HEIGHT);
        }

        // 2. Draw Game Objects (Player, Enemies, Projectiles)
        for (GameObject obj : model.getObjects()) {
            // Invincibility flashing logic for Player
            if (obj instanceof Player && model.isInvincible()) {
                // Toggle visibility every 100ms
                if (System.currentTimeMillis() % 200 < 100) {
                    continue;
                }
            }
            obj.draw(g);
        }

        // 3. Draw Top HUD (Score, Stage, Lives)
        drawTopHUD(g);

        // 4. Draw Bottom HUD (Ability Slots) -> NEW!
        drawBottomHUD(g);
    }

    // Helper method to draw the Top HUD
    private void drawTopHUD(Graphics g) {
        // Draw dark background bar
        g.setColor(new Color(50, 50, 80));
        g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.HUD_HEIGHT)
    ;

        // Draw white separator line
        g.setColor(Color.WHITE);
        g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH,
    GameConstants.HUD_HEIGHT);

        // Font settings
        setPixelFont(g, 18f);
        int textY = 35;

```

```
120         // A. Score
121         g.drawString("SCORE:" + model.getScore(), 10, textY);
122 
123         // B. Stage (Centered)
124         String stageText = model.getStageText();
125         int stageX = (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
    stringWidth(stageText)) / 2;
126         g.drawString(stageText, stageX, textY);
127 
128         // C. Hearts / Lives (Right aligned)
129         int maxLives = GameConstants.PLAYER_MAX_LIVES;
130         int currentLives = model.getLives();
131         int heartSize = 32;
132         int spacing = 8;
133         int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (heartSize
    + spacing));
134         int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
135 
136         for (int i = 0; i < maxLives; i++) {
137             // Determine which icon to draw (Full or Empty)
138             BufferedImage icon = (i < currentLives) ? ResourceManager.
    heartFullImg : ResourceManager.heartEmptyImg;
139 
140             if (icon != null) {
141                 g.drawImage(icon, startX + (i * (heartSize + spacing)), heartY
    , heartSize, heartSize, null);
142             } else {
143                 // Fallback drawing if images are missing
144                 g.setColor(i < currentLives ? Color.RED : Color.GRAY);
145                 g.fillOval(startX + (i * (heartSize + spacing)), heartY,
    heartSize, heartSize);
146             }
147         }
148     }
149 
150     // Helper method to draw the Bottom HUD (Ability Slots)
151     // Helper method to draw the Bottom HUD (Ability Slots)
152     private void drawBottomHUD(Graphics g) {
153         // Calculate start Y position (below the game field)
154         int startY = GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT;
```

```
155         int height = GameConstants.BOTTOM_HUD_HEIGHT;
156 
157         // 1. Background Bar
158         g.setColor(new Color(50, 50, 80));
159         g.fillRect(0, startY, GameConstants.WINDOW_WIDTH, height);
160 
161         // 2. Separator Line
162         g.setColor(Color.WHITE);
163         g.drawLine(0, startY, GameConstants.WINDOW_WIDTH, startY);
164 
165         // 3. Draw Slots
166         int slotSize = 60;
167         int gap = 40;
168         int totalWidth = (3 * slotSize) + (2 * gap);
169         int startX = (GameConstants.WINDOW_WIDTH - totalWidth) / 2;
170         int slotY = startY + (height - slotSize) / 2 - 9;
171 
172         setPixelFont(g, 14f);
173 
174         for (int i = 0; i < 3; i++) {
175             int x = startX + (i * (slotSize + gap));
176 
177             // A. Draw Slot Background (Black)
178             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
    unlocked
182             if (i == 0 && model.isAbilityUnclocked(1)) {
183 
184                 // 1. Draw the Icon (The Sun)
185                 if (ResourceManager.sunImg != null) {
186                     g.drawImage(ResourceManager.sunImg, x, slotY, slotSize,
    slotSize, null);
187                 }
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
193             if (timer > 0) {
194                 // Calculate percentage of time remaining (0.0 to 1.0)
195                 float ratio = (float) timer / maxTime;
196
197                 // Calculate height of the dark overlay based on the ratio
198                 // If ratio is 1.0 (just used), height is full (60).
199                 // If ratio is 0.5, height is half (30), covering the top
    half.
200                 // This creates the effect of the color "filling up from
    bottom".
201                 int overlayHeight = (int) (slotSize * ratio);
202
203                 // Set color to semi-transparent black
204                 g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (
    transparency)
205
206                 // Draw the overlay from the top of the slot downwards
207                 g.fillRect(x, slotY, slotSize, overlayHeight);
208
209                 // Optional: Draw the text timer on top if you want
210                  g.setColor(Color.WHITE);
211                 String keyNum = String.valueOf(timer/60 + 1);
212                 int numWidth = g.getFontMetrics().stringWidth(keyNum);
213                 g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY
    + 37);
214             }
215         }
216
217         // Check if this is the first slot (Index 0) AND if Ability 1 is
    unlocked
218         if (i == 1 && model.isAbilityUnclocked(2)) {
219
220             // 1. Draw the Icon (The Sun)
221             if (ResourceManager.lightingImg != null) {
222                 g.drawImage(ResourceManager.lightingImg, x, slotY,
    slotSize, slotSize, null);
223             }
224
225             // 2. Draw Cooldown Overlay (The fading effect)
```

```
226                 int timer = model.getAbilityNthTimer(2);
227                 int maxTime = GameConstants.ABILITY2TIMER; // Make sure this
    is set correctly in Constants!
228
229                 if (timer > 0) {
230                     // Calculate percentage of time remaining (0.0 to 1.0)
231                     float ratio = (float) timer / maxTime;
232
233                     // Calculate height of the dark overlay based on the ratio
234                     // If ratio is 1.0 (just used), height is full (60).
235                     // If ratio is 0.5, height is half (30), covering the top
    half.
236                     // This creates the effect of the color "filling up from
    bottom".
237                     int overlayHeight = (int) (slotSize * ratio);
238
239                     // Set color to semi-transparent black
240                     g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (
    transparency)
241
242                     // Draw the overlay from the top of the slot downwards
243                     g.fillRect(x, slotY, slotSize, overlayHeight);
244
245                     // 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
    + 37);
250                 }
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 +
    slotSize + 25);
263         }
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.
    WINDOW_HEIGHT);
287
288         // 2. "PAUSE" Text
289         g.setColor(Color.WHITE);
290         setPixelFont(g, 40f); // Large font
291         String pauseText = "PAUSE";
292         int pauseWidth = g.getFontMetrics().stringWidth(pauseText);
293         // Center text
294         g.drawString(pauseText, (GameConstants.WINDOW_WIDTH - pauseWidth) / 2,
     GameConstants.WINDOW_HEIGHT / 2 - 100);
295
296         // 3. Instruction Text
297         setPixelFont(g, 20f); // Smaller font
```

```
298         String resumeText = "Press [P] to Resume";
299         int resumeWidth = g.getFontMetrics().stringWidth(resumeText);
300         g.drawString(resumeText, (GameConstants.WINDOW_WIDTH - resumeWidth) /
    2, GameConstants.WINDOW_HEIGHT / 2 - 50);
301     }
302
303     private void drawMessageScreen(Graphics g) {
304         // 1. Semi-transparent black background for the whole screen (dimming)
305         g.setColor(new Color(0, 0, 0, 100));
306         g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
    WINDOW_HEIGHT);
307
308         // 2. The Message Box Dimensions
309         int boxWidth = 500;
310         int boxHeight = 400;
311         int boxX = (GameConstants.WINDOW_WIDTH - boxWidth) / 2;
312         int boxY = (GameConstants.WINDOW_HEIGHT - boxHeight) / 2;
313
314         // 3. Draw the Box Background (Dark Blue)
315         g.setColor(new Color(20, 20, 80));
316         g.fillRect(boxX, boxY, boxWidth, boxHeight);
317
318         // 4. Draw the Box Border (White)
319         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,
    boxY + boxHeight - 20);
351     }
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 
359         // Game Over Text
360         g.setColor(Color.RED);
361         setPixelFont(g, 35f); // Big Red Text
362         String GO = "GAME OVER";
363         int goWidth = g.getFontMetrics().stringWidth(GO);
364         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,
    320);
```

```
372
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().
    stringWidth(cont))/2, 400);
378         g.drawString(quit, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
    stringWidth(quit))/2, 440);
379     }
380
381     // Update Player Velocity based on key states
382     private void updatePlayerVelocity() {
383         Player p = model.getPlayer();
384
385         int vx = 0;
386         int vy = 0;
387
388         if (leftPressed && !rightPressed) vx = -1;
389         if (rightPressed && !leftPressed) vx = 1;
390         if (upPressed && !downPressed) vy = -1;
391         if (downPressed && !upPressed) vy = 1;
392
393         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 34: 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
101             // APOLLO
```

```
102             apolloImg = loadTexture("res/Apollo.png");
103             apolloImg2 = loadTexture("res/ApolloRed.png");
104             apolloHitImg = createWhiteSilhouette(apolloImg);
105             // APOLLO's SUN
106             sunImg = loadTexture("res/sun.png");
107             sunImg2 = loadTexture("res/sunRed.png");
108             // ZEUS
109             zeusImg = loadTexture("res/Zeus.png");
110             zeusImg2 = loadTexture("res/ZeusAngry.png");
111             zeusHitImg = createWhiteSilhouette(zeusImg);
112             zeusHitImg2 = createWhiteSilhouette(zeusImg2);
113             // ZEUS's LIGHTNING
114             lightingImg = loadTexture("res/lighting.png");
115             lightingImg2 = loadTexture("res/lightingAngry.png");
116
117             //******************
118             // HUD
119             //******************
120
121             // STAGES
122             stage1Img = loadTexture("res/stage1.png");
123             stage2Img = loadTexture("res/stage2.png");
124             stage3Img = loadTexture("res/stage3.png");
125             // HEART
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().
    getResourceAsStream("res/PixelFont.ttf");
133
134                 if (is != null) {
135                     // Create the font object (default size is 1pt)
136                     pixelFont = Font.createFont(Font.TRUETYPE_FONT, is);
137                     System.out.println("Pixel Font loaded successfully!");
138                 } else {
139                     System.err.println("Error: PixelFont.ttf not found. Using
    default font.");
```

```
140                     pixelFont = new Font("Arial", Font.BOLD, 20); // Fallback
141                 }
142             } catch (FontFormatException | IOException e) {
143                 e.printStackTrace();
144                 pixelFont = new Font("Arial", Font.BOLD, 20); // Fallback
145             }
146 
147             System.out.println("All Resources loaded successfully!");
148         } catch (IOException e) {
149             System.err.println("Error: Could not load images.");
150             e.printStackTrace();
151         }
152     }
153 
154     // 画像を安全に読み込むためのヘルパーメソッド
155     private static BufferedImage loadTexture(String path) throws IOException {
156         // クラスパスからリソースを探す
157         java.net.URL url = ResourceManager.class.getClassLoader().getResource(
    path);
158 
159         if (url == null) {
160             // もし getResource で見つからない場合（フォルダ構成の違いなど)、
161             // 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）
162             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 
171     // 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
172     private static BufferedImage createWhiteSilhouette(BufferedImage original)
     {
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) |
    255;
192                     whiteImg.setRGB(x, y, whiteColor);
193                 }
194             }
195         }
196         return whiteImg;
197     }
198
199     private static BufferedImage createBlackSilhouette(BufferedImage original)
     {
200         BufferedImage blackImg = new BufferedImage(original.getWidth(),
    original.getHeight(), BufferedImage.TYPE_INT_ARGB);
201         for (int x = 0; x < original.getWidth(); x++) {
202             for (int y = 0; y < original.getHeight(); y++) {
203                 int p = original.getRGB(x, y);
204                 int a = (p >> 24) & 0xff; // Get Alpha
205
206                 // If the pixel is not transparent, make it BLACK
207                 if (a > 0) {
208                     // ARGB: Alpha + R(0) + G(0) + B(0)
209                     int blackColor = (a << 24) | (0 << 16) | (0 << 8) | 0;
210                     blackImg.setRGB(x, y, blackColor);
211                 }
212             }
213         }
214         return blackImg;
```

```
215         }
216 }
```

(文責：佐々木)