

情報理工学域 メディア情報学プログラム
プログラミング演習 最終レポート

DX2 グループ 11 Gladiator

2411593 斎藤寛仁

2411603 佐々木晃誠

2411712 Puller Stefano Daniele

2026 年 2 月 18 日

1 概要説明

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

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

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

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



図 1: プレイ画面

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

(文責：佐々木)

2 設計方針

3 プログラムの説明

3.1 斎藤担当

私が作成したのは、コントローラーと第二ボス Zeus の実装です。

3.1.1 コントローラーの実装

Listing 1: GamePanel.java (一部抜粋)

```
1  private void updatePlayerVelocity() {
2      Player p = model.getPlayer();
3
4      int vx = 0;
5      int vy = 0;
6
7      if (leftPressed && !rightPressed) vx = -1;
8      if (rightPressed && !leftPressed) vx = 1;
9      if (upPressed && !downPressed) vy = -1;
10     if (downPressed && !upPressed) vy = 1;
11
12     p.setVelX(vx);
13     p.setVelY(vy);
14 }
15
16 // Reset keys when restarting game
17 private void resetKeyState() {
18     leftPressed = false;
19     rightPressed = false;
20     upPressed = false;
```

```

21     downPressed = false;
22
23     Player p = model.getPlayer();
24     if (p != null) {
25         p.setVelX(0);
26         p.setVelY(0);
27     }
28 }
```

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

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

Listing 2: GamePanel.java (一部抜粋)

```

1  @Override
2  public void keyPressed(KeyEvent e) {
3      int key = e.getKeyCode();
4      GameState state = model.getState();
5
6      // Title Screen Input
7      if (state == GameState.TITLE) {
8          if (key == KeyEvent.VK_SPACE) {
9              model.initGame(); // Start Game
10             startTime = System.currentTimeMillis();
11             endTime = 0;
12         }
13     }
```

```

14
15     // Playing State Input
16     else if (state == GameState.PLAYING) {
17         if (key == KeyEvent.VK_LEFT) leftPressed = true;
18         if (key == KeyEvent.VK_RIGHT) rightPressed = true;
19         if (key == KeyEvent.VK_UP) upPressed = true;
20         if (key == KeyEvent.VK_DOWN) downPressed = true;
21
22         if (key == KeyEvent.VK_SPACE) {
23             model.setFiring(true);
24         }
25
26         if (key == KeyEvent.VK_P) {
27             model.setState(GameState.PAUSED);
28             resetKeyState();
29             System.out.println("Game Paused");
30         }
31
32         updatePlayerVelocity();
33
34         // Placeholder for Abilities
35         // ABILITY 1
36         if(model.getCurrentLevelIndex() > 3 && key == KeyEvent.VK_1) {
37             model.ability1();
38         }
39
40         // ABILITY 2
41         if (model.getCurrentLevelIndex() > 7 && key == KeyEvent.VK_2) {
42             model.ability2();
43         }
44
45         // ABILITY 3
46         if (key == KeyEvent.VK_3) System.out.println("Ability 3 pressed");
47     }
48
49     else if (state == GameState.PAUSED) {
50         if (key == KeyEvent.VK_P) {
51             model.setState(GameState.PLAYING);
52             resetKeyState();
53             System.out.println("Game Resumed");

```

```

54         }
55     }
56
57     else if (state == GameState.MESSAGE) {
58         if (key == KeyEvent.VK_SPACE) {
59             model.resumeGame(); // Go back to Playing
60             resetKeyState();
61         }
62     }
63
64     // Game Over State Input
65     else if (state == GameState.GAMEOVER) {
66         if (key == KeyEvent.VK_C) {
67             model.initGame(); // Retry
68             resetKeyState();
69             startTime = System.currentTimeMillis();
70             endTime = 0;
71         } else if (key == KeyEvent.VK_Q) {
72             System.exit(0); // Quit App
73         }
74     }
75 }
```

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

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

続いて、16 行目から 47 行目です。ゲームをしている時は、Player の移動だけではなく、矢を放ったり、一時停止や、特殊能力まであり、ここもこれら全て条件分岐で処理をしています。まず、移動は、キーが押されたときの処理を行っているため、押されている方向キーのところは全て true に変更しています。そして、方向キーの押されているかの処理を終えた後、Player が動かないといけないので、updatePlayerVelocity を実行し、方向キーの変更を移動という形で表せるようにした。他の動作については、動作にあったキーを押された時に、その動作のクラスを呼び出します。

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

Listing 3: GamePanel.java (一部抜粋)

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

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

3.1.2 第二ボス Zeus の実装

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

Listing 4: Zeus.java (一部抜粋)

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

Listing 5: Zeus.java (一部抜粋)

```
1  @Override
2  public void move() {
3      super.move();
4      if (ability2Timer > 0 || ability1Phase){
5          if (ability1Timer > 0) {
6              // Update horizontal position
7              x += speedX;
8
9              // Bounce logic: If it hits the screen edges
10             if (x <= 0 ) {
```

```

11         x = 0;
12         speedX = -speedX; // Reverse direction
13         ability1Timer--;
14     } else if (x >= GameConstants.WINDOW_WIDTH - width) {
15         x = GameConstants.WINDOW_WIDTH - width;
16         speedX = -speedX; // Reverse direction
17         ability1Timer--;
18     }
19     if(shootTimer <= 0){
20         shootLighting();
21         resetShootTimer();
22     }
23     shootTimer--;
24 } else if (ability1Pause <= 0){
25     ability1();
26 } else {
27     ability1Pause--;
28 }
29 } else {
30     if (!ability2Started) {
31         if (Math.random() < 0.5) {
32             x = 0;
33             speedX = GameConstants.ZEUS_SPEED2;
34         } else {
35             x = GameConstants.WINDOW_WIDTH - width;
36             speedX = -GameConstants.ZEUS_SPEED2;
37         }
38         Random random = new Random();
39         ability2Repetitions = random.nextInt(3) + 1;
40         ability2Started = true;
41     }
42     x += speedX;

43
44     if (((x < (GameConstants.PLAYER_WIDTH - 20)) && (speedX < 0)) || (
45 x > (GameConstants.WINDOW_WIDTH - (width + GameConstants.PLAYER_WIDTH)) &&
46 speedX > 0)){
47         } else {
48             shootLighting();
49         }
50     }
51 }
```

```

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 ability1, ability2 として表しています。そして、このボスは、第一段階と第二段階 (secondPhase) の二つの状態を持っています。move メソッドでは、ability2 を発動していない時は、画面の端に行ったら折り返しながら、攻撃をするという動きをしています。そして、ability2 は、画面の端から端まで素早く動きながら攻撃し続けます。この端から恥まで動く動きを random を用いてランダムに決めています。この move メソッドの工夫点としては、変数 isSecondPhase と ability2Started の用意である。ability2 が発動できる状態になったら、ability2Started を true にし、そうでなければ false にしておくことで、簡単に Zeus が ability1 を使うのか、ability2 を使うのかを判別することができる。また、同様に、isSecondPhase により、Zeus の第二段階にいるかをこれによって判別することができることで、それぞれの段階の攻撃を用意することが容易にできた。

次に、雷 (Lighting) についてです。

Listing 6: Lighting.java (一部抜粋)

```

1 public class Lighting extends BossProjectile {
2 }
```

```

3   private int vely;
4
5   public Lighting(int x, int y, int ZeusSpeedX, boolean isSecondPhase,
6     boolean friendly, boolean ability2Started) {
7     super(0,
8       0,
9       GameConstants.LIGHTING_WIDTH,
10      GameConstants.LIGHTING_HEIGHT,
11      isSecondPhase ? ResourceManager.lightingImg2 : ResourceManager
12      .lightingImg,
13      friendly ? Alignment.PLAYER : Alignment.ENEMY,
14      3,
15      GameConstants.LIGHTING_DAMAGE);
16
17
18      this.isPlayerProjectile = friendly;
19      this.isPenetrating = true;
20
21
22      if (isSecondPhase) {
23        vely = GameConstants.LIGHTING_SPEED2;
24      } else {
25        vely = GameConstants.LIGHTING_SPEED1;
26      }
27
28
29      if(isPlayerProjectile){
30        vely = -vely;
31        maxHP = GameConstants.LIGHTING_HP;
32        currentHP = maxHP;
33        // set position for player
34        this.x = x + (GameConstants.PLAYER_WIDTH - width) / 2;
35        this.y = y - GameConstants.PLAYER_HEIGHT;
36      } else {
37        if (!ability2Started){
38
39          // set position for Zeus
40          this.x = (ZeusSpeedX > 0) ? x + GameConstants.ZEUS_WIDTH - width :
41          x + width;
42        } else {
43          this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
44          width;
45        }
46        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 <
47             GameConstants.HUD_HEIGHT - height) {
48             isDead = true;
49         }
50     }

```

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

(文責：齋藤)

3.2 佐々木担当

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

1. ゲーム画面および HUD の実装

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

Listing 7: HUD の描画 (GamePanel.java)

```

1  public class GamePanel extends JPanel implements KeyListener {
2      private void drawTopHUD(Graphics g) {
3          // Draw dark background bar
4          g.setColor(new Color(50, 50, 80));
5          g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
6          HUD_HEIGHT);
7
8          // Draw white separator line
9          g.setColor(Color.WHITE);
10         g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH
11         , GameConstants.HUD_HEIGHT);
12
13         // Font settings
14         setPixelFont(g, 18f);
15         int textY = 35;
16
17         // A. Score
18         g.drawString("SCORE:" + model.getScore(), 10, textY);
19
20         // B. Stage (Centered)
21         String stageText = model.getStageText();
22         int stageX = (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
23         stringWidth(stageText)) / 2;
24         g.drawString(stageText, stageX, textY);
25
26         // C. Hearts / Lives (Right aligned)
27         int maxLives = GameConstants.PLAYER_MAX_LIVES;
28         int currentLives = model.getLives();
29         int heartSize = 32;
30         int spacing = 8;
31         int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (
32             heartSize + spacing));
33         int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
34
35         for (int i = 0; i < maxLives; i++) {
36             // Determine which icon to draw (Full or Empty)
37             BufferedImage icon = (i < currentLives) ? ResourceManager.
38             heartFullImg : ResourceManager.heartEmptyImg;
39
40             if (icon != null) {

```

```

36             g.drawImage(icon, startX + (i * (heartSize + spacing)),
37 heartY, heartSize, heartSize, null);
38         } else {
39             // Fallback drawing if images are missing
40             g.setColor(i < currentLives ? Color.RED : Color.GRAY);
41             g.fillOval(startX + (i * (heartSize + spacing)), heartY,
42 heartSize, heartSize);
43         }
44     }

```

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)

```

1  public class GameModel {
2      // 追加ライフ機能用変数:
3      private int lives; // 初期ライフ
4      private int damageTimer; // ダメージを受けた後の無敵時間（フレーム数）
5
6      public void initGame() {
7          // 追加ライフ初期化:
8          lives = GameConstants.PLAYER_MAX_LIVES;
9          damageTimer = 0;
10     }
11 }

```

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

```

1  public class GameModel {

```

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

(文責：佐々木)

3.3 Puller 担当

4 実行例

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

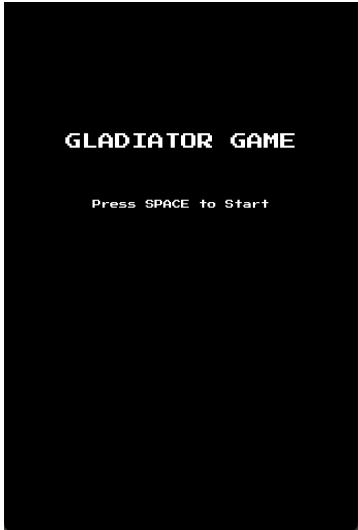


図 2: タイトル画面



図 3: スタート画面

再びスペースキーを押すと、右上の画面から左下の画面に移る。この画面にならたら、方向キーとスペースキーを使って敵を倒す。一定の敵を倒すと、右の画面に移り、第一ボスのアポロンが登場する。



図 4: 第一ステージ



図 5: 第一ボス（アポロン）登場

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



図 6: 第一ボス (アポロン)
第一フェーズ



図 7: 第一ボス (アポロン)
第二フェーズ

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



図 8: 第一ステージ クリ
ア



図 9: 第二ボス (ゼウス)
登場

左下のように、第一ボスを倒して、得られた太陽を使いながら第二ステージを攻略す

る。ステージが進むと、右下のように第二ボスゼウスが登場する。



図 10: 第二ステージ 特殊能力 1(太陽)

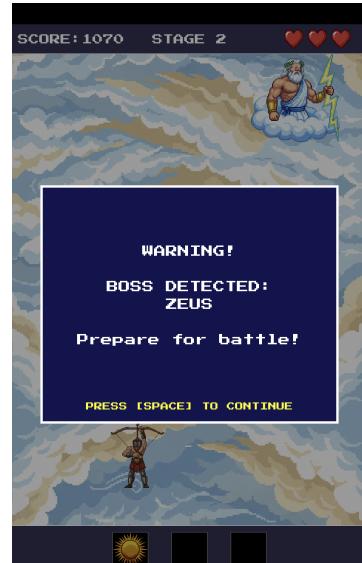


図 11: 第二ステージ

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



図 12: 第二ボス (ゼウス)
第一フェーズ



図 13: 第二ボス (ゼウス)
第二フェーズ

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



図 14: 第二ステージ クリア



図 15: 第三ステージ 特殊能力 1(太陽)

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



図 16: 第三ステージ 特殊能力 2(雷)



図 17: ゲームオーバー画面

5 考察

6 感想

6.1 斎藤

6.2 佐々木

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

開発手法としては MVC モデルを採用し、私は主に V と M の一部を担当しました。具体的には、ゲームの入り口となるタイトル画面やスタート画面、そしてプレイ画面やゲームオーバー画面といった UI 全般の実装に注力しました。単にグラフィックを表示するだけでなく、ゲームの状態に応じて動的に表示を切り替えるロジック部分の構築も経験し、UI と内部処理がどう連携すべきかを深く意識することができました。

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

(文責：佐々木)

6.3 Puller

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。

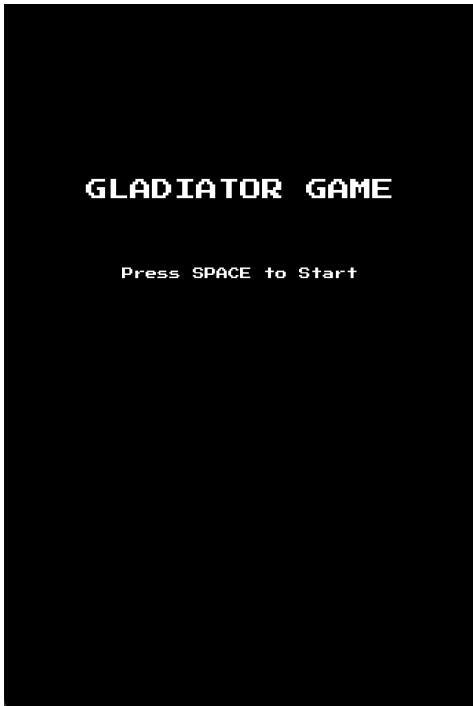


図 18: タイトル画面



図 19: 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 20: 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    }
}
```

```

14     // 2. Setup the game window
15     JFrame frame = new JFrame("Shooting Game MVC");
16     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17     frame.add(new GamePanel());
18     frame.pack();
19     frame.setLocationRelativeTo(null); // ウィンドウを画面中央に
20     frame.setVisible(true);
21 }
22 }
```

Listing 11: GameModel.java

```

1 package model;
2
3 import view.ResourceManager;
4
5 import java.awt.*;
6 import java.util.ArrayList;
7 import java.util.List;
8 import java.util.Random;
9 import java.awt.geom.Area;
10
11 // --- GameModel (M) ---
12 public class GameModel {
13     private ArrayList<GameObject> objects;
14     private Player player;
15     private boolean isGameOver = false;
16     private Random rand = new Random();
17     private ArrayList<GameObject> newObjectsBuffer; // 弾を追加するための予約リスト（ループ中のエラー回避用）
18     private GameState state; // 現在のゲーム状態
19     private boolean isFiring; // スペースキーが押されているか
20     private int shotTimer; // 連射間隔を制御するタイマー
21     private int arrowDamage;
22     private int arrowInterval;
23
24     // Score progression
25     private static int score = 0; //スコアの導入
26     private int nextTargetScore;
27     private int currentLevelIndex = 0;
28     private boolean isBossActive = false; //ボスのフェーズの確認
```

```

29
30 // 追加ライフ機能用変数:
31 private int lives; // 初期ライフ
32 private int damageTimer; // ダメージを受けた後の無敵時間（フレーム数）
33
34 private int ability1Timer;
35 private int ability2Timer;
36 private int ability3Timer;
37
38 //background
39 private Background background;
40
41 // for writing the stage number in GamePanel
42 private int currentStage;
43
44 private String[] currentMessageLines;
45
46 private List<EnemySpawner> activeSpawners;
47
48 public GameModel() {
49     objects = new ArrayList<>();
50     newObjectsBuffer = new ArrayList<>();
51     activeSpawners = new ArrayList<>();
52     state = GameState.TITLE; // 最初はタイトル画面から
53 }
54
55 public void initGame() {
56     objects.clear();
57     newObjectsBuffer.clear();
58     activeSpawners.clear();
59     player = new Player((GameConstants.WINDOW_WIDTH - GameConstants.
60     PLAYER_WIDTH)/2, GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT -
61     GameConstants.PLAYER_HEIGHT);
62     objects.add(player);
63
64     // Initialize Background
65     background = new Background();
66
67     isFiring = false;
68     shotTimer = 0;

```

```

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[
79     currentLevelIndex];
80     isBossActive = false;
81
82     // STAGE 1 SETUP: Add Harpy Spawner
83     activeSpawners.add(new EnemySpawner(Harpy.class, GameConstants.
84     HARPY_SPAWN_INTERVAL, GameConstants.HARPY_SPAWN_VARIANCE));
85     this.currentStage = 1;
86
87     String tutorial = "WELCOME GLADIATOR!\n\n" +
88         "Controls:\n" +
89         "[KEY-ARROWS] Move\n" +
90         "[SPACE] Shoot\n" +
91         "[P] Pause\n\n" +
92         "Defeat enemies\n" +
93         "and survive.\n" +
94         "Good Luck!";
95     showMessage(tutorial);
96 }
97
98 public static void addScore(int points) {
99     score += points;
100 }
101
102 public void setFiring(boolean firing) {
103     this.isFiring = firing;
104 }
105
106 public GameState getState() {
107 }

```

```
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.
183 HARPY_WIDTH); // Simple random X
184         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 * Allows enemies (Minions/Bosses) to add projectiles to the game.
196 * Using a specific method is safer than exposing the whole list.
197 */
198 public void spawnEnemyProjectile(Projectile p) {
199     if (p != null) {
200         newObjectsBuffer.add(p);
201     }
202 }
203
204
205 private void checkLevelProgression() {
206     // ボスがいたら、何もしない
207     if (isBossActive) return;
208     //次のフェーズがなかつたら return
209     if (currentLevelIndex >= GameConstants.LEVEL_MILESTONES.length) return
;
210
211     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
229                     (0.8);
230                     break;
231
232             case 2:
233                 System.out.println("Difficulty UP!");
234                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
235                     (0.8);
236                     break;
237
238             case 3:
239                 showMessage("WARNING!\n\nBOSS DETECTED:\nAPOLLO\n\nPrepare for
240 battle!");
241                 isBossActive = true;
242                 clearEverything();
243                 spawnApollo();
244                 healPlayer();
245                 break;
246             case 4:
247                 showMessage("STAGE 1 CLEARED!\n\nEntering the Heavens...\n\n
248 nArrow Damage doubled!\n\n Press [1] to use\nABILITY 1:\nAPOLLO'S SUN");
249                 if (background != null) {
250                     clearEverything();
251                     healPlayer();
252                     background.setImage(ResourceManager.stage2Img);
253                     background.setSpeed(GameConstants.SCREEN_SPEED);
254                     ability1Timer = 0;
255                     this.currentStage = 2;
256
257                     // DOUBLE ARROW DAMAGE
258                     arrowDamage*=2;

```

```

255
256         // Clear old spawners (remove Stage 1 config)
257         activeSpawners.clear();
258         // Harpy Spawner
259         activeSpawners.add(new EnemySpawner(Harpy.class, 100, 50))
260     ;
261         // Cyclops Spawner
262         activeSpawners.add(new EnemySpawner(Cyclops.class,
263                             GameConstants.CYCLOPS_SPAWN_INTERVAL,
264                             GameConstants.CYCLOPS_SPAWN_VARIANCE));
265     }
266     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\n
Shooting speed increased!\n\nPress [2] to use\nABILITY 2:\nZEUS'S
LIGHTING");
284     if (background != null) {
285         clearEverything();
286         background.setImage(ResourceManager.stage3Img);
287         background.setSpeed(0);
288         ability2Timer = 0;

```

```

289         arrowInterval = GameConstants.ARROW_INTERVAL2;
290         this.currentStage = 3;
291         activeSpawners.clear();
292         activeSpawners.add(new EnemySpawner(Harpy.class, 100, 50))
293     ;
294         // Cyclops Spawner
295         activeSpawners.add(new EnemySpawner(Cyclops.class,
296             GameConstants.CYCLOPS_SPAWN_INTERVAL,
297             GameConstants.CYCLOPS_SPAWN_VARIANCE));
298     }
299     break;
300     case 9:
301   }
302 }
303
304 public void showMessage(String text) {
305     // Split the text by newline character to handle multiple lines
306     this.currentMessageLines = text.split("\n");
307     this.state = GameState.MESSAGE;
308 }
309
310 public void bossDefeated() {
311     System.out.println("BOSS DEFEATED! Stage clear.");
312     this.isBossActive = false;
313     healPlayer();
314 }
315
316 // 全ての敵を消すメソッド
317 private void clearEverything() {
318     // 敵または羽の場合、消す!!!
319     objects.removeIf(obj -> obj instanceof HostileEntity || obj instanceof
320     Projectile);
321     newObjectsBuffer.removeIf(obj -> obj instanceof HostileEntity || obj
322     instanceof Projectile);
323 }
324
325 // プレイヤーが撃つ（から呼ばれる）Controller
326 public void playerShoot() {
327     if (!isGameOver) {
328         // 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
390     // complicated calculations
391     if (!s1.getBounds2D().intersects(s2.getBounds2D())) {
392         return false;
393     }
394
395     // 3. Precise Calculation
396     Area area1 = new Area(s1);
397     Area area2 = new Area(s2);
398
399     area1.intersect(area2);
400
401     // if the area is not empty it means they are touching
402     return !area1.isEmpty();
403 }
```

```

403
404     /**
405      * Centralized Collision Logic.
406      * Iterates through all objects to check for intersections.
407      */
408
409     private void checkCollisions() {
410         for (int i = 0; i < objects.size(); i++) {
411             GameObject objA = objects.get(i);
412             if (objA.isDead()) continue;
413
414             for (int j = i + 1; j < objects.size(); j++) {
415                 GameObject objB = objects.get(j);
416                 if (objB.isDead()) continue;
417
418                 if (checkIntersection(objA, objB)) {
419                     handleCollision(objA, objB);
420                 }
421             }
422         }
423     }
424
425     /**
426      * Handles the specific logic when two objects collide.
427      * Uses Projectile Power Levels and Alignment to determine the outcome.
428      */
429
430     private void handleCollision(GameObject a, GameObject b) {
431
432         // --- CASE 1: PROJECTILE vs PROJECTILE ---
433         if (a instanceof Projectile && b instanceof Projectile) {
434             Projectile p1 = (Projectile) a;
435             Projectile p2 = (Projectile) b;
436
437             // Same team projectiles do not destroy each other
438             if (p1.getAlignment() == p2.getAlignment()) return;
439
440             // Compare Power Levels to see who survives
441             if (p1.getPowerLevel() > p2.getPowerLevel()) {
442                 p2.setDead(); // p1 dominates
443
444                 // SUN CASE: p1 is a Sun, it takes damage equal to p2's damage

```

```

443         if (p1 instanceof BossProjectile) {
444             ((BossProjectile) p1).reduceHealth(p2.getDamage());
445         }
446
447     } else if (p2.getPowerLevel() > p1.getPowerLevel()) {
448         p1.setDead(); // p2 dominates
449
450         // SUN CASE: If p2 is a Sun, it takes damage equal to p1's
451         // damage
452         if (p2 instanceof BossProjectile) {
453             ((BossProjectile) p2).reduceHealth(p1.getDamage());
454         }
455
456     } else {
457         // Equal power (e.g., Arrow vs Feather) -> Both destroyed
458         p1.setDead();
459         p2.setDead();
460     }
461
462     return;
463 }
464
465 // --- CASE 2: PROJECTILE vs LIVING ENTITY (Player or Enemy) ---
466 Projectile proj = null;
467 GameObject entity = null;
468
469 // Identify which is which
470 if (a instanceof Projectile) { proj = (Projectile) a; entity = b; }
471 else if (b instanceof Projectile) { proj = (Projectile) b; entity = a; }
472
473 if (proj != null) {
474     // Sub-case A: Projectile hits Player
475     if (entity instanceof Player) {
476         if (proj.getAlignment() == Alignment.ENEMY) {
477             playerTakesDamage();
478             if (!proj.isPenetrating()) proj.setDead();
479         }
480     }
481     // Sub-case B: Projectile hits Enemy (Harpy, Apollo, Golem, etc.)
482     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
521     default:
522         System.out.println("getability ERROR");
523     }
524     return ability1Timer;
525 }
526
527 public boolean isAbilityUnclocked(int abilityIndex) {
528     // Logic for Ability 1 (Sun)
529     if (abilityIndex == 1) {
530         // Unlocks after defeating the first boss (Apollo)
531         // Apollo is Level Index 4. So > 4 means Stage 2 started.
532         return this.currentLevelIndex > 4;
533     }
534     // Logic for Ability 2 (Lighting)
535     if (abilityIndex == 2) {
536         // Unlocks after defeating the second boss (Zeus)
537         // Zeus is Level Index 8. So > 8 means Stage 3 started.
538         return this.currentLevelIndex > 8;
539     }
540     // Future logic for Ability 2 and 3
541     return false;
542 }
543
544 public void resumeGame() {
545     this.state = GameState.PLAYING;
546 }
547
548 //SETTERS & GETTERS
549 public ArrayList<GameObject> getObjects() {
550     return objects;
551 }
552
553 public Player getPlayer() {
554     return player;
555 }
556
557 public boolean isGameOver() {
558     return isGameOver;
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     // HEIGHTS
8     public static final int HUD_HEIGHT = 50;
9     public static final int FIELD_HEIGHT = 800;
10    public static final int BOTTOM_HUD_HEIGHT = 100;
```

```
11 public static final int WINDOW_HEIGHT = HUD_HEIGHT + FIELD_HEIGHT +
12 BOTTOM_HUD_HEIGHT;
13
14 // ゲームの設定
15 public static final int FPS = 60;
16
17 // 背景の設定
18 public static final double SCREEN_SPEED = 1.0;
19
20 // の設定 PLAYER
21 public static final int PLAYER_WIDTH = 90;
22 public static final int PLAYER_HEIGHT = PLAYER_WIDTH*923/721; // set to
the image height width ration
23 public static final int PLAYER_SPEED = 8;
24 public static final int PLAYER_MAX_LIVES = 3;
25
26 //ABILITY TIMERS
27 public static final int ABILITY1TIMER = FPS * 15; // 15 seconds
28 public static final int ABILITY2TIMER = FPS * 8; // 10 seconds
29 public static final int ABILITY3TIMER = FPS * 10; // 10 seconds
30
31 // の設定 ARROW
32 public static final int ARROW_WIDTH = 10;
33 public static final int ARROW_HEIGHT = 70; // set to the image height
width ration
34 public static final int ARROW_SPEED = 30;
35 public static final int ARROW_INTERVAL = 20;
36 public static final int ARROW_INTERVAL2 = 15;
37 public static final int ARROW_DAMAGE = 1;
38
39 // ****
40 // の設定 MINIONS
41 // ****
42
43 // の設定 HARPY
44 public static final int HARPY_WIDTH = 100;
45 public static final int HARPY_HEIGHT = HARPY_WIDTH *1911/1708; // set to
the image height width ration
46 public static final int HARPY_XSPEED = 4;
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
55     to the image height width ration
56     public static final int FEATHER_SPEED = 7;
57     public static final int FEATHER_FIRE_INTERVAL = 90;
58     public static final int FEATHER_FIRE_VARIANCE = FEATHER_FIRE_INTERVAL / 2;
59     public static final int FEATHER_DAMAGE = 1;
60
61     // の設定 CYCLOPS
62     public static final int CYCLOPS_WIDTH = 150;
63     public static final int CYCLOPS_HEIGHT = CYCLOPS_WIDTH;
64     public static final double CYCLOPS_YSPEED = 2;
65     public static final int CYCLOPS_HP = 10;
66     public static final int CYCLOPS_SCORE_POINTS = 50;
67     public static final int CYCLOPS_SPAWN_INTERVAL = 400;
68     public static final int CYCLOPS_SPAWN_VARIANCE = CYCLOPS_SPAWN_INTERVAL /
69     2;
70     public static final int CYCLOPS_MOVEMENT_TIMER = 20;
71     public static final int CYCLOPS_ATTACK_TIMER = 60;
72
73     // の設定 BOULDER
74     public static final int BOULDER_WIDTH = 100;
75     public static final int BOULDER_HEIGHT = BOULDER_WIDTH;
76     public static final double BOULDER_INITIAL_SPEED = 0;
77     public static final double BOULDER_GRAVITY = 0.3;
78     public static final int BOULDER_DAMAGE = 5;
79
80     // ****
81     // の設定 BOSS
82     // ****
83
84     // の設定 APOLLO
85     public static final int APOLLO_WIDTH = 200;
86     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 ZEUS_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; //
123    100
124    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 +
127 APOLLO_SCORE_POINTS; //1500
128 public static final int SCORE_STAGE2_PHASE2 = SCORE_STAGE2_PHASE1 + 5; //
129 2000
130 public static final int SCORE_STAGE2_PHASE3 = SCORE_STAGE2_PHASE2 + 7; //
131 2750
132 public static final int SCORE_FOR_BOSS_2 = SCORE_STAGE2_PHASE3 + 7; //
133 3500
134 // STAGE 3
135 public static final int SCORE_STAGE3_PHASE1 = SCORE_FOR_BOSS_2 +
136 ZEUS_SCORE_POINTS; // 5000
137 public static final int SCORE_STAGE3_PHASE2 = 100000; // 6000
138 public static final int SCORE_STAGE3_PHASE3 = 300000; // 7000
139 public static final int SCORE_FOR_BOSS_3 = 500000; // 8000
140 // EXTRA STAGE
141 public static final int SCORE_EXTRA_STAGE = 10000; // 10000
142
143 // We put them all in an Array
144 public static final int[] LEVEL_MILESTONES = {
145     SCORE_STAGE1_PHASE1, SCORE_STAGE1_PHASE2, SCORE_STAGE1_PHASE3,
146     SCORE_FOR_BOSS_1,
147     SCORE_STAGE2_PHASE1, SCORE_STAGE2_PHASE2, SCORE_STAGE2_PHASE3,
148     SCORE_FOR_BOSS_2,
149     SCORE_STAGE3_PHASE1, SCORE_STAGE3_PHASE2, SCORE_STAGE3_PHASE3,
150     SCORE_FOR_BOSS_3,
151     SCORE_EXTRA_STAGE
152 };
153
154 //他の設定
155 public static final int FLASH_TIMER = 5;
156 private GameConstants(){} //オブジェクトを作らないようにする private
157 }

```

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
```

```

9  * Handles the infinite scrolling background using a "Mirroring" technique.
10 * It draws the original image and a vertically flipped copy above it
11 * to create a seamless loop without needing a specific seamless texture.
12 */
13 public class Background {
14     private double y; // Vertical position (double for smooth movement)
15     private double speed; // Scrolling speed
16     private BufferedImage image;
17
18     // Screen dimensions
19     private final int WIDTH = GameConstants.WINDOW_WIDTH;
20     private final int HEIGHT = GameConstants.FIELD_HEIGHT / 2;
21
22     public Background() {
23         this.image = ResourceManager.stage1Img;
24         this.speed = 0;
25         this.y = GameConstants.HUD_HEIGHT;
26     }
27
28     public void setImage(BufferedImage newImage){
29         this.image = newImage;
30     }
31
32     public void setSpeed(double newSpeed) {
33         this.speed = newSpeed;
34     }
35
36     public void update() {
37         // Move the background downwards
38         y += speed;
39
40         // Reset position when a full cycle is completed to prevent overflow
41         if (y >= HEIGHT * 2 + GameConstants.HUD_HEIGHT) {
42             y = GameConstants.HUD_HEIGHT;
43         }
44     }
45
46     public void draw(Graphics g) {
47         if (image == null) return;
48

```

```

49     int currentY = (int) y;
50
51     if (isStage2Img()){
52         // DRAWING STRATEGY:
53         // The screen is 800px tall. Our "Tile" is 400px.
54         // We need to cover the screen from Y=0 to Y=800.
55         // Since 'currentY' moves down, we need to draw tiles above and
below it.
56
57         // We assume the pattern is: [Normal] [Flipped] [Normal] [Flipped]
58         ...
59
60         // 1. Draw Normal Tile at current Y
61         // Covers: y to y+400 x
62         drawForceSize(g, currentY, false);
63
64         // 2. Draw Flipped Tile BELOW
65         // Covers: y+400 to y+800
66         drawForceSize(g, currentY + HEIGHT, true);
67
68         // 3. Draw Flipped Tile ABOVE
69         // Covers: y-400 to y. (Crucial for when y starts at 0 or is small
    )
70         drawForceSize(g, currentY - HEIGHT, true);
71
72         // 4. Draw Normal Tile ABOVE that
73         // Covers: y-800 to y-400. (Crucial for the loop wrap-around)
74         drawForceSize(g, currentY - (HEIGHT * 2), false);
75     } else {
76         g.drawImage(image, 0, GameConstants.HUD_HEIGHT, WIDTH, HEIGHT * 2,
77         null);
78     }
79
80 /**
81 * Helper method to draw a tile either normally or vertically flipped.
82 * @param g Graphics context
83 * @param yPos The Y position to draw at
84 * @param isFlipped If true, draws the image upside down (mirrored)
85 */

```

```

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,
11 * Boulders).
12 * It introduces the concept of "Power Level" to handle projectile-vs-
13 * projectile collisions.
14 */
15 public abstract class Projectile extends GameObject {
16
17     protected Alignment alignment;
18
19     // Power Level determines priority when two projectiles collide:
20     // 0 = Ephemeral (Destroyed by anything)
21     // 1 = Light (Arrow, Feather)
```

```

20 // 2 = Heavy (Boulder - Destroys Light)
21 // 3 = Ultimate (Sun - Destroys Heavy and Light)
22 protected int powerLevel;
23
24 protected int damage;
25
26 // If true, the projectile does not vanish after hitting a target (e.g.,
27 // The Sun)
27 protected boolean isPenetrating;
28
29 public Projectile(int x, int y, int w, int h, BufferedImage image,
30 Alignment alignment, int powerLevel, int damage) {
31     super(x, y, w, h, image);
32     this.alignment = alignment;
33     this.powerLevel = powerLevel;
34     this.damage = damage;
35     this.isPenetrating = false; // Default: destroys itself on impact
36 }
37
38 // Getters
39 public Alignment getAlignment() { return alignment; }
40 public int getPowerLevel() { return powerLevel; }
41 public int getDamage() { return damage; }
42 public boolean isPenetrating() { return isPenetrating; }
43 }
```

Listing 18: Player.java

```

1 package model;
2
3 import view.ResourceManager; // Import the manager
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6 import java.awt.geom.Ellipse2D;
7
8 // --- クラス Player 自分 () ---
9 public class Player extends GameObject {
10     private int velX = 0; // 横の移動速度
11     private int velY = 0; // 縦の移動速度
12     private int speed = GameConstants.PLAYER_SPEED;
```

```

13     private int level = 1;
14
15     public Player(int x, int y) {
16         super(x, y, GameConstants.PLAYER_WIDTH, GameConstants.PLAYER_HEIGHT,
17               ResourceManager.playerImg); // 40の四角
18         x40
19     }
20
21     @Override
22     public void move() {
23         x += velX;
24         y += velY;
25
26         // 画面からはみ出さないように制限
27         if (x < 0) x = 0;
28         if (x > GameConstants.WINDOW_WIDTH - width) x = GameConstants.
29             WINDOW_WIDTH - width;
30         if (y < GameConstants.HUD_HEIGHT) y = GameConstants.HUD_HEIGHT;
31         if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height)
32             y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height;
33     }
34
35     @Override
36     public void draw(Graphics g) {
37         if(image != null){
38             g.drawImage(image, x, y, width, height, null);
39         }
40         else{
41             // Fallback if image failed to load
42             g.setColor(Color.BLUE);
43             g.fillRect(x, y, width, height);
44         }
45     }
46
47     @Override
48     public Shape getShape() {
49         // we define the margin
50         float paddingX = width * 0.3f; // remove 20 % width
51         float paddingY = height * 0.2f; // remove 10 % height

```

```

49     // the smaller hitbox get centered compared to the original immage
50     return new Ellipse2D.Float(
51         x + paddingX / 2,           // move right
52         y + paddingY / 2,           // move down
53         width - paddingX,          // reduce width
54         height - paddingY        // reduce height
55     );
56 }
57
58 // から呼ばれるメソッド Controller
59 public void setVelX(int vx) {
60     this.velX = vx * speed;
61 }
62
63 public void setVelY(int vy) {
64     this.velY = vy * speed;
65 }
66
67 public int getLevel(){
68     return level;
69 }
70 }
```

Listing 19: Arrow.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 public class Arrow extends Projectile {
8
9     public Arrow(int x, int y, int arrowDamage) {
10         // Alignment: PLAYER
11         // Power Level: 1 (Light) -> Can be destroyed by Boulders (Lvl 2) or
12         // Sun (Lvl 3)
13         // Damage: Standard Arrow Damage
14         super(x, y, GameConstants.ARROW_WIDTH, GameConstants.ARROW_HEIGHT,
15               ResourceManager.arrowImg,
```

```

14             Alignment.PLAYER, 1, arrowDamage);
15         }
16
17     @Override
18     public void move() {
19         y -= GameConstants.ARROW_SPEED;
20         if (y < -height) {
21             isDead = true;
22         }
23     }
24
25     @Override
26     public void draw(Graphics g) {
27         if (ResourceManager.arrowImg != null) {
28             g.drawImage(image, x, y, width, height, null);
29         } else {
30             g.setColor(Color.YELLOW);
31             g.fillRect(x, y, width, height);
32         }
33     }
34 }
```

Listing 20: BossProjectile.java

```

1 package model;
2
3 import java.awt.image.BufferedImage;
4
5 public abstract class BossProjectile extends Projectile {
6
7     protected boolean isPlayerProjectile;
8     protected int maxHP;
9     protected int currentHP;
10
11     public BossProjectile(int x, int y, int w, int h, BufferedImage image,
12                           Alignment alignment, int powerLevel, int damage){
13         super(x, y, w, h, image, alignment, powerLevel, damage);
14     }
15
16     public void reduceHealth(int damage) {
```

```

16     // Only Player's Projectiles can die
17     if (!isPlayerProjectile) return;
18     this.currentHP -= damage;
19     if (this.currentHP <= 0) {
20         this.isDead = true;
21     }
22 }
23 }
```

Listing 21: Boss.java

```

1 package model;
2
3 import java.awt.*;
4 import java.awt.image.BufferedImage;
5
6 public abstract class Boss extends HostileEntity {
7
8     public Boss(int x, int y, int w, int h, BufferedImage image, int hp, int
9     scorePoints, GameModel model) {
10        super(x, y, w, h, image, hp, scorePoints, model);
11    }
12
13    @Override
14    public void takeDamage(int dmg) {
15        super.takeDamage(dmg);
16
17        if (this.isDead){
18            model.bossDefeated();
19        }
20    }
}
```

Listing 22: Apollo.java

```

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

```

6
7 /**
8 * Apollo Class
9 * Represents the first Boss of the game.
10 * It moves horizontally, bounces off walls, and shoots "Sun" projectiles.
11 * It has two phases: Normal and Enraged (Red & Fast).
12 */
13 public class Apollo extends Boss {
14
15     private int speedX = GameConstants.APOLLO_SPEED1;
16     private boolean secondPhase = false; // Flag to track if the boss is in "
17     Rage Mode"
18
19     public Apollo(GameModel model) {
20         // Call the parent constructor (Boss -> HostileEntity)
21         // Parameters: x, y, width, height, HP, Score Points
22         super(
23             (GameConstants.WINDOW_WIDTH - GameConstants.APOLLO_WIDTH) / 2,
24             // Start in the middle
25             GameConstants.HUD_HEIGHT, // Start below HUD
26             GameConstants.APOLLO_WIDTH,
27             GameConstants.APOLLO_HEIGHT,
28             ResourceManager.apolloImg,
29             GameConstants.APOLLO_HP,
30             GameConstants.APOLLO_SCORE_POINTS, // Score awarded when
31             defeated
32             model
33         );
34         this.image = ResourceManager.apolloImg;
35     }
36
37     @Override
38     public void move() {
39         super.move();
40         // Update horizontal position
41         x += speedX;
42
43         // Bounce logic: If it hits the screen edges
44         if (x <= 0 || x >= GameConstants.WINDOW_WIDTH - width) {
45             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
46         // Red/Fast
47         shootSun();
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
58         // death, adds score)
59         super.takeDamage(dmg);
60
61         // 2. Specific Logic for Apollo: Check for Second Phase (Rage Mode)
62         // If HP drops below 50% and we are not yet in the second phase...
63         if (hp <= maxHp / 2 && !secondPhase) {
64             image = ResourceManager.apolloImg2; // Change sprite to Red Apollo
65             speedX = (speedX > 0) ? GameConstants.APOLLO_SPEED2 : -
66             GameConstants.APOLLO_SPEED2; // Double the movement speed
67             secondPhase = true; // Activate the flag
68             System.out.println("Apollo entering Phase 2!");
69         }
70     }
71
72     @Override
73     public void draw(Graphics g) {
74         BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
75         apolloHitImg : image;
76
77         if (image != null) {
78             // Directional Flipping Logic
79             if (speedX > 0) {
80                 // Moving RIGHT: Draw normally
81                 g.drawImage(imgToDraw, x, y, width, height, null);

```

```

79     } else {
80         // Moving LEFT: Flip the image horizontally
81         // We draw starting at (x + width) and use a negative width to
82         flip
83             g.drawImage(imgToDraw, x + width, y, -width, height, null);
84         }
85     } else {
86         // Fallback: Draw an Orange rectangle if images fail to load
87         g.setColor(Color.ORANGE);
88         g.fillRect(x, y, width, height);
89     }
90 }
```

Listing 23: Sun.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.geom.Ellipse2D;
6 import java.awt.image.BufferedImage;
7
8 public class Sun extends BossProjectile {
9
10    private double preciseX, preciseY;
11    private double velX, velY;
12
13    private double initialSize;
14
15    public Sun(int summonerX, int summonerY, int ApolloSpeedX, boolean
16    isSecondPhase, boolean friendly) {
17        // Call parent constructor
18        // HP = 1 (doesn't matter), Score = 0
19        super(0,
20              0,
21              GameConstants.SUN_WIDTH,
22              GameConstants.SUN_HEIGHT,
23              isSecondPhase ? ResourceManager.sunImg2 : ResourceManager.
sunImg,
```

```

23         friendly ? Alignment.PLAYER : Alignment.ENEMY,
24         3,
25         GameConstants.SUN_DAMAGE);
26
27     this.isPlayerProjectile = friendly;
28     this.isPenetrating = true;
29
30     if (isPlayerProjectile) {
31         maxHP = GameConstants.SUN_HP;
32         currentHP = maxHP;
33     }
34
35     this.initialSize = GameConstants.SUN_WIDTH;
36
37     velX = 1;
38     velY = 1;
39     double currentSpeed;
40     double angleRadians;
41
42     if (isSecondPhase) {
43         currentSpeed = GameConstants.SUN_SPEED2;
44     } else {
45         currentSpeed = GameConstants.SUN_SPEED1;
46     }
47
48     if (!friendly){
49         // sun comes from apollo
50         preciseX = (ApolloSpeedX > 0) ? (summonerX + GameConstants.
APOLLO_WIDTH + width / 2) : (summonerX - width / 2);
51         preciseY = summonerY + height;
52         x = (int) preciseX;
53         y = (int) preciseY;
54         // --- Phase Logic ---
55         // --- Trajectory Calculation ---
56         double minAngle = 20.0;
57         double maxAngle = 70;
58         angleRadians = Math.toRadians(minAngle + Math.random() * (maxAngle
- minAngle));
59     } else {
60         // sun comes from ability

```

```

61     preciseX = summonerX;
62     preciseY = summonerY - height / 2;
63
64     // 2. right boarder check
65     if (preciseX > GameConstants.WINDOW_WIDTH - width / 2) {
66         preciseX = GameConstants.WINDOW_WIDTH - width / 2;
67     }
68
69     // 3. left boarder check
70     if (preciseX < width / 2) {
71         preciseX = width / 2;
72     }
73
74     x = (int) preciseX;
75     y = (int) preciseY;
76     angleRadians = Math.toRadians(360-30);
77     velX = (Math.random() < 0.5) ? 1 : -1;
78 }
79
80     velX *= currentSpeed * Math.cos(angleRadians);
81     velY *= currentSpeed * Math.sin(angleRadians);
82
83     if (ApolloSpeedX < 0) {
84         velX = -velX;
85     }
86 }
87
88 @Override
89 public void move() {
90     preciseX += velX;
91     preciseY += velY;
92
93     x = (int) preciseX;
94     y = (int) preciseY;
95
96     double currentRadius = getCurrentSize() / 2.0;
97
98     // Wall Bounce
99     if (x - currentRadius < 0) {
100        x = (int)currentRadius;

```

```

101         preciseX = x;
102         velX = -velX;
103     }
104
105     if (x + currentRadius > GameConstants.WINDOW_WIDTH) {
106         x = (int)(GameConstants.WINDOW_WIDTH - currentRadius);
107         preciseX = x;
108         velX = -velX;
109     }
110
111     // Despawn
112     if (y - currentRadius > GameConstants.FIELD_HEIGHT + GameConstants.
113 HUD_HEIGHT ||
114         y + currentRadius < GameConstants.HUD_HEIGHT) {
115         isDead = true;
116     }
117
118     // Helper to calculate size based on HP
119     private double getCurrentSize() {
120         if (!isPlayerProjectile) return initialSize;
121
122         // Calculate ratio: currentHP / maxHP
123         double ratio = (double) currentHP / maxHP;
124         return initialSize * ratio;
125     }
126
127     @Override
128     public void draw(Graphics g) {
129         double size = getCurrentSize();
130         double radius = size / 2.0;
131
132         int drawX = (int) (x - radius);
133         int drawY = (int) (y - radius);
134         int drawSize = (int) size;
135
136         if (image != null) {
137             g.drawImage(image, drawX, drawY, drawSize, drawSize, null);
138         } else {
139             g.setColor(Color.YELLOW);
140             g.fillOval(x, y, width, height);

```

```

140     }
141 }
142
143 @Override
144 public Shape getShape() {
145     double size = getCurrentSize();
146     double radius = size / 2.0;
147     return new Ellipse2D.Float((float)(x - radius), (float)(y - radius), (
148         float)size, (float)size);
149 }
150 }
```

Listing 24: Zeus.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6 import java.util.Random;
7
8 public class Zeus extends Boss {
9     private int speedX = GameConstants.ZEUS_SPEED;
10    private boolean secondPhase = false; // Flag to track if the boss is in "Rage Mode"
11    private int shootTimer;
12    private int maxShootTimer;
13    private int ability1Timer;
14    private int ability2Timer;
15    private int ability1Pause;
16    private int maxAbility1Pause;
17    private int ability1Position;
18    private BufferedImage hitImg;
19    boolean ability1Phase;
20    boolean ability2Started;
21    private int ability2Repetitions;
22
23    public Zeus (GameModel model) {
24        super ( (GameConstants.WINDOW_HEIGHT - GameConstants.ZEUS_WIDTH) / 2,
```

```

25         GameConstants.HUD_HEIGHT,
26         GameConstants.ZEUS_WIDTH,
27         GameConstants.ZEUS_HEIGHT,
28         ResourceManager.zeusImg,
29         GameConstants.ZEUS_HP,
30         GameConstants.ZEUS_SCORE_POINTS,
31         model
32     );
33     maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER;
34     resetShootTimer();
35     maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE;
36     setAbility1Timer();
37     ability1Position = 1;
38     hitImg = ResourceManager.zeusHitImg;
39     ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
40     ability2Started = false;
41 }
42
43 @Override
44 public void move() {
45     super.move();
46     if (ability2Timer > 0 || ability1Phase){
47         if (ability1Timer > 0) {
48             // Update horizontal position
49             x += speedX;
50
51             // Bounce logic: If it hits the screen edges
52             if (x <= 0 ) {
53                 x = 0;
54                 speedX = -speedX; // Reverse direction
55                 ability1Timer--;
56             } else if (x >= GameConstants.WINDOW_WIDTH - width) {
57                 x = GameConstants.WINDOW_WIDTH - width;
58                 speedX = -speedX; // Reverse direction
59                 ability1Timer--;
60             }
61             if(shootTimer <= 0){
62                 shootLighting();
63                 resetShootTimer();
64             }
}

```

```

65         shootTimer--;
66     } else if (ability1Pause <= 0){
67         ability1();
68     } else {
69         ability1Pause--;
70     }
71 } else {
72     if (!ability2Started) {
73         if (Math.random() < 0.5) {
74             x = 0;
75             speedX = GameConstants.ZEUS_SPEED2;
76         } else {
77             x = GameConstants.WINDOW_WIDTH - width;
78             speedX = -GameConstants.ZEUS_SPEED2;
79         }
80         Random random = new Random();
81         ability2Repetitions = random.nextInt(3) + 1;
82         ability2Started = true;
83     }
84     x += speedX;
85
86     if (((x < (GameConstants.PLAYER_WIDTH - 20)) && (speedX < 0)) || (
87         x > (GameConstants.WINDOW_WIDTH - (width + GameConstants.PLAYER_WIDTH)) &&
88         speedX > 0)){
89         } else {
90             shootLighting();
91         }
92
93         if (x <= 0 ) {
94             x = 0;
95             speedX = -speedX; // Reverse direction
96             ability2Repetitions--;
97         } else if (x >= GameConstants.WINDOW_WIDTH - width) {
98             x = GameConstants.WINDOW_WIDTH - width;
99             speedX = -speedX; // Reverse direction
100            ability2Repetitions--;
101        }
102        if (ability2Repetitions == 0){
103            ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
104            ability2Started = false;

```

```

103         resetShootTimer();
104     }
105
106 }
107
108 if (secondPhase && ability2Timer > 0){
109     ability2Timer--;
110 }
111
112
113 private void ability1() {
114     if (ability1Position == 1){
115         ability1Phase = true;
116     }
117
118     if (ability1Position == 5) {
119         setAbility1Timer();
120         ability1Position = 1;
121         resetShootTimer();
122         return;
123     }
124
125     if(speedX > 0) {
126         x = ability1Position * (GameConstants.WINDOW_WIDTH-width)/4;
127     } else {
128         x = (4 - ability1Position) * (GameConstants.WINDOW_WIDTH-width)/4;
129     }
130     if (ability1Position % 2 == 0){
131         y = GameConstants.HUD_HEIGHT;
132     } else {
133         y = GameConstants.HUD_HEIGHT + height / 3;
134     }
135
136     shootLighting();
137     ability1Pause = maxAbility1Pause;
138     ability1Position++;
139 }
140
141 private void shootLighting(){
142     Lighting l = new Lighting(x, y, speedX, secondPhase, false,

```

```

    ability2Started);
    model.spawnEnemyProjectile(1);
}

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

    if (hp <= maxHp / 2 && !secondPhase) {
        image = ResourceManager.zeusImg2;
        hitImg = ResourceManager.zeusHitImg2;
        speedX = (speedX > 0) ? GameConstants.ZEUS_SPEED2 : -GameConstants
.ZEUS_SPEED2; // Double the movement speed
        secondPhase = true;
        maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER2;
        maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE2;
    }
}

private void setAbility1Timer(){
    Random random = new Random();
    ability1Timer = random.nextInt(4) + 1;;
    ability1Phase = false;
}

private void resetShootTimer(){
    shootTimer = maxShootTimer;
}

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

    if (image != null) {
        if (speedX > 0) {
            g.drawImage(imgToDraw, x, y, width, height, null);
        } else {
            g.drawImage(imgToDraw, x + width, y, -width, height, null);
        }
    } else {

```

```

181         g.setColor(Color.ORANGE);
182         g.fillRect(x, y, width, height);
183     }
184 }
185 }
```

Listing 25: Lighting.java

```

1 package model;

2

3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.geom.Ellipse2D;

6

7 public class Lighting extends BossProjectile {

8

9     private int velY;
10    public Lighting(int x, int y, int ZeusSpeedX, boolean isSecondPhase,
11                   boolean friendly, boolean ability2Started) {
12        super(0,
13              0,
14              GameConstants.LIGHTING_WIDTH,
15              GameConstants.LIGHTING_HEIGHT,
16              isSecondPhase ? ResourceManager.lightingImg2 : ResourceManager
17              .lightingImg,
18              friendly ? Alignment.PLAYER : Alignment.ENEMY,
19              3,
20              GameConstants.LIGHTING_DAMAGE);

21

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

24

25    if (isSecondPhase) {
26        velY = GameConstants.LIGHTING_SPEED2;
27    } else {
28        velY = GameConstants.LIGHTING_SPEED1;
29    }

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

```

31         maxHP = GameConstants.LIGHTING_HP;
32         currentHP = maxHP;
33         // set position for player
34         this.x = x + (GameConstants.PLAYER_WIDTH - width) / 2;
35         this.y = y - GameConstants.PLAYER_HEIGHT;
36     } else {
37         if (!ability2Started){
38
39             // set position for Zeus
40             this.x = (ZeusSpeedX > 0) ? x + GameConstants.ZEUS_WIDTH - width :
41             x + width;
42             } else {
43                 this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
44             width;
45             }
46         this.y = y + height;
47     }
48
49     @Override
50     public void move() {
51         y += vely;
52
53         if (y > GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT || y <
54             GameConstants.HUD_HEIGHT - height) {
55             isDead = true;
56         }
57     }
58
59     @Override
60     public void draw(Graphics g) {
61         if (image != null) {
62             g.drawImage(image, x, y, width, height, null);
63         } else {
64             g.setColor(Color.YELLOW);
65             g.fillOval(x, y, width, height);
66         }
67     }
68
69     @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
9  * Golems).
10 */
11
12 private Class<? extends Minion> enemyType; // The class of the enemy to
spaw
```

```

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,
19                           int variance) {
20         this.enemyType = enemyType;
21         this.baseInterval = baseInterval;
22         this.variance = variance;
23         resetTimer(); // Start the timer immediately
24     }
25
26     /**
27      * Updates the timer.
28      * @return true if it's time to spawn an enemy, false otherwise.
29      */
30
31     public boolean update() {
32         timer--;
33         if (timer <= 0) {
34             resetTimer();
35             return true;
36         }
37         return false;
38     }
39
40     private void resetTimer() {
41         // Calculate next spawn time: base +/- random variance
42         int var = (variance > 0) ? rand.nextInt(variance * 2 + 1) - variance :
43             0;
44         this.timer = baseInterval + var;
45         if (this.timer < 30) this.timer = 30; // Minimum safety limit (0.5 sec)
46     }
47
48     /**
49      * Increases difficulty by reducing the spawn interval.
50      * @param multiplier Factor to multiply interval (e.g., 0.8 for 20% faster
51      *).
52      */

```

```

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
12 * by the Sun.
13 */
14 public class Boulder extends Projectile {
15
16     private double preciseY;
17     private double velY;
18     private double gravity = GameConstants.Boulder_GRAVITY; // Accelerates
19     downwards
20
21     public Boulder(int x, int y) {
22         super(x, y,
23               GameConstants.Boulder_WIDTH,
24               GameConstants.Boulder_HEIGHT,
25               ResourceManager.boulderImg,
26               Alignment.ENEMY,
```

```

25         2,
26         GameConstants.Boulder_DAMAGE
27     );
28
29     this.preciseY = y;
30     this.velY = GameConstants.Boulder_INITIAL_SPEED; // Initial throw
31     speed
32 }
33
34 @Override
35 public void move() {
36     // Apply Gravity
37     velY += gravity;
38     preciseY += velY;
39     y = (int) preciseY;
40
41     // Despawn if off-screen
42     if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT) {
43         isDead = true;
44     }
45 }
46
47 @Override
48 public void draw(Graphics g) {
49     if (image != null) {
50         g.drawImage(image, x, y, width, height, null);
51     } else {
52         // Placeholder: Gray Rock
53         g.setColor(Color.GRAY);
54         g.fillOval(x, y, width, height);
55     }
56 }
57
58 @Override
59 public Shape getShape() {
60     // set hitbox to a circle
61     return new Ellipse2D.Float(x, y, width, height);
62 }

```

Listing 29: Cyclops.java

```
1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 public class Cyclops extends Minion{
8
9     private double preciseY;
10    private double velY;
11    private int movementTimer;
12    private int attackTimer;
13    private Boulder myBoulder = null;
14    private boolean reachedMidScreen;
15    private boolean wingsClosed = false;
16
17    public Cyclops (int x, int y, GameModel model){
18        super(x, y,
19              GameConstants.CYCLOPS_WIDTH,
20              GameConstants.CYCLOPS_HEIGHT,
21              ResourceManager.cyclopsImg,
22              GameConstants.CYCLOPS_HP,
23              GameConstants.CYCLOPS_SCORE_POINTS,
24              model);
25        this.preciseY = y;
26        this.velY = GameConstants.CYCLOPS_YSPEED;
27        this.movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
28        this.attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
29        this.reachedMidScreen = false;
30    }
31
32    @Override
33    public void move() {
34        super.move();
35        if (movementTimer <= 0){
36            if (velY > 0){
37                // Go up
38                velY = GameConstants.CYCLOPS_YSPEED / -4;
39                movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 / 2;
```

```

40         wingsClosed = true; // STATE: closed wings
41     }
42     else {
43         // go down
44         wingsClosed = false; // STATE: opened wings
45
46         if (!reachedMidScreen) {
47             vely = GameConstants.CYCLOPS_YSPEED;
48             movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
49         } else {
50             vely = GameConstants.CYCLOPS_YSPEED / 4;
51             movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 /
52                 2;
53         }
54         this.image = wingsClosed ? ResourceManager.cyclopsImg2 :
55         ResourceManager.cyclopsImg;
56
57         preciseY += vely;
58         y = (int) preciseY;
59
60         if(y > (GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT/2 -
61             height)){
62             reachedMidScreen = true;
63         }
64
65         // CHECK IF THERE IS BOULDER OR IF IT IS DEAD
66         if (myBoulder == null || myBoulder.isDead()) {
67             // REDUCE TIMER
68             if(attackTimer > 0){
69                 attackTimer--;
70             }
71             // ONCE THE TIMER IS 0 SPAWN BOULDER AND RESET TIMER
72             if (attackTimer == 0){
73                 throwBoulder();
74                 attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
75             }
76         }
77         movementTimer--;
78     }

```

```

77
78     @Override
79     public void draw(Graphics g) {
80         BufferedImage imgToDraw;
81
82         if (image == ResourceManager.cyclopsImg){
83             imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg :
84             image;
85         } else {
86             imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg2 :
87             image;
88         }
89
90         if (image != null) {
91             g.drawImage(imgToDraw, x, y, width, height, null);
92         } else {
93             g.setColor(Color.RED);
94             g.fillRect(x, y, width, height);
95         }
96
97     private void throwBoulder(){
98         Boulder b = new Boulder (x + (width - GameConstants.Boulder_WIDTH)/2,
99         y + height);
100        this.myBoulder = b;
101        model.spawnEnemyProjectile(b);
102    }
103 }
```

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
```

```

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

Listing 31: Harpy.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 public class Harpy extends Minion {
8
9     private int velX;
10    private int velY;
11    private int fireTimer;
```

```

12     private boolean isInScreen;
13
14     public Harpy(int x, int y, GameModel model) {
15         // Pass params to parent: x, y, width, height, HP, Score Points
16         super(x, y,
17             GameConstants.HARPY_WIDTH,
18             GameConstants.HARPY_HEIGHT,
19             ResourceManager.harpyImg,
20             GameConstants.HARPY_HP,
21             GameConstants.HARPY_SCORE_POINTS,
22             model);
23
24         // --- MOVEMENT SETUP ---
25         this.velY = GameConstants.HARPY_YSPEED;
26         this.velX = GameConstants.HARPY_XSPEED;
27         this.isInScreen = false;
28
29         // Randomize direction
30         if (Math.random() < 0.5) {
31             this.velX = -this.velX;
32         }
33         resetFireTimer();
34     }
35
36     @Override
37     public void move() {
38         super.move();
39         x += velX;
40         y += velY;
41
42         // Bounce Logic
43         if (x < 0) {
44             x = 0;
45             velX = -velX;
46         }
47         if (x > GameConstants.WINDOW_WIDTH - width) {
48             x = GameConstants.WINDOW_WIDTH - width;
49             velX = -velX;
50         }
51         if (y < GameConstants.HUD_HEIGHT && isInScreen) {

```

```

52         y = GameConstants.HUD_HEIGHT;
53         vely = -vely;
54     }
55     if (y > GameConstants.HUD_HEIGHT) {
56         isInScreen = true;
57     }
58     if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
59 ) {
60         y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
61 ;
62         vely = -vely;
63     }
64
65     if(fireTimer > 0) {
66         fireTimer--;
67     }
68     if (fireTimer <= 0) {
69         throwFeather();
70         resetFireTimer();
71     }
72 }
73
74 private void throwFeather() {
75     Feather f = new Feather (x + (width - GameConstants.FEATHER_WIDTH)/2,
76     y + height);
77     model.spawnEnemyProjectile(f);
78 }
79
80
81 public void resetFireTimer() {
82     int base = GameConstants.FEATHER_FIRE_INTERVAL;
83     int variance = GameConstants.FEATHER_FIRE_VARIANCE;
84
85     int randomVariation = (int)(Math.random() * (variance * 2)) - variance
86 ;
87
88     this.fireTimer = base + randomVariation;
89 }
90
91
92 @Override
93 public void draw(Graphics g) {

```

```

88     BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
89     harpyHitImg : image;
90
91     if (image != null) {
92         if (velX < 0) {
93             g.drawImage(imgToDraw, x, y, width, height, null);
94         } else {
95             g.drawImage(imgToDraw, x + width, y, -width, height, null);
96         }
97     } else {
98         g.setColor(Color.RED);
99         g.fillRect(x, y, width, height);
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
12         scorePoints, GameModel model) {
13         super(x, y, w, h, image, hp, scorePoints, model);
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,
29                                         GameConstants.WINDOW_HEIGHT));
30         this.setBackground(Color.BLACK);
31         this.setFocusable(true);
32         this.addKeyListener(this);
33
34         // Game Loop Timer
35         timer = new Timer(1000/GameConstants.FPS, e -> {
36             model.update();
37             repaint();
38         });
39         timer.start();
40     }
41
42     @Override
43     protected void paintComponent(Graphics g) {
44         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);

```

```

83         // Fill only the game field area
84         g.fillRect(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH
85 , GameConstants.FIELD_HEIGHT);
86     }
87
88     // 2. Draw Game Objects (Player, Enemies, Projectiles)
89     for (GameObject obj : model.getObjects()) {
90         // Invincibility flashing logic for Player
91         if (obj instanceof Player && model.isInvincible()) {
92             // Toggle visibility every 100ms
93             if (System.currentTimeMillis() % 200 < 100) {
94                 continue;
95             }
96             obj.draw(g);
97         }
98
99         // 3. Draw Top HUD (Score, Stage, Lives)
100        drawTopHUD(g);
101
102        // 4. Draw Bottom HUD (Ability Slots) -> NEW!
103        drawBottomHUD(g);
104    }
105
106    // Helper method to draw the Top HUD
107    private void drawTopHUD(Graphics g) {
108        // Draw dark background bar
109        g.setColor(new Color(50, 50, 80));
110        g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.HUD_HEIGHT)
111 ;
112
113        // Draw white separator line
114        g.setColor(Color.WHITE);
115        g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH,
116 GameConstants.HUD_HEIGHT);
117
118        // Font settings
119        setPixelFont(g, 18f);

```

```

120     // 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().
126     stringWidth(stageText)) / 2;
127     g.drawString(stageText, stageX, textY);
128
129     // C. Hearts / Lives (Right aligned)
130     int maxLives = GameConstants.PLAYER_MAX_LIVES;
131     int currentLives = model.getLives();
132     int heartSize = 32;
133     int spacing = 8;
134     int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (heartSize
135     + spacing));
136     int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
137
138     for (int i = 0; i < maxLives; i++) {
139         // Determine which icon to draw (Full or Empty)
140         BufferedImage icon = (i < currentLives) ? ResourceManager.
141         heartFullImg : ResourceManager.heartEmptyImg;
142
143         if (icon != null) {
144             g.drawImage(icon, startX + (i * (heartSize + spacing)), heartY
145             , heartSize, heartSize, null);
146         } else {
147             // Fallback drawing if images are missing
148             g.setColor(i < currentLives ? Color.RED : Color.GRAY);
149             g.fillOval(startX + (i * (heartSize + spacing)), heartY,
150             heartSize, heartSize);
151         }
152     }
153
154     // Helper method to draw the Bottom HUD (Ability Slots)
155     // Helper method to draw the Bottom HUD (Ability Slots)
156     private void drawBottomHUD(Graphics g) {
157         // Calculate start Y position (below the game field)
158         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
182         // unlocked
183         if (i == 0 && model.isAbilityUnclocked(1)) {
184
185             // 1. Draw the Icon (The Sun)
186             if (ResourceManager.sunImg != null) {
187                 g.drawImage(ResourceManager.sunImg, x, slotY, slotSize,
188                 slotSize, null);
189             }
190
191             // 2. Draw Cooldown Overlay (The fading effect)
192             int timer = model.getAbilityNthTimer(1);
193             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
200         // half.
201         // This creates the effect of the color "filling up from
202         // bottom".
203         int overlayHeight = (int) (slotSize * ratio);
204
205         // Set color to semi-transparent black
206         g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (
207         // transparency)
208
209         // Draw the overlay from the top of the slot downwards
210         g.fillRect(x, slotY, slotSize, overlayHeight);
211
212         // Optional: Draw the text timer on top if you want
213         g.setColor(Color.WHITE);
214         String keyNum = String.valueOf(timer/60 + 1);
215         int numWidth = g.getFontMetrics().stringWidth(keyNum);
216         g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY
217             + 37);
218     }
219 }
220
221     // Check if this is the first slot (Index 0) AND if Ability 1 is
222     // unlocked
223     if (i == 1 && model.isAbilityUnclocked(2)) {
224
225         // 1. Draw the Icon (The Sun)
226         if (ResourceManager.lightingImg != null) {
227             g.drawImage(ResourceManager.lightingImg, x, slotY,
228             slotSize, slotSize, null);
229         }
230
231         // 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 +
263 slotSize + 25);
264     }
265 }
266
267 // Draw Title Screen
268 private void drawTitleScreen(Graphics g) {
269     g.setColor(Color.WHITE);
270
271     // Title: Big Pixel Font
272     setPixelFont(g, 28f);
273     String title = "GLADIATOR GAME"; // Cambia il nome se vuoi
274     int titleWidth = g.getFontMetrics().stringWidth(title);
275     g.drawString(title, (GameConstants.WINDOW_WIDTH - titleWidth)/2, 250);
276
277     // Subtitle: Smaller
278     setPixelFont(g, 15f);
279     String msg = "Press SPACE to Start";
280     int msgWidth = g.getFontMetrics().stringWidth(msg);
281     g.drawString(msg, (GameConstants.WINDOW_WIDTH - msgWidth)/2, 350);
282 }
283
284 private void drawPauseScreen(Graphics g) {
285     // 1. Semi-transparent black overlay
286     g.setColor(new Color(0, 0, 0, 150)); // 150 = Alpha (Transparency)
287     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
288 WINDOW_HEIGHT);
289
290     // 2. "PAUSE" Text
291     g.setColor(Color.WHITE);
292     setPixelFont(g, 40f); // Large font
293     String pauseText = "PAUSE";
294     int pauseWidth = g.getFontMetrics().stringWidth(pauseText);
295     // Center text
296     g.drawString(pauseText, (GameConstants.WINDOW_WIDTH - pauseWidth) / 2,
297 GameConstants.WINDOW_HEIGHT / 2 - 100);
298
299     // 3. Instruction Text
300     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) /
301     2, GameConstants.WINDOW_HEIGHT / 2 - 50);
302 }
303
304 private void drawMessageScreen(Graphics g) {
305     // 1. Semi-transparent black background for the whole screen (dimming)
306     g.setColor(new Color(0, 0, 0, 100));
307     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
308     WINDOW_HEIGHT);
309
310     // 2. The Message Box Dimensions
311     int boxWidth = 500;
312     int boxHeight = 400;
313     int boxX = (GameConstants.WINDOW_WIDTH - boxWidth) / 2;
314     int boxY = (GameConstants.WINDOW_HEIGHT - boxHeight) / 2;
315
316     // 3. Draw the Box Background (Dark Blue)
317     g.setColor(new Color(20, 20, 80));
318     g.fillRect(boxX, boxY, boxWidth, boxHeight);
319
320     // 4. Draw the Box Border (White)
321     g.setColor(Color.WHITE);
322     Graphics2D g2 = (Graphics2D) g;
323     g2.setStroke(new BasicStroke(4)); // Thicker border
324     g2.drawRect(boxX, boxY, boxWidth, boxHeight);
325
326     // 5. Draw the Text
327     String[] lines = model.getCurrentMessageLines();
328     if (lines != null) {
329         setPixelFont(g, 20f); // Size for text
330         g.setColor(Color.WHITE);
331
332         int lineHeight = 30;
333         // Calculate starting Y to center the block of text vertically
334         int totalTextHeight = lines.length * lineHeight;
335         int startTextY = boxY + (boxHeight - totalTextHeight) / 2 + 10; //
+10 adjustment

```

```

335     for (int i = 0; i < lines.length; i++) {
336         String line = lines[i];
337         // Center align each line horizontally
338         int lineWidth = g.getFontMetrics().stringWidth(line);
339         int lineX = (GameConstants.WINDOW_WIDTH - lineWidth) / 2;
340
341         g.drawString(line, lineX, startTextY + (i * lineHeight));
342     }
343
344
345     // 6. Draw "Press Space" prompt at the bottom of the box
346     setPixelFont(g, 14f);
347     g.setColor(Color.YELLOW);
348     String prompt = "PRESS [SPACE] TO CONTINUE";
349     int promptWidth = g.getFontMetrics().stringWidth(prompt);
350     g.drawString(prompt, (GameConstants.WINDOW_WIDTH - promptWidth) / 2,
351     boxY + boxHeight - 20);
352 }
353
354 // Draw Game Over Screen
355 private void drawGameOverScreen(Graphics g) {
356     // Semi-transparent overlay
357     g.setColor(new Color(0, 0, 0, 150));
358     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
359     WINDOW_HEIGHT);
360
361     // Game Over Text
362     g.setColor(Color.RED);
363     setPixelFont(g, 35f); // Big Red Text
364     String GO = "GAME OVER";
365     int goWidth = g.getFontMetrics().stringWidth(GO);
366     g.drawString(GO, (GameConstants.WINDOW_WIDTH - goWidth)/2, 250);
367
368     g.setColor(Color.WHITE);
369     setPixelFont(g, 20f);
370
371     String scoreMsg = "Final Score: " + model.getScore();
372     int scoreWidth = g.getFontMetrics().stringWidth(scoreMsg);
373     g.drawString(scoreMsg, (GameConstants.WINDOW_WIDTH - scoreWidth)/2,
374     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().
378 stringWidth(cont))/2, 400);
379     g.drawString(quit, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
380 stringWidth(quit))/2, 440);
381 }
382
383 // Update Player Velocity based on key states
384 private void updatePlayerVelocity() {
385     Player p = model.getPlayer();
386
387     int vx = 0;
388     int vy = 0;
389
390     if (leftPressed && !rightPressed) vx = -1;
391     if (rightPressed && !leftPressed) vx = 1;
392     if (upPressed && !downPressed) vy = -1;
393     if (downPressed && !upPressed) vy = 1;
394
395     p.setVelX(vx);
396     p.setVelY(vy);
397 }
398
399 // Reset keys when restarting game
400 private void resetKeyState() {
401     leftPressed = false;
402     rightPressed = false;
403     upPressed = false;
404     downPressed = false;
405
406     Player p = model.getPlayer();
407     if (p != null) {
408         p.setVelX(0);
409         p.setVelY(0);
410     }
411 }
```

```

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 // BOSSSES
37 //*****
38
39 // APOLLO
40 public static BufferedImage apolloImg;
41 public static BufferedImage apolloImg2;
42 public static BufferedImage apolloHitImg;
43 // APOLLO's SUN
44 public static BufferedImage sunImg;
45 public static BufferedImage sunImg2;
46 // ZEUS
47 public static BufferedImage zeusImg;
48 public static BufferedImage zeusImg2;
49 public static BufferedImage zeusHitImg;
50 public static BufferedImage zeusHitImg2;
51 // ZEUS's LIGHTNING
52 public static BufferedImage lightingImg;
53 public static BufferedImage lightingImg2;
54
55 //*****
56 // HUD
57 //*****
58
59 // STAGES
60 public static BufferedImage stage1Img;
61 public static BufferedImage stage2Img;
```

```

62     public static BufferedImage stage3Img;
63     // HEART
64     public static BufferedImage heartFullImg;
65     public static BufferedImage heartEmptyImg;
66
67     // PIXEL FONT
68     public static Font pixelFont;
69
70     /**
71      * "res" フォルダからすべてのリソースを読み込む"
72      */
73     public static void loadImages() {
74         try {
75             System.out.println("Loading resources...");
76
77             // PLAYER
78             playerImg = loadTexture("res/player.png");
79             arrowImg   = loadTexture("res/arrow.png");
80
81             //*****
82             // MINIONS
83             //*****
84             // HARPY
85             harpyImg = loadTexture("res/enemy.png");
86             harpyHitImg = createWhiteSilhouette(harpyImg);
87             // HARPY's FEATHER
88             featherImg = loadTexture("res/feather.png");
89             // CYCLOPS
90             cyclopsImg = loadTexture("res/cyclops_openedwings.png");
91             cyclopsImg2 = loadTexture("res/cyclops_closedwings.png");
92             cyclopsHitImg = createWhiteSilhouette(cyclopsImg);
93             cyclopsHitImg2 = createWhiteSilhouette(cyclopsImg2);
94             // CYCLOPS's BOULDER
95             boulderImg = loadTexture("res/boulder.png");
96
97             //*****
98             // BOSSES
99             //*****
100            // APOLLO

```

```

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().
133             getResourceAsStream("res/PixelFont.ttf");
134
135         if (is != null) {
136             // Create the font object (default size is 1pt)
137             pixelFont = Font.createFont(Font.TRUETYPE_FONT, is);
138             System.out.println("Pixel Font loaded successfully!");
139         } else {
140             System.err.println("Error: PixelFont.ttf not found. Using
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(
158         path);
159
160     if (url == null) {
161         // もし getResource で見つからない場合（フォルダ構成の違いなど）、
162         // 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）
163         try {
164             return ImageIO.read(new File(path));
165         } catch (IOException ex) {
166             throw new IOException("Image not found: " + path);
167         }
168     }
169     return ImageIO.read(url);
170 }

171
172 // 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
173 private static BufferedImage createWhiteSilhouette(BufferedImage original)
174 {
175     // 元の画像と同じサイズで、空の画像を作成
176     BufferedImage whiteImg = new BufferedImage(
177         original.getWidth(),
178         original.getHeight(),
179         BufferedImage.TYPE_INT_ARGB

```

```

178 );
179
180 // すべてのピクセルを走査する
181 for (int x = 0; x < original.getWidth(); x++) {
182     for (int y = 0; y < original.getHeight(); y++) {
183         int p = original.getRGB(x, y);
184
185         // アルファ値（透明度）を取得
186         int a = (p >> 24) & 0xff;
187
188         // 透明ではない部分（キャラクター部分）だけを「真っ白」に塗りつぶす
189         if (a > 0) {
190             // ARGB: アルファ値 + R(255) + G(255) + B(255)
191             int whiteColor = (a << 24) | (255 << 16) | (255 << 8) |
192             255;
193             whiteImg.setRGB(x, y, whiteColor);
194         }
195     }
196     return whiteImg;
197 }
198
199 private static BufferedImage createBlackSilhouette(BufferedImage original)
200 {
201     BufferedImage blackImg = new BufferedImage(original.getWidth(),
202         original.getHeight(), BufferedImage.TYPE_INT_ARGB);
203
204     for (int x = 0; x < original.getWidth(); x++) {
205         for (int y = 0; y < original.getHeight(); y++) {
206             int p = original.getRGB(x, y);
207             int a = (p >> 24) & 0xff; // Get Alpha
208
209             // If the pixel is not transparent, make it BLACK
210             if (a > 0) {
211                 // ARGB: Alpha + R(0) + G(0) + B(0)
212                 int blackColor = (a << 24) | (0 << 16) | (0 << 8) | 0;
213                 blackImg.setRGB(x, y, blackColor);
214             }
215         }
216     }
217     return blackImg;

```

215 }

216 }

(文責：佐々木)