

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

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 1: 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);
```

```

9         g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH
10        , GameConstants.HUD_HEIGHT);
11
12        // Font settings
13        setPixelFont(g, 18f);
14        int textY = 35;
15
16        // A. Score
17        g.drawString("SCORE:" + model.getScore(), 10, textY);
18
19        // B. Stage (Centered)
20        String stageText = model.getStageText();
21        int stageX = (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
22        stringWidth(stageText)) / 2;
23        g.drawString(stageText, stageX, textY);
24
25        // C. Hearts / Lives (Right aligned)
26        int maxLives = GameConstants.PLAYER_MAX_LIVES;
27        int currentLives = model.getLives();
28        int heartSize = 32;
29        int spacing = 8;
30        int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (
31        heartSize + spacing));
32        int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
33
34        for (int i = 0; i < maxLives; i++) {
35            // Determine which icon to draw (Full or Empty)
36            BufferedImage icon = (i < currentLives) ? ResourceManager.
37            heartFullImg : ResourceManager.heartEmptyImg;
38
39            if (icon != null) {
40                g.drawImage(icon, startX + (i * (heartSize + spacing)),
41                heartY, heartSize, heartSize, null);
42            } else {
43                // Fallback drawing if images are missing
44                g.setColor(i < currentLives ? Color.RED : Color.GRAY);
45                g.fillOval(startX + (i * (heartSize + spacing)), heartY,
46                heartSize, heartSize);
47            }
48        }

```

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になってしまい「多段ヒット」の問題が発生していました。これを解消するために、ダメージを受けた直後に一定時間の「無敵時間(クールタイム)」を導入しました。ダメージ発生時にdamageTimerをセットし、このタイマーが作動している間は新たなダメージを受け付けないように処理を分岐させています。さらに、無敵時間中はプレイヤーキャラクターを点滅させることで、視覚的にもダメージを受けたことと無敵状態であることをユーザーが理解しやすいようにしました。

3.3 Puller 担当

4 実行例

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

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

(文責：佐々木)

6.3 Puller

7 付録 1：操作法マニュアル

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



図 2: タイトル画面



図 3: スタート画面

次に、プレイ中のキー操作について説明します。

表1: プレイ中のキー操作

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

ゲームオーバー時には、C を押すとゲームをもう一度プレイ (Continue) でき、Q を押すとゲームをやめる (Quit) ことができます。



図4: ゲームオーバー画面

(文責：佐々木)

8 付録2：プログラムリスト

Listing 2: 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 3: 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;
69         arrowDamage = GameConstants.ARROW_DAMAGE;
70         arrowInterval = GameConstants.ARROW_INTERVAL;
71         state = GameState.PLAYING;
72         score = 0; //スコアをリセット
73
74         // 追加ライフ初期化:
75         lives = GameConstants.PLAYER_MAX_LIVES;
76         damageTimer = 0;
77
78         //Score progression resets
79         this.currentLevelIndex = 0;
80         this.nextTargetScore = GameConstants.LEVEL_MILESTONES[
81             currentLevelIndex];
82         isBossActive = false;
83
84
85         // STAGE 1 SETUP: Add Harpy Spawner
86         activeSpawners.add(new EnemySpawner(Harpy.class, GameConstants.
87             HARPY_SPAWN_INTERVAL, GameConstants.HARPY_SPAWN_VARIANCE));
88         this.currentStage = 1;
89
90
91         String tutorial = "WELCOME GLADIATOR!\n\n" +
92             "Controls:\n" +
93             "[KEY-ARROWS] Move\n" +
94             "[SPACE] Shoot\n" +
95             "[P] Pause\n\n" +

```

```

91         "Defeat enemies\n" +
92         "and survive.\n" +
93         "Good Luck!";
94     showMessage(tutorial);
95
96
97 }
98
99 public static void addScore(int points){
100     score += points;
101 }
102
103 public void setFiring(boolean firing) {
104     this.isFiring = firing;
105 }
106
107 public GameState getState() {
108     return state;
109 }
110
111 public void setState(GameState s) {
112     this.state = s;
113 }
114
115 public void update() {
116     if (state != GameState.PLAYING) return;
117
118     // Update background scrolling
119     if(background!= null) {
120         background.update();
121     }
122
123     checkLevelProgression();
124
125
126
127     // --- 連射ロジック ---
128     if (isFiring) {
129         if (shotTimer == 0) {
130             playerShoot();

```

```

131         shotTimer = GameConstants.ARROW_INTERVAL;
132     }
133 }
134 if (shotTimer > 0) {
135     shotTimer--;
136 }
137
138 // 無敵時間の更新
139 if (damageTimer > 0) {
140     damageTimer--;
141 }
142
143 if (ability1Timer > 0) {
144     ability1Timer--;
145 }
146
147 if (ability2Timer > 0) {
148     ability2Timer--;
149 }
150
151 if (ability3Timer > 0) {
152     ability3Timer--;
153 }
154
155 // --- NEW SPAWN LOGIC ---
156 // Iterate through all active spawners
157 if (!isBossActive) {
158     for (EnemySpawner spawner : activeSpawners) {
159         // If spawner says "True", create that enemy
160         if (spawner.update()) {
161             spawnMinion(spawner.getEnemyType());
162         }
163     }
164 }
165
166 // オブジェクト追加
167 objects.addAll(newObjectsBuffer);
168 newObjectsBuffer.clear();
169
170 // 移動

```

```

171     for (GameObject obj : objects) {
172         obj.move();
173     }
174
175     //当たり判定
176     checkCollisions();
177
178     //削除
179     objects.removeIf(obj -> obj.isDead());
180 }
181
182 /**
183 * Helper method to instantiate the correct enemy based on Class type.
184 */
185 private void spawnMinion(Class<? extends Minion> type) {
186     int x,y;
187     if (type == Harpy.class) {
188         x = rand.nextInt(GameConstants.WINDOW_WIDTH - GameConstants.
189 HARPY_WIDTH); // Simple random X
190         y = GameConstants.HUD_HEIGHT - GameConstants.HARPY_HEIGHT; // Start at top
191         Harpy h = new Harpy(x, y, this);
192         newObjectsBuffer.add(h);
193     } else if (type == Cyclops.class) {
194         x = rand.nextInt(GameConstants.WINDOW_WIDTH - GameConstants.
195 CYCLOPS_WIDTH); // Simple random X
196         y = GameConstants.HUD_HEIGHT - GameConstants.CYCLOPS_HEIGHT;
197         Cyclops g = new Cyclops(x, y, this); // Pass 'this' (GameModel)
198         newObjectsBuffer.add(g);
199     }
200 }
201
202 /**
203 * Allows enemies (Minions/Bosses) to add projectiles to the game.
204 * Using a specific method is safer than exposing the whole list.
205 */
206 public void spawnEnemyProjectile(Projectile p) {
207     if (p != null) {
        newObjectsBuffer.add(p);
    }
}

```

```

208     }
209 }
210
211
212     private void checkLevelProgression() {
213         // ボスがいたら、何もしない
214         if (isBossActive) return;
215         //次のフェーズがなかつたら return
216         if (currentLevelIndex >= GameConstants.LEVEL_MILESTONES.length) return
217         ;
218
219         if (score >= nextTargetScore){
220             //apply the effect we decided in the applyLevelEffects function
221             applyLevelEffects(currentLevelIndex);
222             currentLevelIndex++;
223             System.out.println("The current level index is: " +
224             currentLevelIndex);
225             nextTargetScore = GameConstants.LEVEL_MILESTONES[currentLevelIndex
226             ];
227         }
228     }
229
230
231     private void applyLevelEffects(int levelIndex) {
232         switch (levelIndex) {
233             case 0: //START OF THE GAME
234                 System.out.println("Start of the Game");
235                 break;
236
237             case 1:
238                 System.out.println("Difficulty UP!");
239                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
240                 (0.8);
241                 break;
242
243             case 2:
244                 System.out.println("Difficulty UP!");
245                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
246                 (0.8);
247                 break;
248         }
249     }

```

```

243     case 3:
244         showMessage("WARNING!\n\nBOSS DETECTED:\nAPOLLO\n\nPrepare for
245         battle!");
246         isBossActive = true;
247         clearEverything();
248         spawnApollo();
249         healPlayer();
250         break;
251     case 4:
252         showMessage("STAGE 1 CLEARED!\n\nEntering the Heavens...\n\
253         nArrow Damage doubled!\n\n Press [1] to use\nABILITY 1:\nAPOLLO'S SUN");
254         if (background != null) {
255             clearEverything();
256             healPlayer();
257             background.setImage(ResourceManager.stage2Img);
258             background.setSpeed(GameConstants.SCREEN_SPEED);
259             ability1Timer = 0;
260             this.currentStage = 2;
261
262             // DOUBLE ARROW DAMAGE
263             arrowDamage*=2;
264
265             // Clear old spawners (remove Stage 1 config)
266             activeSpawners.clear();
267             // Harpy Spawner
268             activeSpawners.add(new EnemySpawner(Harpy.class, 100, 50))
269 ;
270             // Cyclops Spawner
271             activeSpawners.add(new EnemySpawner(Cyclops.class,
272                             GameConstants.CYCLOPS_SPAWN_INTERVAL,
273                             GameConstants.CYCLOPS_SPAWN_VARIANCE));
274
275         }
276         break;
277     case 5:
278         System.out.println("Difficulty UP!");
279         for (EnemySpawner s : activeSpawners) s.increaseDifficulty
280 (0.9);
281         break;
282     case 6:

```

```

279     System.out.println("Difficulty UP!");
280     for (EnemySpawner s : activeSpawners) s.increaseDifficulty
281     (0.9);
282     break;
283   case 7:
284     showMessage("WARNING!\n\nBOSS DETECTED:\nZEUS\n\nPrepare for
285     battle!");
286     isBossActive = true;
287     clearEverything();
288     spawnZeus();
289     healPlayer();
290     break;
291   case 8:
292     showMessage("STAGE 2 CLEARED!\n\nEntering the INFERNO...\n\n
293     Shooting speed increased!\n\nPress [2] to use\nABILITY 2:\nZEUS'S
294     LIGHTING");
295     if (background != null) {
296       clearEverything();
297       background.setImage(ResourceManager.stage3Img);
298       background.setSpeed(0);
299       ability2Timer = 0;
300       arrowInterval = GameConstants.ARROW_INTERVAL2;
301       this.currentStage = 3;
302       activeSpawners.clear();
303       activeSpawners.add(new EnemySpawner(Harpy.class, 100, 50))
304     ;
305     // Cyclops Spawner
306     activeSpawners.add(new EnemySpawner(Cyclops.class,
307       GameConstants.CYCLOPS_SPAWN_INTERVAL,
308       GameConstants.CYCLOPS_SPAWN_VARIANCE));
309   }
310   break;
311 case 9:
312 }
313

public void showMessage(String text) {
  // Split the text by newline character to handle multiple lines
  this.currentMessageLines = text.split("\n");

```

```

314     this.state = GameState.MESSAGE;
315 }
316
317 public void bossDefeated(){
318     System.out.println("BOSS DEFEATED! Stage clear.");
319     this.isBossActive = false;
320     healPlayer();
321 }
322
323 // 全ての敵を消すメソッド
324 private void clearEverything() {
325     // 敵または羽の場合、消す"""
326     objects.removeIf(obj -> obj instanceof HostileEntity || obj instanceof
327     Projectile);
328     newObjectsBuffer.removeIf(obj -> obj instanceof HostileEntity || obj
329     instanceof Projectile);
330 }
331
332 // プレイヤーが撃つ（から呼ばれる）Controller
333 public void playerShoot() {
334     if (!isGameOver) {
335         // プレイヤーの中央上から発射
336         Arrow a = new Arrow(player.getX() + (GameConstants.PLAYER_WIDTH-
337         GameConstants.ARROW_WIDTH)/2, player.getY() - GameConstants.ARROW_HEIGHT,
338         arrowDamage);
339         newObjectsBuffer.add(a);
340     }
341 }
342
343 public void ability1(){
344     if (ability1Timer > 0) return;
345     ability1Timer = GameConstants.ABILITY1TIMER;
346     Sun sun = new Sun(
347         player.getX() + GameConstants.PLAYER_WIDTH / 2,
348         player.getY(),
349         0, false, true
350     );      newObjectsBuffer.add(sun);
351     System.out.println("Player shoots sun");
352 }
353

```

```

350     public void ability2(){
351         if (ability2Timer > 0) return;
352         ability2Timer = GameConstants.ABILITY2TIMER;
353         Lighting l = new Lighting(
354             player.getX() + GameConstants.PLAYER_WIDTH / 2,
355             player.getY(),
356             0, false, true, false
357         );           newObjectsBuffer.add(l);
358         System.out.println("Player shoots Lighting");
359     }
360
361     private void spawnApollo() {
362         Apollo apollo = new Apollo(this);
363         objects.add(apollo);
364         System.out.println("APOLLO HAS DESCENDED!");
365     }
366
367     public void spawnZeus() {
368         Zeus zeus = new Zeus(this);
369         objects.add(zeus);
370         System.out.println("ZEUS HAS DESCENDED!");
371     }
372
373     // ダメージ処理メソッド
374     private void playerTakesDamage() {
375         if (damageTimer == 0) { // 無敵時間中でなければダメージ
376             lives--;
377             damageTimer = 120; // フレーム（約秒）無敵にする 1803
378             System.out.println("Damage taken! Lives remaining: " + lives);
379
380             if (lives <= 0) {
381                 state = GameState.GAMEOVER;
382                 System.out.println("GAME OVER");
383             }
384         }
385     }
386
387     private void healPlayer(){
388         lives = GameConstants.PLAYER_MAX_LIVES;
389     }

```

```

390
391
392     private boolean checkIntersection(GameObject obj1, GameObject obj2) {
393         // 1. take the precise shapes
394         Shape s1 = obj1.getShape();
395         Shape s2 = obj2.getShape();
396
397         // 2. Quick check: if the outer rectangles don't touch we skip the
398         // complicated calculations
398         if (!s1.getBounds2D().intersects(s2.getBounds2D())) {
399             return false;
400         }
401
402         // 3. Precise Calculation
403         Area area1 = new Area(s1);
404         Area area2 = new Area(s2);
405
406         area1.intersect(area2);
407
408         // if the area is not empty it means they are touching
409         return !area1.isEmpty();
410     }
411
412     /**
413      * Centralized Collision Logic.
414      * Iterates through all objects to check for intersections.
415      */
416     private void checkCollisions() {
417         for (int i = 0; i < objects.size(); i++) {
418             GameObject objA = objects.get(i);
419             if (objA.isDead()) continue;
420
421             for (int j = i + 1; j < objects.size(); j++) {
422                 GameObject objB = objects.get(j);
423                 if (objB.isDead()) continue;
424
425                 if (checkIntersection(objA, objB)) {
426                     handleCollision(objA, objB);
427                 }
428             }
429         }
430     }

```

```

429     }
430 }
431
432 /**
433 * Handles the specific logic when two objects collide.
434 * Uses Projectile Power Levels and Alignment to determine the outcome.
435 */
436 private void handleCollision(GameObject a, GameObject b) {
437
438     // --- CASE 1: PROJECTILE vs PROJECTILE ---
439     if (a instanceof Projectile && b instanceof Projectile) {
440         Projectile p1 = (Projectile) a;
441         Projectile p2 = (Projectile) b;
442
443         // Same team projectiles do not destroy each other
444         if (p1.getAlignment() == p2.getAlignment()) return;
445
446         // Compare Power Levels to see who survives
447         if (p1.getPowerLevel() > p2.getPowerLevel()) {
448             p2.setDead(); // p1 dominates
449
450             // SUN CASE: p1 is a Sun, it takes damage equal to p2's damage
451             if (p1 instanceof BossProjectile) {
452                 ((BossProjectile) p1).reduceHealth(p2.getDamage());
453             }
454
455         } else if (p2.getPowerLevel() > p1.getPowerLevel()) {
456             p1.setDead(); // p2 dominates
457
458             // SUN CASE: If p2 is a Sun, it takes damage equal to p1's
459             // damage
460             if (p2 instanceof BossProjectile) {
461                 ((BossProjectile) p2).reduceHealth(p1.getDamage());
462             }
463
464         } else {
465             // Equal power (e.g., Arrow vs Feather) -> Both destroyed
466             p1.setDead();
467             p2.setDead();
468     }

```

```

468         return;
469     }
470
471     // --- CASE 2: PROJECTILE vs LIVING ENTITY (Player or Enemy) ---
472     Projectile proj = null;
473     GameObject entity = null;
474
475     // Identify which is which
476     if (a instanceof Projectile) { proj = (Projectile) a; entity = b; }
477     else if (b instanceof Projectile) { proj = (Projectile) b; entity = a; }
478
479     if (proj != null) {
480         // Sub-case A: Projectile hits Player
481         if (entity instanceof Player) {
482             if (proj.getAlignment() == Alignment.ENEMY) {
483                 playerTakesDamage();
484                 if (!proj.isPenetrating()) proj.setDead();
485             }
486         }
487         // Sub-case B: Projectile hits Enemy (Harpy, Apollo, Golem, etc.)
488         else if (entity instanceof HostileEntity) {
489             HostileEntity enemy = (HostileEntity) entity;
490
491             // Only damage if the projectile belongs to the Player
492             if (proj.getAlignment() == Alignment.PLAYER) {
493                 enemy.takeDamage(proj.getDamage());
494
495                 // SUN CASE: If the projectile is a Sun, it also loses HP
upon contact
496                 if (proj instanceof BossProjectile) {
497                     ((BossProjectile) proj).reduceHealth(1);
498                 }
499
500                 else if (!proj.isPenetrating()) {
501                     proj.setDead();
502                 }
503             }
504         }
505     }

```

```

506     }
507
508     // --- CASE 3: PHYSICAL COLLISION (Player vs Enemy Body) ---
509     if ((a instanceof Player && b instanceof HostileEntity) ||
510         (b instanceof Player && a instanceof HostileEntity)) {
511         playerTakesDamage();
512     }
513 }
514
515
516 // 無敵時間中かどうか
517 public boolean isInvincible() {
518     return damageTimer > 0;
519 }
520
521 public int getAbilityNthTimer(int n) {
522     switch (n){
523         case 1:
524             return ability1Timer;
525         case 2:
526             return ability2Timer;
527         case 3:
528             return ability3Timer;
529         default:
530             System.out.println("getability ERROR");
531     }
532     return ability1Timer;
533 }
534
535 public boolean isAbilityUnclocked(int abilityIndex) {
536     // Logic for Ability 1 (Sun)
537     if (abilityIndex == 1) {
538         // Unlocks after defeating the first boss (Apollo)
539         // Apollo is Level Index 4. So > 4 means Stage 2 started.
540         return this.currentLevelIndex > 4;
541     }
542     // Logic for Ability 2 (Lighting)
543     if (abilityIndex == 2) {
544         // Unlocks after defeating the second boss (Zeus)
545         // Zeus is Level Index 8. So > 8 means Stage 3 started.

```

```

546         return this.currentLevelIndex > 8;
547     }
548
549     // Future logic for Ability 2 and 3
550     return false;
551 }
552
553 public void resumeGame() {
554     this.state = GameState.PLAYING;
555 }
556
557 //SETTERS & GETTERS
558 public ArrayList<GameObject> getObjects() {
559     return objects;
560 }
561
562 public Player getPlayer() {
563     return player;
564 }
565
566 public boolean isGameOver() {
567     return isGameOver;
568 }
569
570 public int getScore() {
571     return score;
572 }
573
574 public int getLives() {
575     return lives;
576 }
577
578 public Background getBackground() {
579     return background;
580 }
581
582 public String getStageText(){
583     if (currentStage > 3) {
584         return "EXTRA STAGE";
585     }

```

```

586     return "STAGE " + currentStage;
587 }
588
589 public int getCurrentLevelIndex(){
590     return this.currentLevelIndex;
591 }
592
593 public String[] getCurrentMessageLines() {
594     return currentMessageLines;
595 }
596
597 }

```

Listing 4: 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;

```

```
24  
25 //ABILITY TIMERS  
26 public static final int ABILITY1TIMER = FPS * 15; // 15 seconds  
27 public static final int ABILITY2TIMER = FPS * 8; // 10 seconds  
28 public static final int ABILITY3TIMER = FPS * 10; // 10 seconds  
29  
30 // の設定 ARROW  
31 public static final int ARROW_WIDTH = 10;  
32 public static final int ARROW_HEIGHT = 70; // set to the image height  
width ration  
33 public static final int ARROW_SPEED = 30;  
34 public static final int ARROW_INTERVAL = 20;  
35 public static final int ARROW_INTERVAL2 = 15;  
36 public static final int ARROW_DAMAGE = 1;  
37  
38 // *****  
39 // の設定 MINIONS  
40 // *****  
41  
42 // の設定 HARPY  
43 public static final int HARPY_WIDTH = 100;  
44 public static final int HARPY_HEIGHT = HARPY_WIDTH *1911/1708; // set to  
the image height width ration  
45 public static final int HARPY_XSPEED = 4;  
46 public static final int HARPY_YSPEED = 2;  
47 public static final int HARPY_HP = 2;  
48 public static final int HARPY_SCORE_POINTS = 10;  
49 public static final int HARPY_SPAWN_INTERVAL = 120;  
50 public static final int HARPY_SPAWN_VARIANCE = HARPY_SPAWN_INTERVAL / 2;  
51  
52 // の設定 FEATHER  
53 public static final int FEATHER_WIDTH = 12;  
54 public static final int FEATHER_HEIGHT = FEATHER_WIDTH * 1698/378; // set  
to the image height width ration  
55 public static final int FEATHER_SPEED = 7;  
56 public static final int FEATHER_FIRE_INTERVAL = 90;  
57 public static final int FEATHER_FIRE_VARIANCE = FEATHER_FIRE_INTERVAL / 2;  
58 public static final int FEATHER_DAMAGE = 1;  
59  
60 // の設定 CYCLOPS
```

```
61     public static final int CYCLOPS_WIDTH = 150;
62     public static final int CYCLOPS_HEIGHT = CYCLOPS_WIDTH;
63     public static final double CYCLOPS_YSPEED = 2;
64     public static final int CYCLOPS_HP = 10;
65     public static final int CYCLOPS_SCORE_POINTS = 50;
66     public static final int CYCLOPS_SPAWN_INTERVAL = 400;
67     public static final int CYCLOPS_SPAWN_VARIANCE = CYCLOPS_SPAWN_INTERVAL /
68         2;
68     public static final int CYCLOPS_MOVEMENT_TIMER = 20;
69     public static final int CYCLOPS_ATTACK_TIMER = 60;

70
71 // の設定 BOULDER
72     public static final int BOULDER_WIDTH = 100;
73     public static final int BOULDER_HEIGHT = BOULDER_WIDTH;
74     public static final double BOULDER_INITIAL_SPEED = 0;
75     public static final double BOULDER_GRAVITY = 0.3;
76     public static final int BOULDER_DAMAGE = 5;

77
78 // ****
79 // の設定 BOSS
80 // ****

81
82 // の設定 APOLLO
83     public static final int APOLLO_WIDTH = 200;
84     public static final int APOLLO_HEIGHT = APOLLO_WIDTH * 1556 / 2463;
85     public static final int APOLLO_SPEED1 = 4;
86     public static final int APOLLO_SPEED2 = APOLLO_SPEED1 * 3 / 2;
87     public static final int APOLLO_HP = 2; //50
88     public static final int APOLLO_SCORE_POINTS = 1000;

89
90 // の設定 SUN
91     public static final int SUN_WIDTH = 150;
92     public static final int SUN_HEIGHT = SUN_WIDTH;
93     public static final double SUN_SPEED1 = 6;
94     public static final double SUN_SPEED2 = SUN_SPEED1 * 3 / 2;
95     public static final int SUN_DAMAGE = 1;
96     public static final int SUN_HP = 20;

97
98 // の設定 ZEUS
99     public static final int ZEUS_SPEED = 4;
```

```
100 public static final int ZEUS_SPEED2 = 8;
101 public static final int ZEUS_WIDTH = 150;
102 public static final int ZUES_HEIGHT = 150;
103 public static final int ZEUS_HP = 10; // 100
104 public static final int ZEUS_SCORE_POINTS = 1500;
105 public static final int ZEUS_SHOOT_TIMER = 60;
106 public static final int ZEUS_SHOOT_TIMER2 = 40;
107 public static final int ZEUS_ABILITY1_PAUSE = 40;
108 public static final int ZEUS_ABILITY1_PAUSE2 = 25;
109 public static final int ZEUS_ABILITY2_TIMER = FPS * 6; // 10 seconds
110
111 //の設定 LIGHTING
112 public static final int LIGHTING_WIDTH = 25;
113 public static final int LIGHTING_HEIGHT = 150;
114 public static final int LIGHTING_SPEED1 = 10;
115 public static final int LIGHTING_SPEED2 = 15;
116 public static final int LIGHTING_DAMAGE = 1;
117 public static final int LIGHTING_HP = 10;
118
119 //の設定 SCORE
120 // STAGE 1
121 public static final int SCORE_STAGE1_PHASE1 = 0;
122 public static final int SCORE_STAGE1_PHASE2 = HARPY_SCORE_POINTS * 1; // 100
123 public static final int SCORE_STAGE1_PHASE3 = HARPY_SCORE_POINTS * 3; // 300
124 public static final int SCORE_FOR_BOSS_1 = HARPY_SCORE_POINTS * 5; // 500
125 // STAGE 2
126 public static final int SCORE_STAGE2_PHASE1 = SCORE_FOR_BOSS_1 +
APOLLO_SCORE_POINTS; //1500
127 public static final int SCORE_STAGE2_PHASE2 = SCORE_STAGE2_PHASE1 + 5; // 2000
128 public static final int SCORE_STAGE2_PHASE3 = SCORE_STAGE2_PHASE2 + 7; // 2750
129 public static final int SCORE_FOR_BOSS_2 = SCORE_STAGE2_PHASE3 + 7; // 3500
130 // STAGE 3
131 public static final int SCORE_STAGE3_PHASE1 = SCORE_FOR_BOSS_2 +
ZEUS_SCORE_POINTS; // 5000
132 public static final int SCORE_STAGE3_PHASE2 = 100000; // 6000
```

```

133     public static final int SCORE_STAGE3_PHASE3 = 300000; // 7000
134     public static final int SCORE_FOR_BOSS_3 = 500000; // 8000
135     // EXTRA STAGE
136     public static final int SCORE_EXTRA_STAGE = 10000; // 10000
137
138     // We put them all in an Array
139     public static final int[] LEVEL_MILESTONES = {
140         SCORE_STAGE1_PHASE1, SCORE_STAGE1_PHASE2, SCORE_STAGE1_PHASE3,
141         SCORE_FOR_BOSS_1,
142         SCORE_STAGE2_PHASE1, SCORE_STAGE2_PHASE2, SCORE_STAGE2_PHASE3,
143         SCORE_FOR_BOSS_2,
144         SCORE_STAGE3_PHASE1, SCORE_STAGE3_PHASE2, SCORE_STAGE3_PHASE3,
145         SCORE_FOR_BOSS_3,
146         SCORE_EXTRA_STAGE
147     };
148
149     //他の設定
150     public static final int FLASH_TIMER = 5;
151     private GameConstants(){} //オブジェクトを作らないようにする private
}

```

Listing 5: 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 6: 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 7: 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 8: 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
56         // below it.
57
58         // We assume the pattern is: [Normal] [Flipped] [Normal] [Flipped]
59         ...
60
61         // 1. Draw Normal Tile at current Y
62         // Covers: y to y+400  x

```

```

61     drawForceSize(g, currentY, false);

62
63     // 2. Draw Flipped Tile BELOW
64     // Covers: y+400 to y+800
65     drawForceSize(g, currentY + HEIGHT, true);

66
67     // 3. Draw Flipped Tile ABOVE
68     // Covers: y-400 to y. (Crucial for when y starts at 0 or is small
69 )
70     drawForceSize(g, currentY - HEIGHT, true);

71     // 4. Draw Normal Tile ABOVE that
72     // Covers: y-800 to y-400. (Crucial for the loop wrap-around)
73     drawForceSize(g, currentY - (HEIGHT * 2), false);
74 } else {
75     g.drawImage(image, 0, GameConstants.HUD_HEIGHT, WIDTH, HEIGHT * 2,
76     null);
77 }
78 }

79 /**
80 * Helper method to draw a tile either normally or vertically flipped.
81 * @param g Graphics context
82 * @param yPos The Y position to draw at
83 * @param isFlipped If true, draws the image upside down (mirrored)
84 */
85 private void drawForceSize(Graphics g, int yPos, boolean isFlipped) {
86     if (!isFlipped) {
87         // NORMAL: Force width 600, height 400
88         g.drawImage(image, 0, yPos, WIDTH, HEIGHT, null);
89     } else {
90         // FLIPPED: Force width 600, height 400 (but drawn upwards)
91         // Destination Y starts at bottom (yPos + HEIGHT)
92         // Height is negative (-HEIGHT) to flip it
93         g.drawImage(image, 0, yPos + HEIGHT, WIDTH, -HEIGHT, null);
94     }
95 }

96
97 private boolean isStage2Img(){
98     return (this.image == ResourceManager.stage2Img);

```

```
99     }
100 }
```

Listing 9: Projectile.java

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

```

32     this.powerLevel = powerLevel;
33     this.damage = damage;
34     this.isPenetrating = false; // Default: destroys itself on impact
35   }
36
37   // Getters
38   public Alignment getAlignment() { return alignment; }
39   public int getPowerLevel() { return powerLevel; }
40   public int getDamage() { return damage; }
41   public boolean isPenetrating() { return isPenetrating; }
42 }
```

Listing 10: 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;
```

```

26     if (x > GameConstants.WINDOW_WIDTH - width) x = GameConstants.
27     WINDOW_WIDTH - width;
28     if (y < GameConstants.HUD_HEIGHT) y = GameConstants.HUD_HEIGHT;
29     if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height)
30     y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height;
31   }
32
33   @Override
34   public void draw(Graphics g) {
35     if(image != null){
36       g.drawImage(image, x, y, width, height, null);
37     }
38     else{
39       // Fallback if image failed to load
40       g.setColor(Color.BLUE);
41       g.fillRect(x, y, width, height);
42     }
43
44   @Override
45   public Shape getShape() {
46     // we define the margin
47     float paddingX = width * 0.3f; // remove 20 % width
48     float paddingY = height * 0.2f; // remove 10 % height
49
50     // the smaller hitbox get centered compared to the original immage
51     return new Ellipse2D.Float(
52       x + paddingX / 2,           // move right
53       y + paddingY / 2,           // move down
54       width - paddingX,          // reduce width
55       height - paddingY         // reduce height
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 11: Arrow.java

```

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

```

29     } else {
30         g.setColor(Color.YELLOW);
31         g.fillRect(x, y, width, height);
32     }
33 }
34 }
```

Listing 12: BossProjectile.java

```

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

Listing 13: 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 14: 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)

```

```

20     // Parameters: x, y, width, height, HP, Score Points
21     super(
22         (GameConstants.WINDOW_WIDTH - GameConstants.APOLLO_WIDTH) / 2,
23         // Start in the middle
24         GameConstants.HUD_HEIGHT, // Start below HUD
25         GameConstants.APOLLO_WIDTH,
26         GameConstants.APOLLO_HEIGHT,
27         ResourceManager.apolloImg,
28         GameConstants.APOLLO_HP,
29         GameConstants.APOLLO_SCORE_POINTS, // Score awarded when
30         defeated
31         model
32     );
33     this.image = ResourceManager.apolloImg;
34 }
35
36 @Override
37 public void move() {
38     super.move();
39     // Update horizontal position
40     x += speedX;
41
42     // Bounce logic: If it hits the screen edges
43     if (x <= 0 || x >= GameConstants.WINDOW_WIDTH - width) {
44         speedX = -speedX; // Reverse direction
45
46         // Trigger shooting mechanism via GameModel
47         // We pass the current phase status to decide if the Sun should be
48         Red/Fast
49         shootSun();
50     }
51 }
52
53 private void shootSun(){
54     Sun s = new Sun(x, y, speedX, secondPhase, false);
55     model.spawnEnemyProjectile(s);
56 }
57
58 @Override
59 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
60     // 2. Specific Logic for Apollo: Check for Second Phase (Rage Mode)
61     // If HP drops below 50% and we are not yet in the second phase...
62     if (hp <= maxHp / 2 && !secondPhase) {
63         image = ResourceManager.apolloImg2; // Change sprite to Red Apollo
64         speedX = (speedX > 0) ? GameConstants.APOLLO_SPEED2 : -
GameConstants.APOLLO_SPEED2; // Double the movement speed
65         secondPhase = true; // Activate the flag
66         System.out.println("Apollo entering Phase 2!");
67     }
68 }
69
70 @Override
71 public void draw(Graphics g) {
72     BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
apolloHitImg : image;
73
74     if (image != null) {
75         // Directional Flipping Logic
76         if (speedX > 0) {
77             // Moving RIGHT: Draw normally
78             g.drawImage(imgToDraw, x, y, width, height, null);
79         } else {
80             // Moving LEFT: Flip the image horizontally
81             // We draw starting at (x + width) and use a negative width to
flip
82             g.drawImage(imgToDraw, x + width, y, -width, height, null);
83         }
84     } else {
85         // Fallback: Draw an Orange rectangle if images fail to load
86         g.setColor(Color.ORANGE);
87         g.fillRect(x, y, width, height);
88     }
89 }
90 }
```

Listing 15: 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.
24              sunImg,
25              friendly ? Alignment.PLAYER : Alignment.ENEMY,
26              3,
27              GameConstants.SUN_DAMAGE);
28
29
30        this.isPlayerProjectile = friendly;
31        this.isPenetrating = true;
32
33        if (isPlayerProjectile) {
34            maxHP = GameConstants.SUN_HP;
35            currentHP = maxHP;
36        }
37
38        this.initialSize = GameConstants.SUN_WIDTH;
39
40        velX = 1;

```

```

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     if (x + currentRadius > GameConstants.WINDOW_WIDTH) {
105         x = (int)(GameConstants.WINDOW_WIDTH - currentRadius);
106         preciseX = x;
107         velX = -velX;
108     }
109
110     // Despawn
111     if (y - currentRadius > GameConstants.FIELD_HEIGHT + GameConstants.
112 HUD_HEIGHT ||
113             y + currentRadius < GameConstants.HUD_HEIGHT) {
114         isDead = true;
115     }

```

```

115     }
116
117     // Helper to calculate size based on HP
118     private double getCurrentSize() {
119         if (!isPlayerProjectile) return initialSize;
120
121         // Calculate ratio: currentHP / maxHP
122         double ratio = (double) currentHP / maxHP;
123         return initialSize * ratio;
124     }
125
126     @Override
127     public void draw(Graphics g) {
128         double size = getCurrentSize();
129         double radius = size / 2.0;
130
131         int drawX = (int) (x - radius);
132         int drawY = (int) (y - radius);
133         int drawSize = (int) size;
134
135         if (image != null) {
136             g.drawImage(image, drawX, drawY, drawSize, drawSize, null);
137         } else {
138             g.setColor(Color.YELLOW);
139             g.fillOval(x, y, width, height);
140         }
141     }
142
143     @Override
144     public Shape getShape() {
145         double size = getCurrentSize();
146         double radius = size / 2.0;
147         return new Ellipse2D.Float((float)(x - radius), (float)(y - radius), (float)size, (float)size);
148     }
149
150 }
```

Listing 16: 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;
105            resetShootTimer();
106        }
107
108        if (secondPhase && ability2Timer > 0){
109            ability2Timer--;
110        }
111    }
112
113    private void ability1() {
114        if (ability1Position == 1){
115            ability1Phase = true;
116        }
117    }

```

```

118     if (ability1Position == 5) {
119         setAbility1Timer();
120         ability1Position = 1;
121         resetShootTimer();
122         return;
123     }
124
125     if(speedX > 0) {
126         x = ability1Position * (GameConstants.WINDOW_WIDTH-width)/4;
127     } else {
128         x = (4 - ability1Position) * (GameConstants.WINDOW_WIDTH-width)/4;
129     }
130     if (ability1Position % 2 == 0){
131         y = GameConstants.HUD_HEIGHT;
132     } else {
133         y = GameConstants.HUD_HEIGHT + height / 3;
134     }
135
136     shootLighting();
137     ability1Pause = maxAbility1Pause;
138     ability1Position++;
139 }
140
141 private void shootLighting(){
142     Lighting l = new Lighting(x, y, speedX, secondPhase, false,
143     ability2Started);
144     model.spawnEnemyProjectile(l);
145 }
146
147 @Override
148 public void takeDamage(int dmg) {
149     super.takeDamage(dmg);
150
151     if (hp <= maxHp / 2 && !secondPhase) {
152         image = ResourceManager.zeusImg2;
153         hitImg = ResourceManager.zeusHitImg2;
154         speedX = (speedX > 0) ? GameConstants.ZEUS_SPEED2 : -GameConstants
155 .ZEUS_SPEED2; // Double the movement speed
156         secondPhase = true;
157         maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER2;

```

```

156         maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE2;
157     }
158 }
159
160 private void setAbility1Timer(){
161     Random random = new Random();
162     ability1Timer = random.nextInt(4) + 1;;
163     ability1Phase = false;
164 }
165
166 private void resetShootTimer(){
167     shootTimer = maxShootTimer;
168 }
169
170 @Override
171 public void draw(Graphics g) {
172     BufferedImage imgToDraw = (flashTimer > 0) ? hitImg : image;
173
174     if (image != null) {
175         if (speedX > 0) {
176             g.drawImage(imgToDraw, x, y, width, height, null);
177         } else {
178             g.drawImage(imgToDraw, x + width, y, -width, height, null);
179         }
180     } else {
181         g.setColor(Color.ORANGE);
182         g.fillRect(x, y, width, height);
183     }
184 }
185 }
```

Listing 17: 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
23        this.isPlayerProjectile = friendly;
24        this.isPenetrating = true;
25
26
27        if (isSecondPhase) {
28            vely = GameConstants.LIGHTING_SPEED2;
29        } else {
30            vely = GameConstants.LIGHTING_SPEED1;
31        }
32
33
34        if(isPlayerProjectile){
35            vely = -vely;
36            maxHP = GameConstants.LIGHTING_HP;
37            currentHP = maxHP;
38            // set position for player
39            this.x = x + (GameConstants.PLAYER_WIDTH - width) / 2;
40            this.y = y - GameConstants.PLAYER_HEIGHT;
41        } else {
42            if (!ability2Started){
43
44                // set position for Zeus
45                this.x = (ZeusSpeedX > 0) ? x + GameConstants.ZEUS_WIDTH - width :
46                x + width;
47            } else {
48                this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
49                width;
50            }
51        }
52    }
53
54    public void update() {
55        if (isPenetrating) {
56            currentHP -= vely;
57            if (currentHP <= 0) {
58                isPenetrating = false;
59            }
60        }
61    }
62
63    public void draw() {
64        if (isPlayerProjectile) {
65            spriteBatch.draw(lightingImg, x, y, width, height);
66        } else {
67            spriteBatch.draw(lightingImg2, x, y, width, height);
68        }
69    }
70
71    public void setVelY(int vely) {
72        this.vely = vely;
73    }
74
75    public void setMaxHP(int maxHP) {
76        this.maxHP = maxHP;
77    }
78
79    public void setCurrentHP(int currentHP) {
80        this.currentHP = currentHP;
81    }
82
83    public void setWidth(int width) {
84        this.width = width;
85    }
86
87    public void setHeight(int height) {
88        this.height = height;
89    }
90
91    public void setX(int x) {
92        this.x = x;
93    }
94
95    public void setY(int y) {
96        this.y = y;
97    }
98
99    public void setAlignment(Alignment alignment) {
100        this.alignment = alignment;
101    }
102
103    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
104        this.isPlayerProjectile = isPlayerProjectile;
105    }
106
107    public void setIsPenetrating(boolean isPenetrating) {
108        this.isPenetrating = isPenetrating;
109    }
110
111    public void setIsSecondPhase(boolean isSecondPhase) {
112        this.isSecondPhase = isSecondPhase;
113    }
114
115    public void setAbility2Started(boolean ability2Started) {
116        this.ability2Started = ability2Started;
117    }
118
119    public void setZeusSpeedX(int zeusSpeedX) {
120        this.ZeusSpeedX = zeusSpeedX;
121    }
122
123    public void setLightingImg(SpriteImage lightingImg) {
124        this.lightingImg = lightingImg;
125    }
126
127    public void setLightingImg2(SpriteImage lightingImg2) {
128        this.lightingImg2 = lightingImg2;
129    }
130
131    public void setResourceManager(ResourceManager resourceManager) {
132        this.resourceManager = resourceManager;
133    }
134
135    public void setWidth(int width) {
136        this.width = width;
137    }
138
139    public void setHeight(int height) {
140        this.height = height;
141    }
142
143    public void setX(int x) {
144        this.x = x;
145    }
146
147    public void setY(int y) {
148        this.y = y;
149    }
150
151    public void setAlignment(Alignment alignment) {
152        this.alignment = alignment;
153    }
154
155    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
156        this.isPlayerProjectile = isPlayerProjectile;
157    }
158
159    public void setIsPenetrating(boolean isPenetrating) {
160        this.isPenetrating = isPenetrating;
161    }
162
163    public void setIsSecondPhase(boolean isSecondPhase) {
164        this.isSecondPhase = isSecondPhase;
165    }
166
167    public void setAbility2Started(boolean ability2Started) {
168        this.ability2Started = ability2Started;
169    }
170
171    public void setZeusSpeedX(int zeusSpeedX) {
172        this.ZeusSpeedX = zeusSpeedX;
173    }
174
175    public void setLightingImg(SpriteImage lightingImg) {
176        this.lightingImg = lightingImg;
177    }
178
179    public void setLightingImg2(SpriteImage lightingImg2) {
180        this.lightingImg2 = lightingImg2;
181    }
182
183    public void setResourceManager(ResourceManager resourceManager) {
184        this.resourceManager = resourceManager;
185    }
186
187    public void setWidth(int width) {
188        this.width = width;
189    }
190
191    public void setHeight(int height) {
192        this.height = height;
193    }
194
195    public void setX(int x) {
196        this.x = x;
197    }
198
199    public void setY(int y) {
200        this.y = y;
201    }
202
203    public void setAlignment(Alignment alignment) {
204        this.alignment = alignment;
205    }
206
207    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
208        this.isPlayerProjectile = isPlayerProjectile;
209    }
210
211    public void setIsPenetrating(boolean isPenetrating) {
212        this.isPenetrating = isPenetrating;
213    }
214
215    public void setIsSecondPhase(boolean isSecondPhase) {
216        this.isSecondPhase = isSecondPhase;
217    }
218
219    public void setAbility2Started(boolean ability2Started) {
220        this.ability2Started = ability2Started;
221    }
222
223    public void setZeusSpeedX(int zeusSpeedX) {
224        this.ZeusSpeedX = zeusSpeedX;
225    }
226
227    public void setLightingImg(SpriteImage lightingImg) {
228        this.lightingImg = lightingImg;
229    }
230
231    public void setLightingImg2(SpriteImage lightingImg2) {
232        this.lightingImg2 = lightingImg2;
233    }
234
235    public void setResourceManager(ResourceManager resourceManager) {
236        this.resourceManager = resourceManager;
237    }
238
239    public void setWidth(int width) {
240        this.width = width;
241    }
242
243    public void setHeight(int height) {
244        this.height = height;
245    }
246
247    public void setX(int x) {
248        this.x = x;
249    }
250
251    public void setY(int y) {
252        this.y = y;
253    }
254
255    public void setAlignment(Alignment alignment) {
256        this.alignment = alignment;
257    }
258
259    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
260        this.isPlayerProjectile = isPlayerProjectile;
261    }
262
263    public void setIsPenetrating(boolean isPenetrating) {
264        this.isPenetrating = isPenetrating;
265    }
266
267    public void setIsSecondPhase(boolean isSecondPhase) {
268        this.isSecondPhase = isSecondPhase;
269    }
270
271    public void setAbility2Started(boolean ability2Started) {
272        this.ability2Started = ability2Started;
273    }
274
275    public void setZeusSpeedX(int zeusSpeedX) {
276        this.ZeusSpeedX = zeusSpeedX;
277    }
278
279    public void setLightingImg(SpriteImage lightingImg) {
280        this.lightingImg = lightingImg;
281    }
282
283    public void setLightingImg2(SpriteImage lightingImg2) {
284        this.lightingImg2 = lightingImg2;
285    }
286
287    public void setResourceManager(ResourceManager resourceManager) {
288        this.resourceManager = resourceManager;
289    }
290
291    public void setWidth(int width) {
292        this.width = width;
293    }
294
295    public void setHeight(int height) {
296        this.height = height;
297    }
298
299    public void setX(int x) {
300        this.x = x;
301    }
302
303    public void setY(int y) {
304        this.y = y;
305    }
306
307    public void setAlignment(Alignment alignment) {
308        this.alignment = alignment;
309    }
310
311    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
312        this.isPlayerProjectile = isPlayerProjectile;
313    }
314
315    public void setIsPenetrating(boolean isPenetrating) {
316        this.isPenetrating = isPenetrating;
317    }
318
319    public void setIsSecondPhase(boolean isSecondPhase) {
320        this.isSecondPhase = isSecondPhase;
321    }
322
323    public void setAbility2Started(boolean ability2Started) {
324        this.ability2Started = ability2Started;
325    }
326
327    public void setZeusSpeedX(int zeusSpeedX) {
328        this.ZeusSpeedX = zeusSpeedX;
329    }
330
331    public void setLightingImg(SpriteImage lightingImg) {
332        this.lightingImg = lightingImg;
333    }
334
335    public void setLightingImg2(SpriteImage lightingImg2) {
336        this.lightingImg2 = lightingImg2;
337    }
338
339    public void setResourceManager(ResourceManager resourceManager) {
340        this.resourceManager = resourceManager;
341    }
342
343    public void setWidth(int width) {
344        this.width = width;
345    }
346
347    public void setHeight(int height) {
348        this.height = height;
349    }
350
351    public void setX(int x) {
352        this.x = x;
353    }
354
355    public void setY(int y) {
356        this.y = y;
357    }
358
359    public void setAlignment(Alignment alignment) {
360        this.alignment = alignment;
361    }
362
363    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
364        this.isPlayerProjectile = isPlayerProjectile;
365    }
366
367    public void setIsPenetrating(boolean isPenetrating) {
368        this.isPenetrating = isPenetrating;
369    }
370
371    public void setIsSecondPhase(boolean isSecondPhase) {
372        this.isSecondPhase = isSecondPhase;
373    }
374
375    public void setAbility2Started(boolean ability2Started) {
376        this.ability2Started = ability2Started;
377    }
378
379    public void setZeusSpeedX(int zeusSpeedX) {
380        this.ZeusSpeedX = zeusSpeedX;
381    }
382
383    public void setLightingImg(SpriteImage lightingImg) {
384        this.lightingImg = lightingImg;
385    }
386
387    public void setLightingImg2(SpriteImage lightingImg2) {
388        this.lightingImg2 = lightingImg2;
389    }
390
391    public void setResourceManager(ResourceManager resourceManager) {
392        this.resourceManager = resourceManager;
393    }
394
395    public void setWidth(int width) {
396        this.width = width;
397    }
398
399    public void setHeight(int height) {
400        this.height = height;
401    }
402
403    public void setX(int x) {
404        this.x = x;
405    }
406
407    public void setY(int y) {
408        this.y = y;
409    }
410
411    public void setAlignment(Alignment alignment) {
412        this.alignment = alignment;
413    }
414
415    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
416        this.isPlayerProjectile = isPlayerProjectile;
417    }
418
419    public void setIsPenetrating(boolean isPenetrating) {
420        this.isPenetrating = isPenetrating;
421    }
422
423    public void setIsSecondPhase(boolean isSecondPhase) {
424        this.isSecondPhase = isSecondPhase;
425    }
426
427    public void setAbility2Started(boolean ability2Started) {
428        this.ability2Started = ability2Started;
429    }
430
431    public void setZeusSpeedX(int zeusSpeedX) {
432        this.ZeusSpeedX = zeusSpeedX;
433    }
434
435    public void setLightingImg(SpriteImage lightingImg) {
436        this.lightingImg = lightingImg;
437    }
438
439    public void setLightingImg2(SpriteImage lightingImg2) {
440        this.lightingImg2 = lightingImg2;
441    }
442
443    public void setResourceManager(ResourceManager resourceManager) {
444        this.resourceManager = resourceManager;
445    }
446
447    public void setWidth(int width) {
448        this.width = width;
449    }
450
451    public void setHeight(int height) {
452        this.height = height;
453    }
454
455    public void setX(int x) {
456        this.x = x;
457    }
458
459    public void setY(int y) {
460        this.y = y;
461    }
462
463    public void setAlignment(Alignment alignment) {
464        this.alignment = alignment;
465    }
466
467    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
468        this.isPlayerProjectile = isPlayerProjectile;
469    }
470
471    public void setIsPenetrating(boolean isPenetrating) {
472        this.isPenetrating = isPenetrating;
473    }
474
475    public void setIsSecondPhase(boolean isSecondPhase) {
476        this.isSecondPhase = isSecondPhase;
477    }
478
479    public void setAbility2Started(boolean ability2Started) {
480        this.ability2Started = ability2Started;
481    }
482
483    public void setZeusSpeedX(int zeusSpeedX) {
484        this.ZeusSpeedX = zeusSpeedX;
485    }
486
487    public void setLightingImg(SpriteImage lightingImg) {
488        this.lightingImg = lightingImg;
489    }
490
491    public void setLightingImg2(SpriteImage lightingImg2) {
492        this.lightingImg2 = lightingImg2;
493    }
494
495    public void setResourceManager(ResourceManager resourceManager) {
496        this.resourceManager = resourceManager;
497    }
498
499    public void setWidth(int width) {
500        this.width = width;
501    }
502
503    public void setHeight(int height) {
504        this.height = height;
505    }
506
507    public void setX(int x) {
508        this.x = x;
509    }
510
511    public void setY(int y) {
512        this.y = y;
513    }
514
515    public void setAlignment(Alignment alignment) {
516        this.alignment = alignment;
517    }
518
519    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
520        this.isPlayerProjectile = isPlayerProjectile;
521    }
522
523    public void setIsPenetrating(boolean isPenetrating) {
524        this.isPenetrating = isPenetrating;
525    }
526
527    public void setIsSecondPhase(boolean isSecondPhase) {
528        this.isSecondPhase = isSecondPhase;
529    }
530
531    public void setAbility2Started(boolean ability2Started) {
532        this.ability2Started = ability2Started;
533    }
534
535    public void setZeusSpeedX(int zeusSpeedX) {
536        this.ZeusSpeedX = zeusSpeedX;
537    }
538
539    public void setLightingImg(SpriteImage lightingImg) {
540        this.lightingImg = lightingImg;
541    }
542
543    public void setLightingImg2(SpriteImage lightingImg2) {
544        this.lightingImg2 = lightingImg2;
545    }
546
547    public void setResourceManager(ResourceManager resourceManager) {
548        this.resourceManager = resourceManager;
549    }
550
551    public void setWidth(int width) {
552        this.width = width;
553    }
554
555    public void setHeight(int height) {
556        this.height = height;
557    }
558
559    public void setX(int x) {
560        this.x = x;
561    }
562
563    public void setY(int y) {
564        this.y = y;
565    }
566
567    public void setAlignment(Alignment alignment) {
568        this.alignment = alignment;
569    }
570
571    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
572        this.isPlayerProjectile = isPlayerProjectile;
573    }
574
575    public void setIsPenetrating(boolean isPenetrating) {
576        this.isPenetrating = isPenetrating;
577    }
578
579    public void setIsSecondPhase(boolean isSecondPhase) {
580        this.isSecondPhase = isSecondPhase;
581    }
582
583    public void setAbility2Started(boolean ability2Started) {
584        this.ability2Started = ability2Started;
585    }
586
587    public void setZeusSpeedX(int zeusSpeedX) {
588        this.ZeusSpeedX = zeusSpeedX;
589    }
590
591    public void setLightingImg(SpriteImage lightingImg) {
592        this.lightingImg = lightingImg;
593    }
594
595    public void setLightingImg2(SpriteImage lightingImg2) {
596        this.lightingImg2 = lightingImg2;
597    }
598
599    public void setResourceManager(ResourceManager resourceManager) {
600        this.resourceManager = resourceManager;
601    }
602
603    public void setWidth(int width) {
604        this.width = width;
605    }
606
607    public void setHeight(int height) {
608        this.height = height;
609    }
610
611    public void setX(int x) {
612        this.x = x;
613    }
614
615    public void setY(int y) {
616        this.y = y;
617    }
618
619    public void setAlignment(Alignment alignment) {
620        this.alignment = alignment;
621    }
622
623    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
624        this.isPlayerProjectile = isPlayerProjectile;
625    }
626
627    public void setIsPenetrating(boolean isPenetrating) {
628        this.isPenetrating = isPenetrating;
629    }
630
631    public void setIsSecondPhase(boolean isSecondPhase) {
632        this.isSecondPhase = isSecondPhase;
633    }
634
635    public void setAbility2Started(boolean ability2Started) {
636        this.ability2Started = ability2Started;
637    }
638
639    public void setZeusSpeedX(int zeusSpeedX) {
640        this.ZeusSpeedX = zeusSpeedX;
641    }
642
643    public void setLightingImg(SpriteImage lightingImg) {
644        this.lightingImg = lightingImg;
645    }
646
647    public void setLightingImg2(SpriteImage lightingImg2) {
648        this.lightingImg2 = lightingImg2;
649    }
650
651    public void setResourceManager(ResourceManager resourceManager) {
652        this.resourceManager = resourceManager;
653    }
654
655    public void setWidth(int width) {
656        this.width = width;
657    }
658
659    public void setHeight(int height) {
660        this.height = height;
661    }
662
663    public void setX(int x) {
664        this.x = x;
665    }
666
667    public void setY(int y) {
668        this.y = y;
669    }
670
671    public void setAlignment(Alignment alignment) {
672        this.alignment = alignment;
673    }
674
675    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
676        this.isPlayerProjectile = isPlayerProjectile;
677    }
678
679    public void setIsPenetrating(boolean isPenetrating) {
680        this.isPenetrating = isPenetrating;
681    }
682
683    public void setIsSecondPhase(boolean isSecondPhase) {
684        this.isSecondPhase = isSecondPhase;
685    }
686
687    public void setAbility2Started(boolean ability2Started) {
688        this.ability2Started = ability2Started;
689    }
690
691    public void setZeusSpeedX(int zeusSpeedX) {
692        this.ZeusSpeedX = zeusSpeedX;
693    }
694
695    public void setLightingImg(SpriteImage lightingImg) {
696        this.lightingImg = lightingImg;
697    }
698
699    public void setLightingImg2(SpriteImage lightingImg2) {
700        this.lightingImg2 = lightingImg2;
701    }
702
703    public void setResourceManager(ResourceManager resourceManager) {
704        this.resourceManager = resourceManager;
705    }
706
707    public void setWidth(int width) {
708        this.width = width;
709    }
710
711    public void setHeight(int height) {
712        this.height = height;
713    }
714
715    public void setX(int x) {
716        this.x = x;
717    }
718
719    public void setY(int y) {
720        this.y = y;
721    }
722
723    public void setAlignment(Alignment alignment) {
724        this.alignment = alignment;
725    }
726
727    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
728        this.isPlayerProjectile = isPlayerProjectile;
729    }
730
731    public void setIsPenetrating(boolean isPenetrating) {
732        this.isPenetrating = isPenetrating;
733    }
734
735    public void setIsSecondPhase(boolean isSecondPhase) {
736        this.isSecondPhase = isSecondPhase;
737    }
738
739    public void setAbility2Started(boolean ability2Started) {
740        this.ability2Started = ability2Started;
741    }
742
743    public void setZeusSpeedX(int zeusSpeedX) {
744        this.ZeusSpeedX = zeusSpeedX;
745    }
746
747    public void setLightingImg(SpriteImage lightingImg) {
748        this.lightingImg = lightingImg;
749    }
750
751    public void setLightingImg2(SpriteImage lightingImg2) {
752        this.lightingImg2 = lightingImg2;
753    }
754
755    public void setResourceManager(ResourceManager resourceManager) {
756        this.resourceManager = resourceManager;
757    }
758
759    public void setWidth(int width) {
760        this.width = width;
761    }
762
763    public void setHeight(int height) {
764        this.height = height;
765    }
766
767    public void setX(int x) {
768        this.x = x;
769    }
770
771    public void setY(int y) {
772        this.y = y;
773    }
774
775    public void setAlignment(Alignment alignment) {
776        this.alignment = alignment;
777    }
778
779    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
780        this.isPlayerProjectile = isPlayerProjectile;
781    }
782
783    public void setIsPenetrating(boolean isPenetrating) {
784        this.isPenetrating = isPenetrating;
785    }
786
787    public void setIsSecondPhase(boolean isSecondPhase) {
788        this.isSecondPhase = isSecondPhase;
789    }
790
791    public void setAbility2Started(boolean ability2Started) {
792        this.ability2Started = ability2Started;
793    }
794
795    public void setZeusSpeedX(int zeusSpeedX) {
796        this.ZeusSpeedX = zeusSpeedX;
797    }
798
799    public void setLightingImg(SpriteImage lightingImg) {
800        this.lightingImg = lightingImg;
801    }
802
803    public void setLightingImg2(SpriteImage lightingImg2) {
804        this.lightingImg2 = lightingImg2;
805    }
806
807    public void setResourceManager(ResourceManager resourceManager) {
808        this.resourceManager = resourceManager;
809    }
810
811    public void setWidth(int width) {
812        this.width = width;
813    }
814
815    public void setHeight(int height) {
816        this.height = height;
817    }
818
819    public void setX(int x) {
820        this.x = x;
821    }
822
823    public void setY(int y) {
824        this.y = y;
825    }
826
827    public void setAlignment(Alignment alignment) {
828        this.alignment = alignment;
829    }
830
831    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
832        this.isPlayerProjectile = isPlayerProjectile;
833    }
834
835    public void setIsPenetrating(boolean isPenetrating) {
836        this.isPenetrating = isPenetrating;
837    }
838
839    public void setIsSecondPhase(boolean isSecondPhase) {
840        this.isSecondPhase = isSecondPhase;
841    }
842
843    public void setAbility2Started(boolean ability2Started) {
844        this.ability2Started = ability2Started;
845    }
846
847    public void setZeusSpeedX(int zeusSpeedX) {
848        this.ZeusSpeedX = zeusSpeedX;
849    }
850
851    public void setLightingImg(SpriteImage lightingImg) {
852        this.lightingImg = lightingImg;
853    }
854
855    public void setLightingImg2(SpriteImage lightingImg2) {
856        this.lightingImg2 = lightingImg2;
857    }
858
859    public void setResourceManager(ResourceManager resourceManager) {
860        this.resourceManager = resourceManager;
861    }
862
863    public void setWidth(int width) {
864        this.width = width;
865    }
866
867    public void setHeight(int height) {
868        this.height = height;
869    }
870
871    public void setX(int x) {
872        this.x = x;
873    }
874
875    public void setY(int y) {
876        this.y = y;
877    }
878
879    public void setAlignment(Alignment alignment) {
880        this.alignment = alignment;
881    }
882
883    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
884        this.isPlayerProjectile = isPlayerProjectile;
885    }
886
887    public void setIsPenetrating(boolean isPenetrating) {
888        this.isPenetrating = isPenetrating;
889    }
890
891    public void setIsSecondPhase(boolean isSecondPhase) {
892        this.isSecondPhase = isSecondPhase;
893    }
894
895    public void setAbility2Started(boolean ability2Started) {
896        this.ability2Started = ability2Started;
897    }
898
899    public void setZeusSpeedX(int zeusSpeedX) {
900        this.ZeusSpeedX = zeusSpeedX;
901    }
902
903    public void setLightingImg(SpriteImage lightingImg) {
904        this.lightingImg = lightingImg;
905    }
906
907    public void setLightingImg2(SpriteImage lightingImg2) {
908        this.lightingImg2 = lightingImg2;
909    }
910
911    public void setResourceManager(ResourceManager resourceManager) {
912        this.resourceManager = resourceManager;
913    }
914
915    public void setWidth(int width) {
916        this.width = width;
917    }
918
919    public void setHeight(int height) {
920        this.height = height;
921    }
922
923    public void setX(int x) {
924        this.x = x;
925    }
926
927    public void setY(int y) {
928        this.y = y;
929    }
930
931    public void setAlignment(Alignment alignment) {
932        this.alignment = alignment;
933    }
934
935    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
936        this.isPlayerProjectile = isPlayerProjectile;
937    }
938
939    public void setIsPenetrating(boolean isPenetrating) {
940        this.isPenetrating = isPenetrating;
941    }
942
943    public void setIsSecondPhase(boolean isSecondPhase) {
944        this.isSecondPhase = isSecondPhase;
945    }
946
947    public void setAbility2Started(boolean ability2Started) {
948        this.ability2Started = ability2Started;
949    }
950
951    public void setZeusSpeedX(int zeusSpeedX) {
952        this.ZeusSpeedX = zeusSpeedX;
953    }
954
955    public void setLightingImg(SpriteImage lightingImg) {
956        this.lightingImg = lightingImg;
957    }
958
959    public void setLightingImg2(SpriteImage lightingImg2) {
960        this.lightingImg2 = lightingImg2;
961    }
962
963    public void setResourceManager(ResourceManager resourceManager) {
964        this.resourceManager = resourceManager;
965    }
966
967    public void setWidth(int width) {
968        this.width = width;
969    }
970
971    public void setHeight(int height) {
972        this.height = height;
973    }
974
975    public void setX(int x) {
976        this.x = x;
977    }
978
979    public void setY(int y) {
980        this.y = y;
981    }
982
983    public void setAlignment(Alignment alignment) {
984        this.alignment = alignment;
985    }
986
987    public void setIsPlayerProjectile(boolean isPlayerProjectile) {
988        this.isPlayerProjectile = isPlayerProjectile;
989    }
990
991    public void setIsPenetrating(boolean isPenetrating) {
992        this.isPenetrating = isPenetrating;
993    }
994
995    public void setIsSecondPhase(boolean isSecondPhase) {
996        this.isSecondPhase = isSecondPhase;
997    }
998
999    public void setAbility2Started(boolean ability2Started) {
1000       this.ability2Started = ability2Started;
1001   }
1002
1003   public void setZeusSpeedX(int zeusSpeedX) {
1004       this.ZeusSpeedX = zeusSpeedX;
1005   }
1006
1007   public void setLightingImg(SpriteImage lightingImg) {
1008       this.lightingImg = lightingImg;
1009   }
1010
1011   public void setLightingImg2(SpriteImage lightingImg2) {
1012       this.lightingImg2 = lightingImg2;
1013   }
1014
1015   public void setResourceManager(ResourceManager resourceManager) {
1016       this.resourceManager = resourceManager;
1017   }
1018
1019   public void setWidth(int width) {
1020       this.width = width;
1021   }
1022
1023   public void setHeight(int height) {
1024       this.height = height;
1025   }
1026
1027   public void setX(int x) {
1028       this.x = x;
1029   }
1030
1031   public void setY(int y) {
1032       this.y = y;
1033   }
1034
1035   public void setAlignment(Alignment alignment) {
1036       this.alignment = alignment;
1037   }
1038
1039   public void setIsPlayerProjectile(boolean isPlayerProjectile) {
1040       this.isPlayerProjectile = isPlayerProjectile;
1041   }
1042
1043   public void setIsPenetrating(boolean isPenetrating) {
1044       this.isPenetrating = isPenetrating;
1045   }
1046
1047   public void setIsSecondPhase(boolean isSecondPhase) {
1048       this.isSecondPhase = isSecondPhase;
1049   }
1050
1051   public void setAbility2Started(boolean ability2Started) {
1052       this.ability2Started = ability2Started;
1053   }
1054
1055   public void setZeusSpeedX(int zeusSpeedX) {
1056       this.ZeusSpeedX = zeusSpeedX;
1057   }
1058
1059   public void setLightingImg(SpriteImage lightingImg) {
1060       this.lightingImg = lightingImg;
1061   }
1062
1063   public void setLightingImg2(SpriteImage lightingImg2) {
1064       this.lightingImg2 = lightingImg2;
1065   }
1066
1067   public void setResourceManager(ResourceManager resourceManager) {
1068       this.resourceManager = resourceManager;
1069   }
1070
1071   public void setWidth(int width) {
1072       this.width = width;
1073   }
1074
1075   public void setHeight(int height) {
1076       this.height = height;
1077   }
1078
1079   public void setX(int x) {
1080       this.x = x;
1081   }
1082
1083   public void setY(int y) {
1084       this.y = y;
1085   }
1086
1087   public void setAlignment(Alignment alignment) {
1088       this.alignment = alignment;
1089   }
1090
1091   public void setIsPlayerProjectile(boolean isPlayerProjectile) {
1092       this.isPlayerProjectile = isPlayerProjectile;
1093   }
1094
1095   public void setIsPenetrating(boolean isPenetrating) {
1096       this.isPenetrating = isPenetrating;
1097   }
1098
1099   public void setIsSecondPhase(boolean isSecondPhase) {
1100       this.isSecondPhase = isSecondPhase;
1101   }
1102
1103   public void setAbility2Started(boolean ability2Started) {
1104       this.ability2Started = ability2Started;
1105   }
1106
1107   public void setZeusSpeedX(int zeusSpeedX) {
1108       this.ZeusSpeedX = zeusSpeedX;
1109   }
1110
1111   public void setLightingImg(SpriteImage lightingImg) {
1112       this.lightingImg = lightingImg;
1113   }
1114
1115   public void setLightingImg2(SpriteImage lightingImg2) {
1116       this.lightingImg2 = lightingImg2;
1117   }
1118
1119   public void setResourceManager(ResourceManager resourceManager) {
1120       this.resourceManager = resourceManager;
1121   }
1122
1123   public void setWidth(int width) {
1124       this.width = width;
1125   }
1126
1127   public void setHeight(int height) {
1128       this.height = height;
1129   }
1130
1131   public void setX(int x) {
1132       this.x = x;
1133   }
1134
1135   public void setY(int y) {
1136       this.y = y;
1137   }
1138
1139   public void setAlignment(Alignment alignment) {
1140       this.alignment = alignment;
1141   }
1142
1143   public void setIsPlayerProjectile(boolean isPlayerProjectile) {
1144       this.isPlayerProjectile = isPlayerProjectile;
1145   }
1146
1147   public void setIsPenetrating(boolean isPenetrating) {
1148       this.isPenetrating = isPenetrating;
1149   }
1150
1151   public void setIsSecondPhase(boolean isSecondPhase) {
1152       this.isSecondPhase = isSecondPhase;
1153   }
1154
1155   public void setAbility2Started(boolean ability2Started) {
1156       this.ability2Started = ability2Started;
1157   }
1158
1159   public void setZeusSpeedX(int zeusSpeedX) {
1160       this.ZeusSpeedX = zeusSpeedX;
1161   }
1162
1163   public void setLightingImg(SpriteImage lightingImg) {
1164       this.lightingImg = lightingImg;
1165   }
1166
1167   public void setLightingImg2(SpriteImage lightingImg2) {
1168       this.lightingImg2 = lightingImg2;
1169   }
1170
1171   public void setResourceManager(ResourceManager resourceManager) {
1172       this.resourceManager = resourceManager;
1173   }
1174
1175   public void setWidth(int width) {
1176       this.width = width;
1177   }
1178
1179   public void setHeight(int height) {
1180       this.height = height;
1181   }
1182
1183   public void setX(int x) {
1184       this.x = x;
1185   }
1186
1187   public void setY(int y) {
1188       this.y = y;
1189   }
1190
1191   public void setAlignment(Alignment alignment) {
1192       this.alignment = alignment;
1193   }
1194
1195   public void setIsPlayerProjectile(boolean isPlayerProjectile) {
1196       this.isPlayerProjectile = isPlayerProjectile;
1197   }
1198
1199   public void setIsPenetrating(boolean isPenetrating) {
1200       this.isPenetrating = isPenetrating;
1201   }
1202
1203   public void setIsSecondPhase(boolean isSecondPhase) {
1204       this.isSecondPhase = isSecondPhase;
1205   }
1206
1207   public void setAbility2Started(boolean ability2Started) {
1208       this.ability2Started = ability2Started;
1209   }
1210
1211   public void setZeusSpeedX(int zeusSpeedX) {
1212       this.ZeusSpeedX = zeusSpeedX;
1213   }
1214
1215   public void setLightingImg(SpriteImage lightingImg) {
1216       this.lightingImg = lightingImg;
1217   }
1218
1219   public void setLightingImg2(SpriteImage lightingImg2) {
1220       this.lightingImg2 = lightingImg2;
1221   }
1222
1223   public void setResourceManager(ResourceManager resourceManager) {
1224       this.resourceManager = resourceManager;
1225   }
1226
1227   public void setWidth(int width) {
1228       this.width = width;
1229   }
1230
1231   public void setHeight(int height) {
1232       this.height = height;
1233   }
1234
1235   public void setX(int x) {
1236       this.x = x;
1237   }
1238
1239   public void setY(int y) {
1240       this.y = y;
1241   }
1242
1243   public void setAlignment(Alignment alignment) {
1244       this.alignment = alignment;
1245   }
1246
1247   public void setIsPlayerProjectile(boolean isPlayerProjectile) {
1248       this.isPlayerProjectile = isPlayerProjectile;
124
```

```

44         this.y = y + height;
45     }
46 }
47
48 @Override
49 public void move() {
50     y += vely;
51
52     if (y > GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT || y <
GameConstants.HUD_HEIGHT - height) {
53         isDead = true;
54     }
55 }
56
57 @Override
58 public void draw(Graphics g) {
59     if (image != null) {
60         g.drawImage(image, x, y, width, height, null);
61     } else {
62         g.setColor(Color.YELLOW);
63         g.fillOval(x, y, width, height);
64     }
65 }
66
67 @Override
68 public Shape getShape() {
69     return new Ellipse2D.Float(x, y, width, height);
70 }
71 }
```

Listing 18: 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 19: EnemySpawner.java

```

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

```

27     * @return true if it's time to spawn an enemy, false otherwise.
28 */
29 public boolean update() {
30     timer--;
31     if (timer <= 0) {
32         resetTimer();
33         return true;
34     }
35     return false;
36 }
37
38 private void resetTimer() {
39     // Calculate next spawn time: base +/- random variance
40     int var = (variance > 0) ? rand.nextInt(variance * 2 + 1) - variance :
41     0;
42     this.timer = baseInterval + var;
43     if (this.timer < 30) this.timer = 30; // Minimum safety limit (0.5 sec
44 }
45 /**
46 * Increases difficulty by reducing the spawn interval.
47 * @param multiplier Factor to multiply interval (e.g., 0.8 for 20% faster
48 ). .
49 */
50 public void increaseDifficulty(double multiplier) {
51     this.baseInterval = (int)(this.baseInterval * multiplier);
52     this.variance = (int)(this.variance * multiplier);
53     // Prevent it from becoming too fast (e.g., limit to 60 frames)
54     if (this.baseInterval < 60) this.baseInterval = 60;
55 }
56
57 public Class<? extends Minion> getEnemyType() {
58     return enemyType;
59 }
}

```

Listing 20: 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
15 public class Boulder extends Projectile {
16
17     private double preciseY;
18     private double velY;
19     private double gravity = GameConstants.Boulder_GRAVITY; // Accelerates
20     downwards
21
22     public Boulder(int x, int y) {
23         super(x, y,
24               GameConstants.Boulder_WIDTH,
25               GameConstants.Boulder_HEIGHT,
26               ResourceManager.boulderImg,
27               Alignment.ENEMY,
28               2,
29               GameConstants.Boulder_DAMAGE
30         );
31
32         this.preciseY = y;
33         this.velY = GameConstants.Boulder_INITIAL_SPEED; // Initial throw
34         speed
35     }
36
37     @Override
38     public void move() {
39         // Apply Gravity
40         velY += gravity;
41         preciseY += velY;
42         y = (int) preciseY;

```

```

39
40     // Despawn if off-screen
41     if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT) {
42         isDead = true;
43     }
44 }
45
46 @Override
47 public void draw(Graphics g) {
48     if (image != null) {
49         g.drawImage(image, x, y, width, height, null);
50     } else {
51         // Placeholder: Gray Rock
52         g.setColor(Color.GRAY);
53         g.fillOval(x, y, width, height);
54     }
55 }
56
57 @Override
58 public Shape getShape() {
59     // set hitbox to a circle
60     return new Ellipse2D.Float(x, y, width, height);
61 }
62 }
```

Listing 21: Cyclops.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 public class Cyclops extends Minion{
8
9     private double preciseY;
10    private double velY;
11    private int movementTimer;
12    private int attackTimer;
13    private Boulder myBoulder = null;
```

```

14     private boolean reachedMidScreen;
15     private boolean wingsClosed = false;
16
17     public Cyclops (int x, int y, GameModel model){
18         super(x, y,
19             GameConstants.CYCLOPS_WIDTH,
20             GameConstants.CYCLOPS_HEIGHT,
21             ResourceManager.cyclopsImg,
22             GameConstants.CYCLOPS_HP,
23             GameConstants.CYCLOPS_SCORE_POINTS,
24             model);
25         this.preciseY = y;
26         this.velY = GameConstants.CYCLOPS_YSPEED;
27         this.movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
28         this.attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
29         this.reachedMidScreen = false;
30     }
31
32     @Override
33     public void move() {
34         super.move();
35         if (movementTimer <= 0){
36             if (velY > 0){
37                 // Go up
38                 velY = GameConstants.CYCLOPS_YSPEED / -4;
39                 movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 / 2;
40                 wingsClosed = true; // STATE: closed wings
41             }
42             else {
43                 // go down
44                 wingsClosed = false; // STATE: opened wings
45
46                 if (!reachedMidScreen) {
47                     velY = GameConstants.CYCLOPS_YSPEED;
48                     movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
49                 } else {
50                     velY = GameConstants.CYCLOPS_YSPEED / 4;
51                     movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 /
52                         2;
53                 }
54             }
55         }
56     }

```

```

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 }
79
80 @Override
81 public void draw(Graphics g) {
82     BufferedImage imgToDraw;
83
84     if (image == ResourceManager.cyclopsImg){
85         imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg :
86         image;
87     } else {
88         imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg2 :
89         image;
90     }
91
92     if (image != null) {

```

```

89         g.drawImage(imgToDraw, x, y, width, height, null);
90     } else {
91         g.setColor(Color.RED);
92         g.fillRect(x, y, width, height);
93     }
94 }
95
96 private void throwBoulder(){
97     Boulder b = new Boulder (x + (width - GameConstants.Boulder_WIDTH)/2,
98     y + height);
99     this.myBoulder = b;
100    model.spawnEnemyProjectile(b);
101}

```

Listing 22: Feather.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5
6 public class Feather extends Projectile {
7
8     public Feather(int x, int y) {
9         // Alignment: ENEMY
10        // Power Level: 1 (Light) -> Clashes equally with Arrows
11        // Damage: Standard Feather Damage
12        super(x, y, GameConstants.FEATHER_WIDTH, GameConstants.FEATHER_HEIGHT,
13             ResourceManager.featherImg,
14             Alignment.ENEMY, 1, GameConstants.FEATHER_DAMAGE);
15    }
16
17    @Override
18    public void move() {
19        y += GameConstants.FEATHER_SPEED;
20        // Despawn logic
21        if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT) {
22            isDead = true;
23        }
24    }

```

```

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 23: 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) {

```

```

64         fireTimer--;
65     }
66     if (fireTimer <= 0) {
67         throwFeather();
68         resetFireTimer();
69     }
70 }
71
72 private void throwFeather() {
73     Feather f = new Feather (x + (width - GameConstants.FEATHER_WIDTH)/2,
74     y + height);
75     model.spawnEnemyProjectile(f);
76 }
77
78 public void resetFireTimer() {
79     int base = GameConstants.FEATHER_FIRE_INTERVAL;
80     int variance = GameConstants.FEATHER_FIRE_VARIANCE;
81
82     int randomVariation = (int)(Math.random() * (variance * 2)) - variance
83     ;
84
85     this.fireTimer = base + randomVariation;
86 }
87
88 @Override
89 public void draw(Graphics g) {
90     BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
91     harpyHitImg : image;
92
93     if (image != null) {
94         if (velX < 0) {
95             g.drawImage(imgToDraw, x, y, width, height, null);
96         } else {
97             g.drawImage(imgToDraw, x + width, y, -width, height, null);
98         }
99     } else {
100        g.setColor(Color.RED);
101        g.fillRect(x, y, width, height);
102    }
103 }

```

101 }

Listing 24: 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     }
15 }
```

Listing 25: 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);
45
46         // Switch drawing based on game state
47         GameState state = model.getState();
48
49         if (state == GameState.TITLE) {
50             drawTitleScreen(g);
51         } else if (state == GameState.PLAYING || state == GameState.PAUSED ||
52         state == GameState.MESSAGE) {
53             drawGameScreen(g);
54             if (state == GameState.PAUSED){
55                 drawPauseScreen(g);
56             }
57             else if (state == GameState.MESSAGE) {
58                 drawMessageScreen(g);
59             }
60         }
61     }

```

```

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         }
97     }
98 }
```

```

96         obj.draw(g);
97     }
98
99     // 3. Draw Top HUD (Score, Stage, Lives)
100    drawTopHUD(g);
101
102    // 4. Draw Bottom HUD (Ability Slots) -> NEW!
103    drawBottomHUD(g);
104}
105
106 // Helper method to draw the Top HUD
107 private void drawTopHUD(Graphics g) {
108     // Draw dark background bar
109     g.setColor(new Color(50, 50, 80));
110     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.HUD_HEIGHT);
111 ;
112
113     // Draw white separator line
114     g.setColor(Color.WHITE);
115     g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH,
116     GameConstants.HUD_HEIGHT);
117
118     // Font settings
119     setPixelFont(g, 18f);
120     int textY = 35;
121
122     // A. Score
123     g.drawString("SCORE:" + model.getScore(), 10, textY);
124
125     // B. Stage (Centered)
126     String stageText = model.getStageText();
127     int stageX = (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
128     stringWidth(stageText)) / 2;
129     g.drawString(stageText, stageX, textY);
130
131     // C. Hearts / Lives (Right aligned)
132     int maxLives = GameConstants.PLAYER_MAX_LIVES;
133     int currentLives = model.getLives();
134     int heartSize = 32;
135     int spacing = 8;

```

```

133     int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (heartSize
+ spacing));
134     int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
135
136     for (int i = 0; i < maxLives; i++) {
137         // Determine which icon to draw (Full or Empty)
138         BufferedImage icon = (i < currentLives) ? ResourceManager.
139         heartFullImg : ResourceManager.heartEmptyImg;
140
141         if (icon != null) {
142             g.drawImage(icon, startX + (i * (heartSize + spacing)), heartY
143             , heartSize, heartSize, null);
144         } else {
145             // Fallback drawing if images are missing
146             g.setColor(i < currentLives ? Color.RED : Color.GRAY);
147             g.fillOval(startX + (i * (heartSize + spacing)), heartY,
148             heartSize, heartSize);
149         }
150     }
151
152     // Helper method to draw the Bottom HUD (Ability Slots)
153     // Helper method to draw the Bottom HUD (Ability Slots)
154     private void drawBottomHUD(Graphics g) {
155         // Calculate start Y position (below the game field)
156         int startY = GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT;
157         int height = GameConstants.BOTTOM_HUD_HEIGHT;
158
159         // 1. Background Bar
160         g.setColor(new Color(50, 50, 80));
161         g.fillRect(0, startY, GameConstants.WINDOW_WIDTH, height);
162
163         // 2. Separator Line
164         g.setColor(Color.WHITE);
165         g.drawLine(0, startY, GameConstants.WINDOW_WIDTH, startY);
166
167         // 3. Draw Slots
168         int slotSize = 60;

```

```

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
194             // is set correctly in Constants!
195
196             if (timer > 0) {
197                 // Calculate percentage of time remaining (0.0 to 1.0)
198                 float ratio = (float) timer / maxTime;
199
200                 // Calculate height of the dark overlay based on the ratio
201                 // If ratio is 1.0 (just used), height is full (60).
202                 // If ratio is 0.5, height is half (30), covering the top
203                 half.
204
205                 // This creates the effect of the color "filling up from
206                 // bottom".
207                 int overlayHeight = (int) (slotSize * ratio);
208
209                 // Set color to semi-transparent black

```

```

204         g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (transparency)
205
206         // Draw the overlay from the top of the slot downwards
207         g.fillRect(x, slotY, slotSize, overlayHeight);
208
209         // Optional: Draw the text timer on top if you want
210         g.setColor(Color.WHITE);
211         String keyNum = String.valueOf(timer/60 + 1);
212         int numWidth = g.getFontMetrics().stringWidth(keyNum);
213         g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY
214             + 37);
215     }
216
217     // Check if this is the first slot (Index 0) AND if Ability 1 is
218     // unlocked
219     if (i == 1 && model.isAbilityUnclocked(2)) {
220
221         // 1. Draw the Icon (The Sun)
222         if (ResourceManager.lightingImg != null) {
223             g.drawImage(ResourceManager.lightingImg, x, slotY,
224                 slotSize, slotSize, null);
225         }
226
227         // 2. Draw Cooldown Overlay (The fading effect)
228         int timer = model.getAbilityNthTimer(2);
229         int maxTime = GameConstants.ABILITY2TIMER; // Make sure this
230         // is set correctly in Constants!
231
232         if (timer > 0) {
233             // Calculate percentage of time remaining (0.0 to 1.0)
234             float ratio = (float) timer / maxTime;
235
236             // Calculate height of the dark overlay based on the ratio
237             // If ratio is 1.0 (just used), height is full (60).
238             // If ratio is 0.5, height is half (30), covering the top
239             // half.
240             // This creates the effect of the color "filling up from
241             // bottom".

```

```

237         int overlayHeight = (int) (slotSize * ratio);
238
239         // Set color to semi-transparent black
240         g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (
transparency)
241
242         // Draw the overlay from the top of the slot downwards
243         g.fillRect(x, slotY, slotSize, overlayHeight);
244
245         // Optional: Draw the text timer on top if you want
246         g.setColor(Color.WHITE);
247         String keyNum = String.valueOf(timer/60 + 1);
248         int numWidth = g.getFontMetrics().stringWidth(keyNum);
249         g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY
+ 37);
250     }
251 }
252 // -----
253
254 // B. Draw Slot Border
255 g.setColor(Color.GRAY);
256 g.drawRect(x, slotY, slotSize, slotSize);
257
258 // C. Draw Key Number (1, 2, 3)
259 g.setColor(Color.WHITE);
260 String keyNum = String.valueOf(i + 1);
261 int numWidth = g.getFontMetrics().stringWidth(keyNum);
262 g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY +
slotSize + 25);
263 }
264 }
265
266 // Draw Title Screen
267 private void drawTitleScreen(Graphics g) {
268     g.setColor(Color.WHITE);
269
270     // Title: Big Pixel Font
271     setPixelFont(g, 28f);
272     String title = "GLADIATOR GAME"; // Cambia il nome se vuoi
273     int titleWidth = g.getFontMetrics().stringWidth(title);

```

```

274     g.drawString(title, (GameConstants.WINDOW_WIDTH - titleWidth)/2, 250);
275
276     // Subtitle: Smaller
277     setPixelFont(g, 15f);
278     String msg = "Press SPACE to Start";
279     int msgWidth = g.getFontMetrics().stringWidth(msg);
280     g.drawString(msg, (GameConstants.WINDOW_WIDTH - msgWidth)/2, 350);
281 }
282
283 private void drawPauseScreen(Graphics g) {
284     // 1. Semi-transparent black overlay
285     g.setColor(new Color(0, 0, 0, 150)); // 150 = Alpha (Transparency)
286     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
287 WINDOW_HEIGHT);
288
289     // 2. "PAUSE" Text
290     g.setColor(Color.WHITE);
291     setPixelFont(g, 40f); // Large font
292     String pauseText = "PAUSE";
293     int pauseWidth = g.getFontMetrics().stringWidth(pauseText);
294     // Center text
295     g.drawString(pauseText, (GameConstants.WINDOW_WIDTH - pauseWidth) / 2,
296 GameConstants.WINDOW_HEIGHT / 2 - 100);
297
298     // 3. Instruction Text
299     setPixelFont(g, 20f); // Smaller font
300     String resumeText = "Press [P] to Resume";
301     int resumeWidth = g.getFontMetrics().stringWidth(resumeText);
302     g.drawString(resumeText, (GameConstants.WINDOW_WIDTH - resumeWidth) /
303 2, GameConstants.WINDOW_HEIGHT / 2 - 50);
304 }
305
306 private void drawMessageScreen(Graphics g) {
307     // 1. Semi-transparent black background for the whole screen (dimming)
308     g.setColor(new Color(0, 0, 0, 100));
309     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.

```

```

310     int boxHeight = 400;
311     int boxX = (GameConstants.WINDOW_WIDTH - boxWidth) / 2;
312     int boxY = (GameConstants.WINDOW_HEIGHT - boxHeight) / 2;
313
314     // 3. Draw the Box Background (Dark Blue)
315     g.setColor(new Color(20, 20, 80));
316     g.fillRect(boxX, boxY, boxWidth, boxHeight);
317
318     // 4. Draw the Box Border (White)
319     g.setColor(Color.WHITE);
320     Graphics2D g2 = (Graphics2D) g;
321     g2.setStroke(new BasicStroke(4)); // Thicker border
322     g2.drawRect(boxX, boxY, boxWidth, boxHeight);
323
324     // 5. Draw the Text
325     String[] lines = model.getCurrentMessageLines();
326     if (lines != null) {
327         setPixelFont(g, 20f); // Size for text
328         g.setColor(Color.WHITE);
329
330         int lineHeight = 30;
331         // Calculate starting Y to center the block of text vertically
332         int totalTextHeight = lines.length * lineHeight;
333         int startTextY = boxY + (boxHeight - totalTextHeight) / 2 + 10; // +10 adjustment
334
335         for (int i = 0; i < lines.length; i++) {
336             String line = lines[i];
337             // Center align each line horizontally
338             int lineWidth = g.getFontMetrics().stringWidth(line);
339             int lineX = (GameConstants.WINDOW_WIDTH - lineWidth) / 2;
340
341             g.drawString(line, lineX, startTextY + (i * lineHeight));
342         }
343     }
344
345     // 6. Draw "Press Space" prompt at the bottom of the box
346     setPixelFont(g, 14f);
347     g.setColor(Color.YELLOW);
348     String prompt = "PRESS [SPACE] TO CONTINUE";

```

```

349     int promptWidth = g.getFontMetrics().stringWidth(prompt);
350     g.drawString(prompt, (GameConstants.WINDOW_WIDTH - promptWidth) / 2,
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);
375
376     // ... Time e Quit/Continue (usa la stessa logica per centrare) ...
377     String cont = "[C] Continue";
378     String quit = "[Q] Quit";
379
380     g.drawString(cont, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
381     stringWidth(cont))/2, 400);
382     g.drawString(quit, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
383     stringWidth(quit))/2, 440);
384 }
385
386 // Update Player Velocity based on key states
387 private void updatePlayerVelocity() {
388     Player p = model.getPlayer();

```

```

384
385     int vx = 0;
386     int vy = 0;
387
388     if (leftPressed && !rightPressed) vx = -1;
389     if (rightPressed && !leftPressed) vx = 1;
390     if (upPressed && !downPressed) vy = -1;
391     if (downPressed && !upPressed) vy = 1;
392
393     p.setVelX(vx);
394     p.setVelY(vy);
395 }
396
397 // Reset keys when restarting game
398 private void resetKeyState() {
399     leftPressed = false;
400     rightPressed = false;
401     upPressed = false;
402     downPressed = false;
403
404     Player p = model.getPlayer();
405     if (p != null) {
406         p.setVelX(0);
407         p.setVelY(0);
408     }
409 }
410
411 @Override
412 public void keyPressed(KeyEvent e) {
413     int key = e.getKeyCode();
414     GameState state = model.getState();
415
416     // Title Screen Input
417     if (state == GameState.TITLE) {
418         if (key == KeyEvent.VK_SPACE) {
419             model.initGame(); // Start Game
420             startTime = System.currentTimeMillis();
421             endTime = 0;
422         }
423     }

```

```

424
425     // Playing State Input
426     else if (state == GameState.PLAYING) {
427         if (key == KeyEvent.VK_LEFT) leftPressed = true;
428         if (key == KeyEvent.VK_RIGHT) rightPressed = true;
429         if (key == KeyEvent.VK_UP) upPressed = true;
430         if (key == KeyEvent.VK_DOWN) downPressed = true;
431
432         if (key == KeyEvent.VK_SPACE) {
433             model.setFiring(true);
434         }
435
436         if (key == KeyEvent.VK_P) {
437             model.setState(GameState.PAUSED);
438             resetKeyState();
439             System.out.println("Game Paused");
440         }
441
442         updatePlayerVelocity();
443
444         // Placeholder for Abilities
445         // ABILITY 1
446         if(model.getCurrentLevelIndex() > 3 && key == KeyEvent.VK_1) {
447             model.ability1();
448         }
449
450         // ABILITY 2
451         if (model.getCurrentLevelIndex() > 7 && key == KeyEvent.VK_2) {
452             model.ability2();
453         }
454
455         // ABILITY 3
456         if (key == KeyEvent.VK_3) System.out.println("Ability 3 pressed");
457     }
458
459     else if (state == GameState.PAUSED) {
460         if (key == KeyEvent.VK_P) {
461             model.setState(GameState.PLAYING);
462             resetKeyState();
463             System.out.println("Game Resumed");

```

```

464         }
465     }
466
467     else if (state == GameState.MESSAGE) {
468         if (key == KeyEvent.VK_SPACE) {
469             model.resumeGame(); // Go back to Playing
470             resetKeyState();
471         }
472     }
473
474     // Game Over State Input
475     else if (state == GameState.GAMEOVER) {
476         if (key == KeyEvent.VK_C) {
477             model.initGame(); // Retry
478             resetKeyState();
479             startTime = System.currentTimeMillis();
480             endTime = 0;
481         } else if (key == KeyEvent.VK_Q) {
482             System.exit(0); // Quit App
483         }
484     }
485 }
486
487 @Override
488 public void keyReleased(KeyEvent e) {
489     int key = e.getKeyCode();
490     if (key == KeyEvent.VK_LEFT) leftPressed = false;
491     if (key == KeyEvent.VK_RIGHT) rightPressed = false;
492     if (key == KeyEvent.VK_UP) upPressed = false;
493     if (key == KeyEvent.VK_DOWN) downPressed = false;
494
495     if (key == KeyEvent.VK_SPACE) {
496         model.setFiring(false);
497     }
498     if (model.getState() == GameState.PLAYING) {
499         updatePlayerVelocity();
500     }
501 }
502
503 @Override

```

```
504     public void keyTyped(KeyEvent e) {}  
505 }
```

Listing 26: 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     // BOSSSES
99     //*****
100
101    // APOLLO
102    apolloImg = loadTexture("res/Apollo.png");
103    apolloImg2 = loadTexture("res/ApolloRed.png");
104    apolloHitImg = createWhiteSilhouette(apolloImg);
105    // APOLLO's SUN
106    sunImg = loadTexture("res/sun.png");
107    sunImg2 = loadTexture("res/sunRed.png");
108    // ZEUS
109    zeusImg = loadTexture("res/Zeus.png");
110    zeusImg2 = loadTexture("res/ZeusAngry.png");
111    zeusHitImg = createWhiteSilhouette(zeusImg);
112    zeusHitImg2 = createWhiteSilhouette(zeusImg2);
113    // ZEUS's LIGHTNING
114    lightingImg = loadTexture("res/lighting.png");
115    lightingImg2 = loadTexture("res/lightingAngry.png");

```

```

116
117     //*****
118     // HUD
119     //*****
120
121     // STAGES
122     stage1Img = loadTexture("res/stage1.png");
123     stage2Img = loadTexture("res/stage2.png");
124     stage3Img = loadTexture("res/stage3.png");
125     // HEART
126     heartFullImg = loadTexture("res/heart.png");
127     heartEmptyImg = createBlackSilhouette(heartFullImg);
128
129     // --- LOAD CUSTOM FONT ---
130     try {
131         // Load the font file from the res folder
132         InputStream is = ResourceManager.class.getClassLoader().
133             getResourceAsStream("res/PixelFont.ttf");
134
135         if (is != null) {
136             // Create the font object (default size is 1pt)
137             pixelFont = Font.createFont(Font.TRUETYPE_FONT, is);
138             System.out.println("Pixel Font loaded successfully!");
139         } else {
140             System.err.println("Error: PixelFont.ttf not found. Using
141 default font.");
142             pixelFont = new Font("Arial", Font.BOLD, 20); // Fallback
143         }
144     } catch (FontFormatException | IOException e) {
145         e.printStackTrace();
146         pixelFont = new Font("Arial", Font.BOLD, 20); // Fallback
147     }
148
149     System.out.println("All Resources loaded successfully!");
150 } catch (IOException e) {
151     System.err.println("Error: Could not load images.");
152     e.printStackTrace();
153 }
```

```

154 // 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
180     );
181
182     // すべてのピクセルを走査する
183     for (int x = 0; x < original.getWidth(); x++) {
184         for (int y = 0; y < original.getHeight(); y++) {
185             int p = original.getRGB(x, y);
186
187             // アルファ値（透明度）を取得
188             int a = (p >> 24) & 0xff;
189
190             // 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
191             if (a > 0) {
192                 // ARGB: アルファ値 + R(255) + G(255) + B(255)
193                 int whiteColor = (a << 24) | (255 << 16) | (255 << 8) |

```

```

255;
192         whiteImg.setRGB(x, y, whiteColor);
193     }
194 }
195 }
196 return whiteImg;
197 }

198
199 private static BufferedImage createBlackSilhouette(BufferedImage original)
200 {
201     BufferedImage blackImg = new BufferedImage(original.getWidth(),
202 original.getHeight(), BufferedImage.TYPE_INT_ARGB);
203     for (int x = 0; x < original.getWidth(); x++) {
204         for (int y = 0; y < original.getHeight(); y++) {
205             int p = original.getRGB(x, y);
206             int a = (p >> 24) & 0xff; // Get Alpha
207
208             // If the pixel is not transparent, make it BLACK
209             if (a > 0) {
210                 // ARGB: Alpha + R(0) + G(0) + B(0)
211                 int blackColor = (a << 24) | (0 << 16) | (0 << 8) | 0;
212                 blackImg.setRGB(x, y, blackColor);
213             }
214         }
215     }
216 }

```

(文責：佐々木)