

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

### 3.3 Puller 担当

## 4 実行例

## 5 考察

## 6 感想

## 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。

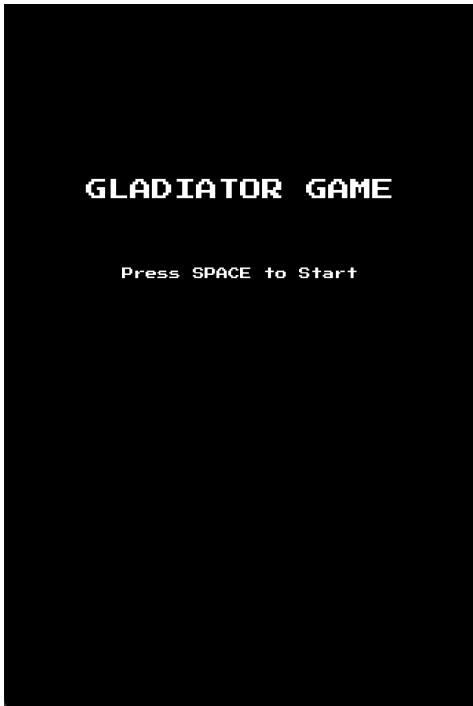


図 2: タイトル画面



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 1: 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}
```

```

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 2: GameModel.java

```

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

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

```

67     arrowDamage = GameConstants.ARROW_DAMAGE;
68     arrowInterval = GameConstants.ARROW_INTERVAL;
69     state = GameState.PLAYING;
70     score = 0; //スコアをリセット
71
72     // 追加ライフ初期化:
73     lives = GameConstants.PLAYER_MAX_LIVES;
74     damageTimer = 0;
75
76     //Score progression resets
77     this.currentLevelIndex = 0;
78     this.nextTargetScore = GameConstants.LEVEL_MILESTONES[
79     currentLevelIndex];
80     isBossActive = false;
81
82     // STAGE 1 SETUP: Add Harpy Spawner
83     activeSpawners.add(new EnemySpawner(Harpy.class, GameConstants.
84     HARPY_SPAWN_INTERVAL, GameConstants.HARPY_SPAWN_VARIANCE));
85     this.currentStage = 1;
86
87     String tutorial = "WELCOME GLADIATOR!\n\n" +
88         "Controls:\n" +
89         "[KEY-ARROWS] Move\n" +
90         "[SPACE] Shoot\n" +
91         "[P] Pause\n\n" +
92         "Defeat enemies\n" +
93         "and survive.\n" +
94         "Good Luck!";
95     showMessage(tutorial);
96
97 }
98
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 * Helper method to instantiate the correct enemy based on Class type.
183 */

```

```

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.
HARPY_WIDTH); // Simple random X
189             y = GameConstants.HUD_HEIGHT - GameConstants.HARPY_HEIGHT; // Start at top
190             Harpy h = new Harpy(x, y, this);
191             newObjectsBuffer.add(h);
192         }
193         else if (type == Cyclops.class) {
194             x = rand.nextInt(GameConstants.WINDOW_WIDTH - GameConstants.
CYCLOPS_WIDTH); // Simple random X
195             y = GameConstants.HUD_HEIGHT - GameConstants.CYCLOPS_HEIGHT;
196             Cyclops g = new Cyclops(x, y, this); // Pass 'this' (GameModel)
197             newObjectsBuffer.add(g);
198         }
199     }
200
201 /**
202 * Allows enemies (Minions/Bosses) to add projectiles to the game.
203 * Using a specific method is safer than exposing the whole list.
204 */
205 public void spawnEnemyProjectile(Projectile p) {
206     if (p != null) {
207         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ったら return
216         if (currentLevelIndex >= GameConstants.LEVEL_MILESTONES.length) return
;
217
218         if (score >= nextTargetScore){
219             //apply the effect we decided in the applyLevelEffects function
220             applyLevelEffects(currentLevelIndex);

```

```

221         currentLevelIndex++;
222         System.out.println("The current level index is: " +
223             currentLevelIndex);
224         nextTargetScore = GameConstants.LEVEL_MILESTONES[currentLevelIndex
225     ];
226     }
227 }
228
229 private void applyLevelEffects(int levelIndex) {
230     switch (levelIndex) {
231         case 0: //START OF THE GAME
232             System.out.println("Start of the Game");
233             break;
234
235         case 1:
236             System.out.println("Difficulty UP!");
237             for (EnemySpawner s : activeSpawners) s.increaseDifficulty
238             (0.8);
239             break;
240
241         case 2:
242             System.out.println("Difficulty UP!");
243             for (EnemySpawner s : activeSpawners) s.increaseDifficulty
244             (0.8);
245             break;
246
247         case 3:
248             showMessage("WARNING!\n\nBOSS DETECTED:\nAPOLLO\n\nPrepare for
249             battle!");
250             isBossActive = true;
251             clearEverything();
252             spawnApollo();
253             healPlayer();
254             break;
255         case 4:
256             showMessage("STAGE 1 CLEARED!\n\nEntering the Heavens...\n\n
257             nArrow Damage doubled!\n\n Press [1] to use\nABILITY 1:\nAPOLLO'S SUN");
258             if (background != null) {
259                 clearEverything();
260                 healPlayer();

```

```

255         background.setImage(ResourceManager.stage2Img);
256         background.setSpeed(GameConstants.SCREEN_SPEED);
257         ability1Timer = 0;
258         this.currentStage = 2;
259
260         // DOUBLE ARROW DAMAGE
261         arrowDamage*=2;
262
263         // Clear old spawners (remove Stage 1 config)
264         activeSpawners.clear();
265         // Harpy Spawner
266         activeSpawners.add(new EnemySpawner(Harpy.class, 100, 50))
267 ;
268         // Cyclops Spawner
269         activeSpawners.add(new EnemySpawner(Cyclops.class,
270                         GameConstants.CYCLOPS_SPAWN_INTERVAL,
271                         GameConstants.CYCLOPS_SPAWN_VARIANCE));
272
273     }
274     break;
275 case 5:
276     System.out.println("Difficulty UP!");
277     for (EnemySpawner s : activeSpawners) s.increaseDifficulty
278 (0.9);
279     break;
280 case 6:
281     System.out.println("Difficulty UP!");
282     for (EnemySpawner s : activeSpawners) s.increaseDifficulty
283 (0.9);
284     break;
285 case 7:
286     showMessage("WARNING!\n\nBOSS DETECTED:\nZEUS\n\nPrepare for
287 battle!");
288     isBossActive = true;
289     clearEverything();
290     spawnZeus();
291     healPlayer();
292     break;
293 case 8:
294     showMessage("STAGE 2 CLEARED!\n\nEntering the INFERNO...\n"

```

```

nShooting speed increased!\n\nPress [2] to use\nABILITY 2:\nZEUS'S
LIGHTING");
291     if (background != null) {
292         clearEverything();
293         background.setImage(ResourceManager.stage3Img);
294         background.setSpeed(0);
295         ability2Timer = 0;
296         arrowInterval = GameConstants.ARROW_INTERVAL2;
297         this.currentStage = 3;
298         activeSpawners.clear();
299         activeSpawners.add(new EnemySpawner(Harpy.class, 100, 50))
300     ;
301         // Cyclops Spawner
302         activeSpawners.add(new EnemySpawner(Cyclops.class,
303             GameConstants.CYCLOPS_SPAWN_INTERVAL,
304             GameConstants.CYCLOPS_SPAWN_VARIANCE));
305     }
306     break;
307 case 9:
308 }
309 }
310
311 public void showMessage(String text) {
312     // Split the text by newline character to handle multiple lines
313     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
Projectile);

```

```

327     newObjectsBuffer.removeIf(obj -> obj instanceof HostileEntity || obj
328         instanceof Projectile);
329
330     // プレイヤーが撃つ（から呼ばれる）Controller
331     public void playerShoot() {
332         if (!isGameOver) {
333             // プレイヤーの中央上から発射
334             Arrow a = new Arrow(player.getX() + (GameConstants.PLAYER_WIDTH-
335             GameConstants.ARROW_WIDTH)/2, player.getY() - GameConstants.ARROW_HEIGHT,
336             arrowDamage);
337             newObjectsBuffer.add(a);
338         }
339     }
340
341     public void ability1(){
342         if (ability1Timer > 0) return;
343         ability1Timer = GameConstants.ABILITY1TIMER;
344         Sun sun = new Sun(
345             player.getX() + GameConstants.PLAYER_WIDTH / 2,
346             player.getY(),
347             0, false, true
348         );           newObjectsBuffer.add(sun);
349         System.out.println("Player shoots sun");
350     }
351
352     public void ability2(){
353         if (ability2Timer > 0) return;
354         ability2Timer = GameConstants.ABILITY2TIMER;
355         Lighting l = new Lighting(
356             player.getX() + GameConstants.PLAYER_WIDTH / 2,
357             player.getY(),
358             0, false, true, false
359         );           newObjectsBuffer.add(l);
360         System.out.println("Player shoots Lighting");
361     }
362
363     private void spawnApollo() {
364         Apollo apollo = new Apollo(this);
365         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
399     if (!s1.getBounds2D().intersects(s2.getBounds2D())) {
400         return false;
401     }
402
403     // 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 }
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;

```

```

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     }
469
470     return;
471
472     // --- CASE 2: PROJECTILE vs LIVING ENTITY (Player or Enemy) ---
473     Projectile proj = null;
474     GameObject entity = null;
475
476     // Identify which is which
477     if (a instanceof Projectile) { proj = (Projectile) a; entity = b; }
478     else if (b instanceof Projectile) { proj = (Projectile) b; entity = a;
479 }
480
481     if (proj != null) {
482         // 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     return;
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 3: GameConstants.java

```
1 package model;
2
3 public final class GameConstants {
4     // 画面のサイズ
5     public static final int WINDOW_WIDTH = 600;
6
7     // HEIGHTS
8     public static final int HUD_HEIGHT = 50;
9     public static final int FIELD_HEIGHT = 800;
10    public static final int BOTTOM_HUD_HEIGHT = 100;
11    public static final int WINDOW_HEIGHT = HUD_HEIGHT + FIELD_HEIGHT +
12        BOTTOM_HUD_HEIGHT;
13
14    // ゲームの設定
15    public static final int FPS = 60;
16
17    // 背景の設定
18    public static final double SCREEN_SPEED = 1.0;
19
20    // の設定 PLAYER
21    public static final int PLAYER_WIDTH = 90;
22    public static final int PLAYER_HEIGHT = PLAYER_WIDTH*923/721; // set to
the image height width ration
23    public static final int PLAYER_SPEED = 8;
24    public static final int PLAYER_MAX_LIVES = 3;
25
26    // ABILITY TIMERS
27    public static final int ABILITY1TIMER = FPS * 15; // 15 seconds
28    public static final int ABILITY2TIMER = FPS * 8; // 10 seconds
29    public static final int ABILITY3TIMER = FPS * 10; // 10 seconds
30
31    // の設定 ARROW
32    public static final int ARROW_WIDTH = 10;
33    public static final int ARROW_HEIGHT = 70; // set to the image height
width ration
34    public static final int ARROW_SPEED = 30;
35    public static final int ARROW_INTERVAL = 20;
36    public static final int ARROW_INTERVAL2 = 15;
37    public static final int ARROW_DAMAGE = 1;
```

```
37
38 // ****
39 // の設定 MINIONS
40 // ****
41
42 // の設定 HARPY
43 public static final int HARPY_WIDTH = 100;
44 public static final int HARPY_HEIGHT = HARPY_WIDTH *1911/1708; // set to
the image height width ration
45 public static final int HARPY_XSPEED = 4;
46 public static final int HARPY_YSPEED = 2;
47 public static final int HARPY_HP = 2;
48 public static final int HARPY_SCORE_POINTS = 10;
49 public static final int HARPY_SPAWN_INTERVAL = 120;
50 public static final int HARPY_SPAWN_VARIANCE = HARPY_SPAWN_INTERVAL / 2;
51
52 // の設定 FEATHER
53 public static final int FEATHER_WIDTH = 12;
54 public static final int FEATHER_HEIGHT = FEATHER_WIDTH * 1698/378; // set
to the image height width ration
55 public static final int FEATHER_SPEED = 7;
56 public static final int FEATHER_FIRE_INTERVAL = 90;
57 public static final int FEATHER_FIRE_VARIANCE = FEATHER_FIRE_INTERVAL / 2;
58 public static final int FEATHER_DAMAGE = 1;
59
60 // の設定 CYCLOPS
61 public static final int CYCLOPS_WIDTH = 150;
62 public static final int CYCLOPS_HEIGHT = CYCLOPS_WIDTH;
63 public static final double CYCLOPS_YSPEED = 2;
64 public static final int CYCLOPS_HP = 10;
65 public static final int CYCLOPS_SCORE_POINTS = 50;
66 public static final int CYCLOPS_SPAWN_INTERVAL = 400;
67 public static final int CYCLOPS_SPAWN_VARIANCE = CYCLOPS_SPAWN_INTERVAL /
2;
68 public static final int CYCLOPS_MOVEMENT_TIMER = 20;
69 public static final int CYCLOPS_ATTACK_TIMER = 60;
70
71 // の設定 BOULDER
72 public static final int BOULDER_WIDTH = 100;
73 public static final int BOULDER_HEIGHT = BOULDER_WIDTH;
```

```
74     public static final double BOULDER_INITIAL_SPEED = 0;
75     public static final double BOULDER_GRAVITY = 0.3;
76     public static final int BOULDER_DAMAGE = 5;
77
78     // ****
79     // の設定 BOSS
80     // ****
81
82     // の設定 APOLLO
83     public static final int APOLLO_WIDTH = 200;
84     public static final int APOLLO_HEIGHT = APOLLO_WIDTH * 1556 / 2463;
85     public static final int APOLLO_SPEED1 = 4;
86     public static final int APOLLO_SPEED2 = APOLLO_SPEED1 * 3 / 2;
87     public static final int APOLLO_HP = 2; //50
88     public static final int APOLLO_SCORE_POINTS = 1000;
89
90     // の設定 SUN
91     public static final int SUN_WIDTH = 150;
92     public static final int SUN_HEIGHT = SUN_WIDTH;
93     public static final double SUN_SPEED1 = 6;
94     public static final double SUN_SPEED2 = SUN_SPEED1 * 3 / 2;
95     public static final int SUN_DAMAGE = 1;
96     public static final int SUN_HP = 20;
97
98     // の設定 ZEUS
99     public static final int ZEUS_SPEED = 4;
100    public static final int ZEUS_SPEED2 = 8;
101    public static final int ZEUS_WIDTH = 150;
102    public static final int ZEUS_HEIGHT = 150;
103    public static final int ZEUS_HP = 10; // 100
104    public static final int ZEUS_SCORE_POINTS = 1500;
105    public static final int ZEUS_SHOOT_TIMER = 60;
106    public static final int ZEUS_SHOOT_TIMER2 = 40;
107    public static final int ZEUS_ABILITY1_PAUSE = 40;
108    public static final int ZEUS_ABILITY1_PAUSE2 = 25;
109    public static final int ZEUS_ABILITY2_TIMER = FPS * 6; // 10 seconds
110
111    // の設定 LIGHTING
112    public static final int LIGHTING_WIDTH = 25;
113    public static final int LIGHTING_HEIGHT = 150;
```

```
114     public static final int LIGHTING_SPEED1 = 10;
115     public static final int LIGHTING_SPEED2 = 15;
116     public static final int LIGHTING_DAMAGE = 1;
117     public static final int LIGHTING_HP = 10;
118
119 //の設定 SCORE
120 // STAGE 1
121     public static final int SCORE_STAGE1_PHASE1 = 0;
122     public static final int SCORE_STAGE1_PHASE2 = HARPY_SCORE_POINTS * 1; // 100
123     public static final int SCORE_STAGE1_PHASE3 = HARPY_SCORE_POINTS * 3; // 300
124     public static final int SCORE_FOR_BOSS_1 = HARPY_SCORE_POINTS * 5; // 500
125 // STAGE 2
126     public static final int SCORE_STAGE2_PHASE1 = SCORE_FOR_BOSS_1 +
APOLLO_SCORE_POINTS; //1500
127     public static final int SCORE_STAGE2_PHASE2 = SCORE_STAGE2_PHASE1 + 5; // 2000
128     public static final int SCORE_STAGE2_PHASE3 = SCORE_STAGE2_PHASE2 + 7; // 2750
129     public static final int SCORE_FOR_BOSS_2 = SCORE_STAGE2_PHASE3 + 7; // 3500
130 // STAGE 3
131     public static final int SCORE_STAGE3_PHASE1 = SCORE_FOR_BOSS_2 +
ZEUS_SCORE_POINTS; // 5000
132     public static final int SCORE_STAGE3_PHASE2 = 100000; // 6000
133     public static final int SCORE_STAGE3_PHASE3 = 300000; // 7000
134     public static final int SCORE_FOR_BOSS_3 = 500000; // 8000
135 // EXTRA STAGE
136     public static final int SCORE_EXTRA_STAGE = 10000; // 10000
137
138 // We put them all in an Array
139     public static final int[] LEVEL_MILESTONES = {
140             SCORE_STAGE1_PHASE1, SCORE_STAGE1_PHASE2, SCORE_STAGE1_PHASE3,
SCORE_FOR_BOSS_1,
141             SCORE_STAGE2_PHASE1, SCORE_STAGE2_PHASE2, SCORE_STAGE2_PHASE3,
SCORE_FOR_BOSS_2,
142             SCORE_STAGE3_PHASE1, SCORE_STAGE3_PHASE2, SCORE_STAGE3_PHASE3,
SCORE_FOR_BOSS_3,
143             SCORE_EXTRA_STAGE
```

```

144     };
145
146     //他の設定
147     public static final int FLASH_TIMER = 5;
148     private GameConstants(){} //オブジェクトを作らないようにする private
149 }

```

Listing 4: 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 void setX(int x) {
35        this.x = x;
36    }
37
38    public void setY(int y) {
39        this.y = y;
40    }
41
42    public void setWidth(int width) {
43        this.width = width;
44    }
45
46    public void setHeight(int height) {
47        this.height = height;
48    }
49
50    public void setDead(boolean isDead) {
51        this.isDead = isDead;
52    }
53
54    public boolean isDead() {
55        return isDead;
56    }
57
58    public int getX() {
59        return x;
60    }
61
62    public int getY() {
63        return y;
64    }
65
66    public int getWidth() {
67        return width;
68    }
69
70    public int getHeight() {
71        return height;
72    }
73
74    public BufferedImage getImage() {
75        return image;
76    }
77
78    public void setImage(BufferedImage image) {
79        this.image = image;
80    }
81
82    public void move(int dx, int dy) {
83        x += dx;
84        y += dy;
85    }
86
87    public void move(int x, int y) {
88        this.x = x;
89        this.y = y;
90    }
91
92    public void draw(Graphics2D g) {
93        g.drawImage(image, x, y, width, height);
94    }
95
96    public void setImage(BufferedImage image) {
97        this.image = image;
98    }
99
100   public void setX(int x) {
101      this.x = x;
102  }
103
104  public void setY(int y) {
105     this.y = y;
106  }
107
108  public void setWidth(int width) {
109    this.width = width;
110  }
111
112  public void setHeight(int height) {
113    this.height = height;
114  }
115
116  public void setDead(boolean isDead) {
117    this.isDead = isDead;
118  }
119
120  public boolean isDead() {
121    return isDead;
122  }
123
124  public int getX() {
125    return x;
126  }
127
128  public int getY() {
129    return y;
130  }
131
132  public int getWidth() {
133    return width;
134  }
135
136  public int getHeight() {
137    return height;
138  }
139
140  public BufferedImage getImage() {
141    return image;
142  }
143
144  public void setImage(BufferedImage image) {
145    this.image = image;
146  }
147
148  public void move(int dx, int dy) {
149    x += dx;
150    y += dy;
151  }
152
153  public void move(int x, int y) {
154    this.x = x;
155    this.y = y;
156  }
157
158  public void draw(Graphics2D g) {
159    g.drawImage(image, x, y, width, height);
160  }
161
162  public void setImage(BufferedImage image) {
163    this.image = image;
164  }
165
166  public void setX(int x) {
167    this.x = x;
168  }
169
170  public void setY(int y) {
171    this.y = y;
172  }
173
174  public void setWidth(int width) {
175    this.width = width;
176  }
177
178  public void setHeight(int height) {
179    this.height = height;
180  }
181
182  public void setDead(boolean isDead) {
183    this.isDead = isDead;
184  }
185
186  public boolean isDead() {
187    return isDead;
188  }
189
190  public int getX() {
191    return x;
192  }
193
194  public int getY() {
195    return y;
196  }
197
198  public int getWidth() {
199    return width;
200  }
201
202  public int getHeight() {
203    return height;
204  }
205
206  public BufferedImage getImage() {
207    return image;
208  }
209
210  public void setImage(BufferedImage image) {
211    this.image = image;
212  }
213
214  public void move(int dx, int dy) {
215    x += dx;
216    y += dy;
217  }
218
219  public void move(int x, int y) {
220    this.x = x;
221    this.y = y;
222  }
223
224  public void draw(Graphics2D g) {
225    g.drawImage(image, x, y, width, height);
226  }
227
228  public void setImage(BufferedImage image) {
229    this.image = image;
230  }
231
232  public void setX(int x) {
233    this.x = x;
234  }
235
236  public void setY(int y) {
237    this.y = y;
238  }
239
240  public void setWidth(int width) {
241    this.width = width;
242  }
243
244  public void setHeight(int height) {
245    this.height = height;
246  }
247
248  public void setDead(boolean isDead) {
249    this.isDead = isDead;
250  }
251
252  public boolean isDead() {
253    return isDead;
254  }
255
256  public int getX() {
257    return x;
258  }
259
260  public int getY() {
261    return y;
262  }
263
264  public int getWidth() {
265    return width;
266  }
267
268  public int getHeight() {
269    return height;
270  }
271
272  public BufferedImage getImage() {
273    return image;
274  }
275
276  public void setImage(BufferedImage image) {
277    this.image = image;
278  }
279
280  public void move(int dx, int dy) {
281    x += dx;
282    y += dy;
283  }
284
285  public void move(int x, int y) {
286    this.x = x;
287    this.y = y;
288  }
289
290  public void draw(Graphics2D g) {
291    g.drawImage(image, x, y, width, height);
292  }
293
294  public void setImage(BufferedImage image) {
295    this.image = image;
296  }
297
298  public void setX(int x) {
299    this.x = x;
300  }
301
302  public void setY(int y) {
303    this.y = y;
304  }
305
306  public void setWidth(int width) {
307    this.width = width;
308  }
309
310  public void setHeight(int height) {
311    this.height = height;
312  }
313
314  public void setDead(boolean isDead) {
315    this.isDead = isDead;
316  }
317
318  public boolean isDead() {
319    return isDead;
320  }
321
322  public int getX() {
323    return x;
324  }
325
326  public int getY() {
327    return y;
328  }
329
330  public int getWidth() {
331    return width;
332  }
333
334  public int getHeight() {
335    return height;
336  }
337
338  public BufferedImage getImage() {
339    return image;
340  }
341
342  public void setImage(BufferedImage image) {
343    this.image = image;
344  }
345
346  public void move(int dx, int dy) {
347    x += dx;
348    y += dy;
349  }
350
351  public void move(int x, int y) {
352    this.x = x;
353    this.y = y;
354  }
355
356  public void draw(Graphics2D g) {
357    g.drawImage(image, x, y, width, height);
358  }
359
360  public void setImage(BufferedImage image) {
361    this.image = image;
362  }
363
364  public void setX(int x) {
365    this.x = x;
366  }
367
368  public void setY(int y) {
369    this.y = y;
370  }
371
372  public void setWidth(int width) {
373    this.width = width;
374  }
375
376  public void setHeight(int height) {
377    this.height = height;
378  }
379
380  public void setDead(boolean isDead) {
381    this.isDead = isDead;
382  }
383
384  public boolean isDead() {
385    return isDead;
386  }
387
388  public int getX() {
389    return x;
390  }
391
392  public int getY() {
393    return y;
394  }
395
396  public int getWidth() {
397    return width;
398  }
399
400  public int getHeight() {
401    return height;
402  }
403
404  public BufferedImage getImage() {
405    return image;
406  }
407
408  public void setImage(BufferedImage image) {
409    this.image = image;
410  }
411
412  public void move(int dx, int dy) {
413    x += dx;
414    y += dy;
415  }
416
417  public void move(int x, int y) {
418    this.x = x;
419    this.y = y;
420  }
421
422  public void draw(Graphics2D g) {
423    g.drawImage(image, x, y, width, height);
424  }
425
426  public void setImage(BufferedImage image) {
427    this.image = image;
428  }
429
430  public void setX(int x) {
431    this.x = x;
432  }
433
434  public void setY(int y) {
435    this.y = y;
436  }
437
438  public void setWidth(int width) {
439    this.width = width;
440  }
441
442  public void setHeight(int height) {
443    this.height = height;
444  }
445
446  public void setDead(boolean isDead) {
447    this.isDead = isDead;
448  }
449
450  public boolean isDead() {
451    return isDead;
452  }
453
454  public int getX() {
455    return x;
456  }
457
458  public int getY() {
459    return y;
460  }
461
462  public int getWidth() {
463    return width;
464  }
465
466  public int getHeight() {
467    return height;
468  }
469
470  public BufferedImage getImage() {
471    return image;
472  }
473
474  public void setImage(BufferedImage image) {
475    this.image = image;
476  }
477
478  public void move(int dx, int dy) {
479    x += dx;
480    y += dy;
481  }
482
483  public void move(int x, int y) {
484    this.x = x;
485    this.y = y;
486  }
487
488  public void draw(Graphics2D g) {
489    g.drawImage(image, x, y, width, height);
490  }
491
492  public void setImage(BufferedImage image) {
493    this.image = image;
494  }
495
496  public void setX(int x) {
497    this.x = x;
498  }
499
500  public void setY(int y) {
501    this.y = y;
502  }
503
504  public void setWidth(int width) {
505    this.width = width;
506  }
507
508  public void setHeight(int height) {
509    this.height = height;
510  }
511
512  public void setDead(boolean isDead) {
513    this.isDead = isDead;
514  }
515
516  public boolean isDead() {
517    return isDead;
518  }
519
520  public int getX() {
521    return x;
522  }
523
524  public int getY() {
525    return y;
526  }
527
528  public int getWidth() {
529    return width;
530  }
531
532  public int getHeight() {
533    return height;
534  }
535
536  public BufferedImage getImage() {
537    return image;
538  }
539
540  public void setImage(BufferedImage image) {
541    this.image = image;
542  }
543
544  public void move(int dx, int dy) {
545    x += dx;
546    y += dy;
547  }
548
549  public void move(int x, int y) {
550    this.x = x;
551    this.y = y;
552  }
553
554  public void draw(Graphics2D g) {
555    g.drawImage(image, x, y, width, height);
556  }
557
558  public void setImage(BufferedImage image) {
559    this.image = image;
560  }
561
562  public void setX(int x) {
563    this.x = x;
564  }
565
566  public void setY(int y) {
567    this.y = y;
568  }
569
570  public void setWidth(int width) {
571    this.width = width;
572  }
573
574  public void setHeight(int height) {
575    this.height = height;
576  }
577
578  public void setDead(boolean isDead) {
579    this.isDead = isDead;
580  }
581
582  public boolean isDead() {
583    return isDead;
584  }
585
586  public int getX() {
587    return x;
588  }
589
590  public int getY() {
591    return y;
592  }
593
594  public int getWidth() {
595    return width;
596  }
597
598  public int getHeight() {
599    return height;
600  }
601
602  public BufferedImage getImage() {
603    return image;
604  }
605
606  public void setImage(BufferedImage image) {
607    this.image = image;
608  }
609
610  public void move(int dx, int dy) {
611    x += dx;
612    y += dy;
613  }
614
615  public void move(int x, int y) {
616    this.x = x;
617    this.y = y;
618  }
619
620  public void draw(Graphics2D g) {
621    g.drawImage(image, x, y, width, height);
622  }
623
624  public void setImage(BufferedImage image) {
625    this.image = image;
626  }
627
628  public void setX(int x) {
629    this.x = x;
630  }
631
632  public void setY(int y) {
633    this.y = y;
634  }
635
636  public void setWidth(int width) {
637    this.width = width;
638  }
639
640  public void setHeight(int height) {
641    this.height = height;
642  }
643
644  public void setDead(boolean isDead) {
645    this.isDead = isDead;
646  }
647
648  public boolean isDead() {
649    return isDead;
650  }
651
652  public int getX() {
653    return x;
654  }
655
656  public int getY() {
657    return y;
658  }
659
660  public int getWidth() {
661    return width;
662  }
663
664  public int getHeight() {
665    return height;
666  }
667
668  public BufferedImage getImage() {
669    return image;
670  }
671
672  public void setImage(BufferedImage image) {
673    this.image = image;
674  }
675
676  public void move(int dx, int dy) {
677    x += dx;
678    y += dy;
679  }
680
681  public void move(int x, int y) {
682    this.x = x;
683    this.y = y;
684  }
685
686  public void draw(Graphics2D g) {
687    g.drawImage(image, x, y, width, height);
688  }
689
690  public void setImage(BufferedImage image) {
691    this.image = image;
692  }
693
694  public void setX(int x) {
695    this.x = x;
696  }
697
698  public void setY(int y) {
699    this.y = y;
700  }
701
702  public void setWidth(int width) {
703    this.width = width;
704  }
705
706  public void setHeight(int height) {
707    this.height = height;
708  }
709
710  public void setDead(boolean isDead) {
711    this.isDead = isDead;
712  }
713
714  public boolean isDead() {
715    return isDead;
716  }
717
718  public int getX() {
719    return x;
720  }
721
722  public int getY() {
723    return y;
724  }
725
726  public int getWidth() {
727    return width;
728  }
729
730  public int getHeight() {
731    return height;
732  }
733
734  public BufferedImage getImage() {
735    return image;
736  }
737
738  public void setImage(BufferedImage image) {
739    this.image = image;
740  }
741
742  public void move(int dx, int dy) {
743    x += dx;
744    y += dy;
745  }
746
747  public void move(int x, int y) {
748    this.x = x;
749    this.y = y;
750  }
751
752  public void draw(Graphics2D g) {
753    g.drawImage(image, x, y, width, height);
754  }
755
756  public void setImage(BufferedImage image) {
757    this.image = image;
758  }
759
760  public void setX(int x) {
761    this.x = x;
762  }
763
764  public void setY(int y) {
765    this.y = y;
766  }
767
768  public void setWidth(int width) {
769    this.width = width;
770  }
771
772  public void setHeight(int height) {
773    this.height = height;
774  }
775
776  public void setDead(boolean isDead) {
777    this.isDead = isDead;
778  }
779
780  public boolean isDead() {
781    return isDead;
782  }
783
784  public int getX() {
785    return x;
786  }
787
788  public int getY() {
789    return y;
790  }
791
792  public int getWidth() {
793    return width;
794  }
795
796  public int getHeight() {
797    return height;
798  }
799
800  public BufferedImage getImage() {
801    return image;
802  }
803
804  public void setImage(BufferedImage image) {
805    this.image = image;
806  }
807
808  public void move(int dx, int dy) {
809    x += dx;
810    y += dy;
811  }
812
813  public void move(int x, int y) {
814    this.x = x;
815    this.y = y;
816  }
817
818  public void draw(Graphics2D g) {
819    g.drawImage(image, x, y, width, height);
820  }
821
822  public void setImage(BufferedImage image) {
823    this.image = image;
824  }
825
826  public void setX(int x) {
827    this.x = x;
828  }
829
830  public void setY(int y) {
831    this.y = y;
832  }
833
834  public void setWidth(int width) {
835    this.width = width;
836  }
837
838  public void setHeight(int height) {
839    this.height = height;
840  }
841
842  public void setDead(boolean isDead) {
843    this.isDead = isDead;
844  }
845
846  public boolean isDead() {
847    return isDead;
848  }
849
850  public int getX() {
851    return x;
852  }
853
854  public int getY() {
855    return y;
856  }
857
858  public int getWidth() {
859    return width;
860  }
861
862  public int getHeight() {
863    return height;
864  }
865
866  public BufferedImage getImage() {
867    return image;
868  }
869
870  public void setImage(BufferedImage image) {
871    this.image = image;
872  }
873
874  public void move(int dx, int dy) {
875    x += dx;
876    y += dy;
877  }
878
879  public void move(int x, int y) {
880    this.x = x;
881    this.y = y;
882  }
883
884  public void draw(Graphics2D g) {
885    g.drawImage(image, x, y, width, height);
886  }
887
888  public void setImage(BufferedImage image) {
889    this.image = image;
890  }
891
892  public void setX(int x) {
893    this.x = x;
894  }
895
896  public void setY(int y) {
897    this.y = y;
898  }
899
900  public void setWidth(int width) {
901    this.width = width;
902  }
903
904  public void setHeight(int height) {
905    this.height = height;
906  }
907
908  public void setDead(boolean isDead) {
909    this.isDead = isDead;
910  }
911
912  public boolean isDead() {
913    return isDead;
914  }
915
916  public int getX() {
917    return x;
918  }
919
920  public int getY() {
921    return y;
922  }
923
924  public int getWidth() {
925    return width;
926  }
927
928  public int getHeight() {
929    return height;
930  }
931
932  public BufferedImage getImage() {
933    return image;
934  }
935
936  public void setImage(BufferedImage image) {
937    this.image = image;
938  }
939
940  public void move(int dx, int dy) {
941    x += dx;
942    y += dy;
943  }
944
945  public void move(int x, int y) {
946    this.x = x;
947    this.y = y;
948  }
949
950  public void draw(Graphics2D g) {
951    g.drawImage(image, x, y, width, height);
952  }
953
954  public void setImage(BufferedImage image) {
955    this.image = image;
956  }
957
958  public void setX(int x) {
959    this.x = x;
960  }
961
962  public void setY(int y) {
963    this.y = y;
964  }
965
966  public void setWidth(int width) {
967    this.width = width;
968  }
969
970  public void setHeight(int height) {
971    this.height = height;
972  }
973
974  public void setDead(boolean isDead) {
975    this.isDead = isDead;
976  }
977
978  public boolean isDead() {
979    return isDead;
980  }
981
982  public int getX() {
983    return x;
984  }
985
986  public int getY() {
987    return y;
988  }
989
990  public int getWidth() {
991    return width;
992  }
993
994  public int getHeight() {
995    return height;
996  }
997
998  public BufferedImage getImage() {
999    return image;
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void move(int dx, int dy) {
999    x += dx;
999    y += dy;
999  }
999
999  public void move(int x, int y) {
999    this.x = x;
999    this.y = y;
999  }
999
999  public void draw(Graphics2D g) {
999    g.drawImage(image, x, y, width, height);
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void setX(int x) {
999    this.x = x;
999  }
999
999  public void setY(int y) {
999    this.y = y;
999  }
999
999  public void setWidth(int width) {
999    this.width = width;
999  }
999
999  public void setHeight(int height) {
999    this.height = height;
999  }
999
999  public void setDead(boolean isDead) {
999    this.isDead = isDead;
999  }
999
999  public boolean isDead() {
999    return isDead;
999  }
999
999  public int getX() {
999    return x;
999  }
999
999  public int getY() {
999    return y;
999  }
999
999  public int getWidth() {
999    return width;
999  }
999
999  public int getHeight() {
999    return height;
999  }
999
999  public BufferedImage getImage() {
999    return image;
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void move(int dx, int dy) {
999    x += dx;
999    y += dy;
999  }
999
999  public void move(int x, int y) {
999    this.x = x;
999    this.y = y;
999  }
999
999  public void draw(Graphics2D g) {
999    g.drawImage(image, x, y, width, height);
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void setX(int x) {
999    this.x = x;
999  }
999
999  public void setY(int y) {
999    this.y = y;
999  }
999
999  public void setWidth(int width) {
999    this.width = width;
999  }
999
999  public void setHeight(int height) {
999    this.height = height;
999  }
999
999  public void setDead(boolean isDead) {
999    this.isDead = isDead;
999  }
999
999  public boolean isDead() {
999    return isDead;
999  }
999
999  public int getX() {
999    return x;
999  }
999
999  public int getY() {
999    return y;
999  }
999
999  public int getWidth() {
999    return width;
999  }
999
999  public int getHeight() {
999    return height;
999  }
999
999  public BufferedImage getImage() {
999    return image;
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void move(int dx, int dy) {
999    x += dx;
999    y += dy;
999  }
999
999  public void move(int x, int y) {
999    this.x = x;
999    this.y = y;
999  }
999
999  public void draw(Graphics2D g) {
999    g.drawImage(image, x, y, width, height);
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void setX(int x) {
999    this.x = x;
999  }
999
999  public void setY(int y) {
999    this.y = y;
999  }
999
999  public void setWidth(int width) {
999    this.width = width;
999  }
999
999  public void setHeight(int height) {
999    this.height = height;
999  }
999
999  public void setDead(boolean isDead) {
999    this.isDead = isDead;
999  }
999
999  public boolean isDead() {
999    return isDead;
999  }
999
999  public int getX() {
999    return x;
999  }
999
999  public int getY() {
999    return y;
999  }
999
999  public int getWidth() {
999    return width;
999  }
999
999  public int getHeight() {
999    return height;
999  }
999
999  public BufferedImage getImage() {
999    return image;
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void move(int dx, int dy) {
999    x += dx;
999    y += dy;
999  }
999
999  public void move(int x, int y) {
999    this.x = x;
999    this.y = y;
999  }
999
999  public void draw(Graphics2D g) {
999    g.drawImage(image, x, y, width, height);
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void setX(int x) {
999    this.x = x;
999  }
999
999  public void setY(int y) {
999    this.y = y;
999  }
999
999  public void setWidth(int width) {
999    this.width = width;
999  }
999
999  public void setHeight(int height) {
999    this.height = height;
999  }
999
999  public void setDead(boolean isDead) {
999    this.isDead = isDead;
999  }
999
999  public boolean isDead() {
999    return isDead;
999  }
999
999  public int getX() {
999    return x;
999  }
999
999  public int getY() {
999    return y;
999  }
999
999  public int getWidth() {
999    return width;
999  }
999
999  public int getHeight() {
999    return height;
999  }
999
999  public BufferedImage getImage() {
999    return image;
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void move(int dx, int dy) {
999    x += dx;
999    y += dy;
999  }
999
999  public void move(int x, int y) {
999    this.x = x;
999    this.y = y;
999  }
999
999  public void draw(Graphics2D g) {
999    g.drawImage(image, x, y, width, height);
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void setX(int x) {
999    this.x = x;
999  }
999
999  public void setY(int y) {
999    this.y = y;
999  }
999
999  public void setWidth(int width) {
999    this.width = width;
999  }
999
999  public void setHeight(int height) {
999    this.height = height;
999  }
999
999  public void setDead(boolean isDead) {
999    this.isDead = isDead;
999  }
999
999  public boolean isDead() {
999    return isDead;
999  }
999
999  public int getX() {
999    return x;
999  }
999
999  public int getY() {
999    return y;
999  }
999
999  public int getWidth() {
999    return width;
999  }
999
999  public int getHeight() {
999    return height;
999  }
999
999  public BufferedImage getImage() {
999    return image;
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void move(int dx, int dy) {
999    x += dx;
999    y += dy;
999  }
999
999  public void move(int x, int y) {
999    this.x = x;
999    this.y = y;
999  }
999
999  public void draw(Graphics2D g) {
999    g.drawImage(image, x, y, width, height);
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void setX(int x) {
999    this.x = x;
999  }
999
999  public void setY(int y) {
999    this.y = y;
999  }
999
999  public void setWidth(int width) {
999    this.width = width;
999  }
999
999  public void setHeight(int height) {
999    this.height = height;
999  }
999
999  public void setDead(boolean isDead) {
999    this.isDead = isDead;
999  }
999
999  public boolean isDead() {
999    return isDead;
999  }
999
999  public int getX() {
999    return x;
999  }
999
999  public int getY() {
999    return y;
999  }
999
999  public int getWidth() {
999    return width;
999  }
999
999  public int getHeight() {
999    return height;
999  }
999
999  public BufferedImage getImage() {
999    return image;
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void move(int dx, int dy) {
999    x += dx;
999    y += dy;
999  }
999
999  public void move(int x, int y) {
999    this.x = x;
999    this.y = y;
999  }
999
999  public void draw(Graphics2D g) {
999    g.drawImage(image, x, y, width, height);
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void setX(int x) {
999    this.x = x;
999  }
999
999  public void setY(int y) {
999    this.y = y;
999  }
999
999  public void setWidth(int width) {
999    this.width = width;
999  }
999
999  public void setHeight(int height) {
999    this.height = height;
999  }
999
999  public void setDead(boolean isDead) {
999    this.isDead = isDead;
999  }
999
999  public boolean isDead() {
999    return isDead;
999  }
999
999  public int getX() {
999    return x;
999  }
999
999  public int getY() {
999    return y;
999  }
999
999  public int getWidth() {
999    return width;
999  }
999
999  public int getHeight() {
999    return height;
999  }
999
999  public BufferedImage getImage() {
999    return image;
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void move(int dx, int dy) {
999    x += dx;
999    y += dy;
999  }
999
999  public void move(int x, int y) {
999    this.x = x;
999    this.y = y;
999  }
999
999  public void draw(Graphics2D g) {
999    g.drawImage(image, x, y, width, height);
999  }
999
999  public void setImage(BufferedImage image) {
999    this.image = image;
999  }
999
999  public void setX(int x) {
999    this.x = x;
999  }
999
999  public void setY(int y) {
999    this.y = y;
999  }
999
999  public void setWidth(int width) {
999    this.width = width;
999  }
999
999  public void setHeight(int height) {
999    this.height = height;
999  }
999
999  public void setDead(boolean isDead) {
999    this.isDead = isDead;
999  }
999
999  public boolean isDead() {
999    return isDead;
999  }
999
999  public int getX() {
999   
```

```

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 5: 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 6: 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 7: Background.java

```
1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 /**
8 * Background Class
9 * Handles the infinite scrolling background using a "Mirroring" technique.
10 * It draws the original image and a vertically flipped copy above it
11 * to create a seamless loop without needing a specific seamless texture.
12 */
13 public class Background {
14     private double y; // Vertical position (double for smooth movement)
15     private double speed; // Scrolling speed
16     private BufferedImage image;
17
18     // Screen dimensions
19     private final int WIDTH = GameConstants.WINDOW_WIDTH;
20     private final int HEIGHT = GameConstants.FIELD_HEIGHT / 2;
21
22     public Background() {
23         this.image = ResourceManager.stage1Img;
24         this.speed = 0;
25         this.y = GameConstants.HUD_HEIGHT;
26     }
27
28     public void setImage(BufferedImage newImage){
29         this.image = newImage;
30     }
31
32     public void setSpeed(double newSpeed) {
33         this.speed = newSpeed;
34     }
35
36     public void update() {
```

```

37     // Move the background downwards
38     y += speed;
39
40     // Reset position when a full cycle is completed to prevent overflow
41     if (y >= HEIGHT * 2 + GameConstants.HUD_HEIGHT) {
42         y = GameConstants.HUD_HEIGHT;
43     }
44 }
45
46 public void draw(Graphics g) {
47     if (image == null) return;
48
49     int currentY = (int) y;
50
51     if (isStage2Img()){
52         // DRAWING STRATEGY:
53         // The screen is 800px tall. Our "Tile" is 400px.
54         // We need to cover the screen from Y=0 to Y=800.
55         // Since 'currentY' moves down, we need to draw tiles above and
below it.
56
57         // We assume the pattern is: [Normal] [Flipped] [Normal] [Flipped]
58         ...
59
60         // 1. Draw Normal Tile at current Y
61         // Covers: y to y+400  x
62         drawForceSize(g, currentY, false);
63
64         // 2. Draw Flipped Tile BELOW
65         // Covers: y+400 to y+800
66         drawForceSize(g, currentY + HEIGHT, true);
67
68         // 3. Draw Flipped Tile ABOVE
69         // Covers: y-400 to y. (Crucial for when y starts at 0 or is small
70     )
71         drawForceSize(g, currentY - HEIGHT, true);
72
73         // 4. Draw Normal Tile ABOVE that
74         // Covers: y-800 to y-400. (Crucial for the loop wrap-around)
75         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 8: Projectile.java

```

1 package model;
2
3 import view.ResourceManager;
4
5 import java.awt.*;
6 import java.awt.image.BufferedImage;
7
8 /**
9  * Projectile Class
```

```

10 * Abstract base class for all flying objects (Arrows, Feathers, Suns,
11   Boulders).
12 * It introduces the concept of "Power Level" to handle projectile-vs-
13   projectile collisions.
14 */
15 public abstract class Projectile extends GameObject {
16
17     protected Alignment alignment;
18
19     // Power Level determines priority when two projectiles collide:
20     // 0 = Ephemeral (Destroyed by anything)
21     // 1 = Light (Arrow, Feather)
22     // 2 = Heavy (Boulder - Destroys Light)
23     // 3 = Ultimate (Sun - Destroys Heavy and Light)
24     protected int powerLevel;
25
26     protected int damage;
27
28     // If true, the projectile does not vanish after hitting a target (e.g.,
29     // The Sun)
30     protected boolean isPenetrating;
31
32     public Projectile(int x, int y, int w, int h, BufferedImage image,
33                       Alignment alignment, int powerLevel, int damage) {
34         super(x, y, w, h, image);
35         this.alignment = alignment;
36         this.powerLevel = powerLevel;
37         this.damage = damage;
38         this.isPenetrating = false; // Default: destroys itself on impact
39     }
40
41     // Getters
42     public Alignment getAlignment() { return alignment; }
43     public int getPowerLevel() { return powerLevel; }
44     public int getDamage() { return damage; }
45     public boolean isPenetrating() { return isPenetrating; }
46 }
```

Listing 9: Player.java



```

37         // Fallback if image failed to load
38         g.setColor(Color.BLUE);
39         g.fillRect(x, y, width, height);
40     }
41 }
42
43 @Override
44 public Shape getShape() {
45     // we define the margin
46     float paddingX = width * 0.3f; // remove 20 % width
47     float paddingY = height * 0.2f; // remove 10 % height
48
49     // the smaller hitbox get centered compared to the original immage
50     return new Ellipse2D.Float(
51             x + paddingX / 2,           // move right
52             y + paddingY / 2,           // move down
53             width - paddingX,          // reduce width
54             height - paddingY);       // reduce height
55 );
56 }
57
58 // から呼ばれるメソッド Controller
59 public void setVelX(int vx) {
60     this.velX = vx * speed;
61 }
62
63 public void setVelY(int vy) {
64     this.velY = vy * speed;
65 }
66
67 public int getLevel(){
68     return level;
69 }
70 }
```

Listing 10: Arrow.java

```

1 package model;
2
3 import view.ResourceManager;
```

```

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

Listing 11: 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 12: 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){

```

```

17         model.bossDefeated();
18     }
19 }
20 }
```

Listing 13: Apollo.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.image.BufferedImage;
6
7 /**
8 * Apollo Class
9 * Represents the first Boss of the game.
10 * It moves horizontally, bounces off walls, and shoots "Sun" projectiles.
11 * It has two phases: Normal and Enraged (Red & Fast).
12 */
13 public class Apollo extends Boss {
14
15     private int speedX = GameConstants.APOLLO_SPEED1;
16     private boolean secondPhase = false; // Flag to track if the boss is in "
17                                         // Rage Mode"
18
19     public Apollo(GameModel model) {
20         // Call the parent constructor (Boss -> HostileEntity)
21         // Parameters: x, y, width, height, HP, Score Points
22         super(
23             (GameConstants.WINDOW_WIDTH - GameConstants.APOLLO_WIDTH) / 2,
24             // Start in the middle
25             GameConstants.HUD_HEIGHT, // Start below HUD
26             GameConstants.APOLLO_WIDTH,
27             GameConstants.APOLLO_HEIGHT,
28             ResourceManager.apolloImg,
29             GameConstants.APOLLO_HP,
30             GameConstants.APOLLO_SCORE_POINTS, // Score awarded when
31             defeated
32             model
33         );
34     }
35 }
```

```

31     this.image = ResourceManager.apolloImg;
32 }
33
34 @Override
35 public void move() {
36     super.move();
37     // Update horizontal position
38     x += speedX;
39
40     // Bounce logic: If it hits the screen edges
41     if (x <= 0 || x >= GameConstants.WINDOW_WIDTH - width) {
42         speedX = -speedX; // Reverse direction
43
44         // Trigger shooting mechanism via GameModel
45         // We pass the current phase status to decide if the Sun should be
46         // Red/Fast
47         shootSun();
48     }
49
50     private void shootSun(){
51         Sun s = new Sun(x, y, speedX, secondPhase, false);
52         model.spawnEnemyProjectile(s);
53     }
54
55     @Override
56     public void takeDamage(int dmg) {
57         // 1. Apply damage using the parent class logic (reduces HP, checks
58         // death, adds score)
59         super.takeDamage(dmg);
60
61         // 2. Specific Logic for Apollo: Check for Second Phase (Rage Mode)
62         // If HP drops below 50% and we are not yet in the second phase...
63         if (hp <= maxHp / 2 && !secondPhase) {
64             image = ResourceManager.apolloImg2; // Change sprite to Red Apollo
65             speedX = (speedX > 0) ? GameConstants.APOLLO_SPEED2 : -
66             GameConstants.APOLLO_SPEED2; // Double the movement speed
67             secondPhase = true; // Activate the flag
68             System.out.println("Apollo entering Phase 2!");
69         }

```

```

68    }
69
70    @Override
71    public void draw(Graphics g) {
72        BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
73            apolloHitImg : image;
74
75        if (image != null) {
76            // Directional Flipping Logic
77            if (speedX > 0) {
78                // Moving RIGHT: Draw normally
79                g.drawImage(imgToDraw, x, y, width, height, null);
80            } else {
81                // Moving LEFT: Flip the image horizontally
82                // We draw starting at (x + width) and use a negative width to
83                // flip
84                g.drawImage(imgToDraw, x + width, y, -width, height, null);
85            }
86        } else {
87            // Fallback: Draw an Orange rectangle if images fail to load
88            g.setColor(Color.ORANGE);
89            g.fillRect(x, y, width, height);
90        }
91    }

```

Listing 14: 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     }
34
35     this.initialSize = GameConstants.SUN_WIDTH;
36
37     velX = 1;
38     velY = 1;
39     double currentSpeed;
40     double angleRadians;
41
42     if (isSecondPhase) {
43         currentSpeed = GameConstants.SUN_SPEED2;
44     } else {
45         currentSpeed = GameConstants.SUN_SPEED1;
46     }
47
48     if (!friendly){
49         // sun comes from apollo
50         preciseX = (ApolloSpeedX > 0) ? (summonerX + GameConstants.

```

```

APOLLO_WIDTH + width / 2) : (summonerX - width / 2);
51     preciseY = summonerY + height;
52     x = (int) preciseX;
53     y = (int) preciseY;
54     // --- Phase Logic ---
55     // --- Trajectory Calculation ---
56     double minAngle = 20.0;
57     double maxAngle = 70;
58     angleRadians = Math.toRadians(minAngle + Math.random() * (maxAngle
59     - minAngle));
60 } else {
61     // sun comes from ability
62     preciseX = summonerX;
63     preciseY = summonerY - height / 2;
64
65     // 2. right boarder check
66     if (preciseX > GameConstants.WINDOW_WIDTH - width / 2) {
67         preciseX = GameConstants.WINDOW_WIDTH - width / 2;
68     }
69
70     // 3. left boarder check
71     if (preciseX < width / 2) {
72         preciseX = width / 2;
73     }
74
75     x = (int) preciseX;
76     y = (int) preciseY;
77     angleRadians = Math.toRadians(360-30);
78     velX = (Math.random() < 0.5) ? 1 : -1;
79 }
80
81     velX *= currentSpeed * Math.cos(angleRadians);
82     velY *= currentSpeed * Math.sin(angleRadians);
83
84     if (ApolloSpeedX < 0) {
85         velX = -velX;
86     }
87
88 @Override

```

```

89     public void move() {
90         preciseX += velX;
91         preciseY += velY;
92
93         x = (int) preciseX;
94         y = (int) preciseY;
95
96         double currentRadius = getCurrentSize() / 2.0;
97
98         // Wall Bounce
99         if (x - currentRadius < 0) {
100             x = (int)currentRadius;
101             preciseX = x;
102             velX = -velX;
103         }
104         if (x + currentRadius > GameConstants.WINDOW_WIDTH) {
105             x = (int)(GameConstants.WINDOW_WIDTH - currentRadius);
106             preciseX = x;
107             velX = -velX;
108         }
109
110         // Despawn
111         if (y - currentRadius > GameConstants.FIELD_HEIGHT + GameConstants.
112             HUD_HEIGHT ||
113             y + currentRadius < GameConstants.HUD_HEIGHT) {
114             isDead = true;
115         }
116
117         // Helper to calculate size based on HP
118         private double getCurrentSize() {
119             if (!isPlayerProjectile) return initialSize;
120
121             // Calculate ratio: currentHP / maxHP
122             double ratio = (double) currentHP / maxHP;
123             return initialSize * ratio;
124         }
125
126         @Override
127         public void draw(Graphics g) {

```

```

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

Listing 15: 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.ZUES_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         }

```

```

91     if (x <= 0 ) {
92         x = 0;
93         speedX = -speedX; // Reverse direction
94         ability2Repetitions--;
95     } else if (x >= GameConstants.WINDOW_WIDTH - width) {
96         x = GameConstants.WINDOW_WIDTH - width;
97         speedX = -speedX; // Reverse direction
98         ability2Repetitions--;
99     }
100    if (ability2Repetitions == 0){
101        ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
102        ability2Started = false;
103        resetShootTimer();
104    }
105
106 }
107
108 if (secondPhase && ability2Timer > 0){
109     ability2Timer--;
110 }
111
112
113 private void ability1() {
114     if (ability1Position == 1){
115         ability1Phase = true;
116     }
117
118     if (ability1Position == 5) {
119         setAbility1Timer();
120         ability1Position = 1;
121         resetShootTimer();
122         return;
123     }
124
125     if(speedX > 0) {
126         x = ability1Position * (GameConstants.WINDOW_WIDTH-width)/4;
127     } else {
128         x = (4 - ability1Position) * (GameConstants.WINDOW_WIDTH-width)/4;
129     }
130     if (ability1Position % 2 == 0){

```

```

131         y = GameConstants.HUD_HEIGHT;
132     } else {
133         y = GameConstants.HUD_HEIGHT + height / 3;
134     }
135
136     shootLighting();
137     ability1Pause = maxAbility1Pause;
138     ability1Position++;
139 }
140
141 private void shootLighting(){
142     Lighting l = new Lighting(x, y, speedX, secondPhase, false,
143     ability2Started);
144     model.spawnEnemyProjectile(l);
145 }
146
147 @Override
148 public void takeDamage(int dmg) {
149     super.takeDamage(dmg);
150
151     if (hp <= maxHp / 2 && !secondPhase) {
152         image = ResourceManager.zeusImg2;
153         hitImg = ResourceManager.zeusHitImg2;
154         speedX = (speedX > 0) ? GameConstants.ZEUS_SPEED2 : -GameConstants
155         .ZEUS_SPEED2; // Double the movement speed
156         secondPhase = true;
157         maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER2;
158         maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE2;
159     }
160 }
161
162 private void setAbility1Timer(){
163     Random random = new Random();
164     ability1Timer = random.nextInt(4) + 1;
165     ability1Phase = false;
166 }
167
168 private void resetShootTimer(){
169     shootTimer = maxShootTimer;
170 }
```

```

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 16: 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);
```

```

19
20     this.isPlayerProjectile = friendly;
21     this.isPenetrating = true;
22
23     if (isSecondPhase) {
24         vely = GameConstants.LIGHTING_SPEED2;
25     } else {
26         vely = GameConstants.LIGHTING_SPEED1;
27     }
28
29     if(isPlayerProjectile){
30         vely = -vely;
31         maxHP = GameConstants.LIGHTING_HP;
32         currentHP = maxHP;
33         // set position for player
34         this.x = x + (GameConstants.PLAYER_WIDTH - width) / 2;
35         this.y = y - GameConstants.PLAYER_HEIGHT;
36     } else {
37         if (!ability2Started){
38
39             // set position for Zeus
40             this.x = (ZeusSpeedX > 0) ? x + GameConstants.ZEUS_WIDTH - width :
41             x + width;
42             } else {
43                 this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
44             width;
45             }
46             this.y = y + height;
47         }
48
49     @Override
50     public void move() {
51         y += vely;
52
53         if (y > GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT || y <
54         GameConstants.HUD_HEIGHT - height) {
55             isDead = true;
56         }
57     }

```

```

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 17: 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 18: EnemySpawner.java

```

1 package model;
2
```

```

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

```

```

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

```

Listing 19: Boulder.java

```

1 package model;
2
3 import view.ResourceManager;
4 import java.awt.*;
5 import java.awt.geom.Ellipse2D;
6 import java.awt.image.BufferedImage;
7
8 /**
9  * Boulder Class
10 * A heavy projectile dropped by the StoneGolem.
11 * It has Power Level 2, meaning it destroys Arrows/Feathers but is destroyed
12 * by the Sun.
13 */
14 public class Boulder extends Projectile {

```

```

14
15     private double preciseY;
16     private double velY;
17     private double gravity = GameConstants.Boulder_GRAVITY; // Accelerates
18     downwards
19
20     public Boulder(int x, int y) {
21         super(x, y,
22             GameConstants.Boulder_WIDTH,
23             GameConstants.Boulder_HEIGHT,
24             ResourceManager.boulderImg,
25             Alignment.ENEMY,
26             2,
27             GameConstants.Boulder_DAMAGE
28         );
29
30         this.preciseY = y;
31         this.velY = GameConstants.Boulder_INITIAL_SPEED; // Initial throw
32         speed
33     }
34
35     @Override
36     public void move() {
37         // Apply Gravity
38         velY += gravity;
39         preciseY += velY;
40         y = (int) preciseY;
41
42         // Despawn if off-screen
43         if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT) {
44             isDead = true;
45         }
46     }
47
48     @Override
49     public void draw(Graphics g) {
50         if (image != null) {
51             g.drawImage(image, x, y, width, height, null);
52         } else {
53             // 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 20: Cyclops.java

```

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

```

27     this.movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
28     this.attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
29     this.reachedMidScreen = false;
30 }
31
32 @Override
33 public void move() {
34     super.move();
35     if (movementTimer <= 0){
36         if (vely > 0){
37             // Go up
38             vely = GameConstants.CYCLOPS_YSPEED / -4;
39             movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 / 2;
40             wingsClosed = true; // STATE: closed wings
41         }
42         else {
43             // go down
44             wingsClosed = false; // STATE: opened wings
45
46             if (!reachedMidScreen) {
47                 vely = GameConstants.CYCLOPS_YSPEED;
48                 movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER;
49             } else {
50                 vely = GameConstants.CYCLOPS_YSPEED / 4;
51                 movementTimer = GameConstants.CYCLOPS_MOVEMENT_TIMER * 3 /
52                     2;
53             }
54         }
55         this.image = wingsClosed ? ResourceManager.cyclopsImg2 :
56         ResourceManager.cyclopsImg;
57     }
58     preciseY += vely;
59     y = (int) preciseY;
60
61     if(y > (GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT/2 -
62     height)){
63         reachedMidScreen = true;
64     }
65
66     // CHECK IF THERE IS BOULDER OR IF IT IS DEAD

```

```

64     if (myBoulder == null || myBoulder.isDead()) {
65         // REDUCE TIMER
66         if(attackTimer > 0){
67             attackTimer--;
68         }
69         // ONCE THE TIMER IS 0 SPAWN BOULDER AND RESET TIMER
70         if (attackTimer == 0){
71             throwBoulder();
72             attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
73         }
74     }
75     movementTimer--;
76 }
77
78 @Override
79 public void draw(Graphics g) {
80     BufferedImage imgToDraw;
81
82     if (image == ResourceManager.cyclopsImg){
83         imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg :
84         image;
85     } else {
86         imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg2 :
87         image;
88     }
89
90     if (image != null) {
91         g.drawImage(imgToDraw, x, y, width, height, null);
92     } else {
93         g.setColor(Color.RED);
94         g.fillRect(x, y, width, height);
95     }
96 }
97
98 private void throwBoulder(){
99     Boulder b = new Boulder (x + (width - GameConstants.Boulder_WIDTH)/2,
y + height);
100    this.myBoulder = b;
101    model.spawnEnemyProjectile(b);
102 }
```

101 }

Listing 21: Feather.java

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

Listing 22: Harpy.java

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

```
40 y += velY;
41
42     // Bounce Logic
43     if (x < 0) {
44         x = 0;
45         velX = -velX;
46     }
47     if (x > GameConstants.WINDOW_WIDTH - width) {
48         x = GameConstants.WINDOW_WIDTH - width;
49         velX = -velX;
50     }
51     if (y < GameConstants.HUD_HEIGHT && isInScreen) {
52         y = GameConstants.HUD_HEIGHT;
53         velY = -velY;
54     }
55     if (y > GameConstants.HUD_HEIGHT) {
56         isInScreen = true;
57     }
58     if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height)
59 ) {
60     y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
61 ;
62     velY = -velY;
63 }
64
65     if(fireTimer > 0) {
66         fireTimer--;
67     }
68     if (fireTimer <= 0) {
69         throwFeather();
70         resetFireTimer();
71     }
72 }
73
74 private void throwFeather() {
75     Feather f = new Feather (x + (width - GameConstants.FEATHER_WIDTH)/2,
76 y + height);
77     model.spawnEnemyProjectile(f);
78 }
```

```

77     public void resetFireTimer() {
78         int base = GameConstants.FEATHER_FIRE_INTERVAL;
79         int variance = GameConstants.FEATHER_FIRE_VARIANCE;
80
81         int randomVariation = (int)(Math.random() * (variance * 2)) - variance
82 ;
83
84         this.fireTimer = base + randomVariation;
85     }
86
87     @Override
88     public void draw(Graphics g) {
89         BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
90 harpyHitImg : image;
91
92         if (image != null) {
93             if (velX < 0) {
94                 g.drawImage(imgToDraw, x, y, width, height, null);
95             } else {
96                 g.drawImage(imgToDraw, x + width, y, -width, height, null);
97             }
98         } else {
99             g.setColor(Color.RED);
100            g.fillRect(x, y, width, height);
101        }
102    }
103 }
```

Listing 23: Minion.java

```

1 package model;
2
3 import java.awt.image.BufferedImage;
4
5 /**
6  * Minion Class
7  * Abstract class representing basic non-boss enemies.
8  * Used to group Harpies, Golems, etc., and manage shared logic like scoring.
9  */
10 public abstract class Minion extends HostileEntity {
```

```
11     public Minion(int x, int y, int w, int h, BufferedImage image, int hp, int
12         scorePoints, GameModel model) {
13     super(x, y, w, h, image, hp, scorePoints, model);
14 }
```

Listing 24: 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 }
```

```

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

```

```

71     }
72 }
73
74 // Main Game Drawing Method
75 private void drawGameScreen(Graphics g) {
76
77     // 1. Draw Background
78     if (model.getBackground() != null) {
79         model.getBackground().draw(g);
80     } else {
81         // Fallback: Black background if image is missing
82         g.setColor(Color.BLACK);
83         // Fill only the game field area
84         g.fillRect(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH
85         , GameConstants.FIELD_HEIGHT);
86     }
87
88     // 2. Draw Game Objects (Player, Enemies, Projectiles)
89     for (GameObject obj : model.getObjects()) {
90         // Invincibility flashing logic for Player
91         if (obj instanceof Player && model.isInvincible()) {
92             // Toggle visibility every 100ms
93             if (System.currentTimeMillis() % 200 < 100) {
94                 continue;
95             }
96             obj.draw(g);
97         }
98
99         // 3. Draw Top HUD (Score, Stage, Lives)
100        drawTopHUD(g);
101
102         // 4. Draw Bottom HUD (Ability Slots) -> NEW!
103        drawBottomHUD(g);
104    }
105
106    // Helper method to draw the Top HUD
107    private void drawTopHUD(Graphics g) {
108        // Draw dark background bar
109        g.setColor(new Color(50, 50, 80));

```

```

110     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.HUD_HEIGHT)
111 ;
112
113     // Draw white separator line
114     g.setColor(Color.WHITE);
115     g.drawLine(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH,
116     GameConstants.HUD_HEIGHT);
117
118     // Font settings
119     setPixelFont(g, 18f);
120     int textY = 35;
121
122     // 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;
136     int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (heartSize
137     + spacing));
138     int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
139
140     for (int i = 0; i < maxLives; i++) {
141         // Determine which icon to draw (Full or Empty)
142         BufferedImage icon = (i < currentLives) ? ResourceManager.
143         heartFullImg : ResourceManager.heartEmptyImg;
144
145         if (icon != null) {
146             g.drawImage(icon, startX + (i * (heartSize + spacing)), heartY
147             , heartSize, heartSize, null);
148         } else {
149             // Fallback drawing if images are missing

```

```

144         g.setColor(i < currentLives ? Color.RED : Color.GRAY);
145         g.fillOval(startX + (i * (heartSize + spacing)), heartY,
146         heartSize, heartSize);
147     }
148 }
149
150 // Helper method to draw the Bottom HUD (Ability Slots)
151 // Helper method to draw the Bottom HUD (Ability Slots)
152 private void drawBottomHUD(Graphics g) {
153     // Calculate start Y position (below the game field)
154     int startY = GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT;
155     int height = GameConstants.BOTTOM_HUD_HEIGHT;
156
157     // 1. Background Bar
158     g.setColor(new Color(50, 50, 80));
159     g.fillRect(0, startY, GameConstants.WINDOW_WIDTH, height);
160
161     // 2. Separator Line
162     g.setColor(Color.WHITE);
163     g.drawLine(0, startY, GameConstants.WINDOW_WIDTH, startY);
164
165     // 3. Draw Slots
166     int slotSize = 60;
167     int gap = 40;
168     int totalWidth = (3 * slotSize) + (2 * gap);
169     int startX = (GameConstants.WINDOW_WIDTH - totalWidth) / 2;
170     int slotY = startY + (height - slotSize) / 2 - 9;
171
172     setPixelFont(g, 14f);
173
174     for (int i = 0; i < 3; i++) {
175         int x = startX + (i * (slotSize + gap));
176
177         // A. Draw Slot Background (Black)
178         g.setColor(Color.BLACK);
179         g.fillRect(x, slotY, slotSize, slotSize);
180
181         // Check if this is the first slot (Index 0) AND if Ability 1 is
unlocked

```

```

182     if (i == 0 && model.isAbilityUnclocked(1)) {
183
184         // 1. Draw the Icon (The Sun)
185         if (ResourceManager.sunImg != null) {
186             g.drawImage(ResourceManager.sunImg, x, slotY, slotSize,
187             slotSize, null);
188         }
189
190         // 2. Draw Cooldown Overlay (The fading effect)
191         int timer = model.getAbilityNthTimer(1);
192         int maxTime = GameConstants.ABILITY1TIMER; // Make sure this
193         is set correctly in Constants!
194
195         if (timer > 0) {
196             // Calculate percentage of time remaining (0.0 to 1.0)
197             float ratio = (float) timer / maxTime;
198
199             // Calculate height of the dark overlay based on the ratio
200             // If ratio is 1.0 (just used), height is full (60).
201             // If ratio is 0.5, height is half (30), covering the top
202             half.
203
204             // This creates the effect of the color "filling up from
205             // bottom".
206             int overlayHeight = (int) (slotSize * ratio);
207
208             // Set color to semi-transparent black
209             g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (
210             transparency)
211
212             // Draw the overlay from the top of the slot downwards
213             g.fillRect(x, slotY, slotSize, overlayHeight);
214
215             // Optional: Draw the text timer on top if you want
216             g.setColor(Color.WHITE);
217             String keyNum = String.valueOf(timer/60 + 1);
218             int numWidth = g.getFontMetrics().stringWidth(keyNum);
219             g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY
220             + 37);
221         }
222     }

```

```

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
241             // This creates the effect of the color "filling up from
242             // bottom".
243             int overlayHeight = (int) (slotSize * ratio);
244
245             // Set color to semi-transparent black
246             g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (
247             // transparency)
248
249             // Draw the overlay from the top of the slot downwards
250             g.fillRect(x, slotY, slotSize, overlayHeight);
251
252             // Optional: Draw the text timer on top if you want
253             g.setColor(Color.WHITE);
254             String keyNum = String.valueOf(timer/60 + 1);
255             int numWidth = g.getFontMetrics().stringWidth(keyNum);
256             g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY

```

```

+ 37);
    }
}
// -----
253
254     // B. Draw Slot Border
255     g.setColor(Color.GRAY);
256     g.drawRect(x, slotY, slotSize, slotSize);
257
258     // C. Draw Key Number (1, 2, 3)
259     g.setColor(Color.WHITE);
260     String keyNum = String.valueOf(i + 1);
261     int numWidth = g.getFontMetrics().stringWidth(keyNum);
262     g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY +
slotSize + 25);
263 }
264
265
266 // Draw Title Screen
267 private void drawTitleScreen(Graphics g) {
268     g.setColor(Color.WHITE);
269
270     // Title: Big Pixel Font
271     setPixelFont(g, 28f);
272     String title = "GLADIATOR GAME"; // Cambia il nome se vuoi
273     int titleWidth = g.getFontMetrics().stringWidth(title);
274     g.drawString(title, (GameConstants.WINDOW_WIDTH - titleWidth)/2, 250);
275
276     // Subtitle: Smaller
277     setPixelFont(g, 15f);
278     String msg = "Press SPACE to Start";
279     int msgWidth = g.getFontMetrics().stringWidth(msg);
280     g.drawString(msg, (GameConstants.WINDOW_WIDTH - msgWidth)/2, 350);
281 }
282
283 private void drawPauseScreen(Graphics g) {
284     // 1. Semi-transparent black overlay
285     g.setColor(new Color(0, 0, 0, 150)); // 150 = Alpha (Transparency)
286     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
WINDOW_HEIGHT);

```

```

287
288     // 2. "PAUSE" Text
289     g.setColor(Color.WHITE);
290     setPixelFont(g, 40f); // Large font
291     String pauseText = "PAUSE";
292     int pauseWidth = g.getFontMetrics().stringWidth(pauseText);
293     // Center text
294     g.drawString(pauseText, (GameConstants.WINDOW_WIDTH - pauseWidth) / 2,
295     GameConstants.WINDOW_HEIGHT / 2 - 100);
296
297     // 3. Instruction Text
298     setPixelFont(g, 20f); // Smaller font
299     String resumeText = "Press [P] to Resume";
300     int resumeWidth = g.getFontMetrics().stringWidth(resumeText);
301     g.drawString(resumeText, (GameConstants.WINDOW_WIDTH - resumeWidth) /
302     2, GameConstants.WINDOW_HEIGHT / 2 - 50);
303 }
304
305 private void drawMessageScreen(Graphics g) {
306     // 1. Semi-transparent black background for the whole screen (dimming)
307     g.setColor(new Color(0, 0, 0, 100));
308     g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
309     WINDOW_HEIGHT);
310
311     // 2. The Message Box Dimensions
312     int boxWidth = 500;
313     int boxHeight = 400;
314     int boxX = (GameConstants.WINDOW_WIDTH - boxWidth) / 2;
315     int boxY = (GameConstants.WINDOW_HEIGHT - boxHeight) / 2;
316
317     // 3. Draw the Box Background (Dark Blue)
318     g.setColor(new Color(20, 20, 80));
319     g.fillRect(boxX, boxY, boxWidth, boxHeight);
320
321     // 4. Draw the Box Border (White)
322     g.setColor(Color.WHITE);
323     Graphics2D g2 = (Graphics2D) g;
324     g2.setStroke(new BasicStroke(4)); // Thicker border
325     g2.drawRect(boxX, boxY, boxWidth, boxHeight);

```

```

324     // 5. Draw the Text
325     String[] lines = model.getCurrentMessageLines();
326     if (lines != null) {
327         setPixelFont(g, 20f); // Size for text
328         g.setColor(Color.WHITE);
329
330         int lineHeight = 30;
331         // Calculate starting Y to center the block of text vertically
332         int totalTextHeight = lines.length * lineHeight;
333         int startTextY = boxY + (boxHeight - totalTextHeight) / 2 + 10; // +10 adjustment
334
335         for (int i = 0; i < lines.length; i++) {
336             String line = lines[i];
337             // Center align each line horizontally
338             int lineWidth = g.getFontMetrics().stringWidth(line);
339             int lineX = (GameConstants.WINDOW_WIDTH - lineWidth) / 2;
340
341             g.drawString(line, lineX, startTextY + (i * lineHeight));
342         }
343     }
344
345     // 6. Draw "Press Space" prompt at the bottom of the box
346     setPixelFont(g, 14f);
347     g.setColor(Color.YELLOW);
348     String prompt = "PRESS [SPACE] TO CONTINUE";
349     int promptWidth = g.getFontMetrics().stringWidth(prompt);
350     g.drawString(prompt, (GameConstants.WINDOW_WIDTH - promptWidth) / 2,
351     boxY + boxHeight - 20);
352
353     // Draw Game Over Screen
354     private void drawGameOverScreen(Graphics g) {
355         // Semi-transparent overlay
356         g.setColor(new Color(0, 0, 0, 150));
357         g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
358         WINDOW_HEIGHT);
359
360         // Game Over Text
361         g.setColor(Color.RED);

```

```

361     setPixelFont(g, 35f); // Big Red Text
362     String GO = "GAME OVER";
363     int goWidth = g.getFontMetrics().stringWidth(GO);
364     g.drawString(GO, (GameConstants.WINDOW_WIDTH - goWidth)/2, 250);
365
366     g.setColor(Color.WHITE);
367     setPixelFont(g, 20f);
368
369     String scoreMsg = "Final Score: " + model.getScore();
370     int scoreWidth = g.getFontMetrics().stringWidth(scoreMsg);
371     g.drawString(scoreMsg, (GameConstants.WINDOW_WIDTH - scoreWidth)/2,
372     320);
373
374     // ... Time e Quit/Continue (usa la stessa logica per centrare) ...
375     String cont = "[C] Continue";
376     String quit = "[Q] Quit";
377
378     g.drawString(cont, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
379     stringWidth(cont))/2, 400);
380     g.drawString(quit, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
381     stringWidth(quit))/2, 440);
382 }
383
384 // Update Player Velocity based on key states
385 private void updatePlayerVelocity() {
386     Player p = model.getPlayer();
387
388     int vx = 0;
389     int vy = 0;
390
391     if (leftPressed && !rightPressed) vx = -1;
392     if (rightPressed && !leftPressed) vx = 1;
393     if (upPressed && !downPressed) vy = -1;
394     if (downPressed && !upPressed) vy = 1;
395
396     p.setVelX(vx);
397     p.setVelY(vy);
398 }
399
400 // 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 25: 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 // BOSSES
98 //*****
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
131     try {
132         // Load the font file from the res folder
133         InputStream is = ResourceManager.class.getClassLoader()
134             .getResourceAsStream("res/PixelFont.ttf");
135
136         if (is != null) {
137             // Create the font object (default size is 1pt)
138             pixelFont = Font.createFont(Font.TRUETYPE_FONT, is);
139             System.out.println("Pixel Font loaded successfully!");
140         } else {
141             System.err.println("Error: PixelFont.ttf not found. Using
142             default font.");
143             pixelFont = new Font("Arial", Font.BOLD, 20); // Fallback
144         }
145     } catch (FontFormatException | IOException e) {
146         e.printStackTrace();
147         pixelFont = new Font("Arial", Font.BOLD, 20); // Fallback
148     }
149
150     System.out.println("All Resources loaded successfully!");
151 } catch (IOException e) {
152     System.err.println("Error: Could not load images.");
153     e.printStackTrace();
154 }
155
156 // 画像を安全に読み込むためのヘルパーメソッド
157 private static BufferedImage loadTexture(String path) throws IOException {
158     // クラスパスからリソースを探す
159     java.net.URL url = ResourceManager.class.getClassLoader().getResource(
160         path);
161
162     if (url == null) {
163         // もし getResource で見つからない場合（フォルダ構成の違いなど）、
164         // 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）
165         try {
166             return ImageIO.read(new File(path));
167         } catch (IOException ex) {
168             throw new IOException("Image not found: " + path);
169         }
170     }

```

```

167     }
168     return ImageIO.read(url);
169 }
170
171 // ダメージ演出用に、透明度を維持したまま「真っ白なシルエット」を作成するメソッド
172 private static BufferedImage createWhiteSilhouette(BufferedImage original)
173 {
174     // 元の画像と同じサイズで、空の画像を作成
175     BufferedImage whiteImg = new BufferedImage(
176         original.getWidth(),
177         original.getHeight(),
178         BufferedImage.TYPE_INT_ARGB
179     );
180
181     // すべてのピクセルを走査する
182     for (int x = 0; x < original.getWidth(); x++) {
183         for (int y = 0; y < original.getHeight(); y++) {
184             int p = original.getRGB(x, y);
185
186             // アルファ値（透明度）を取得
187             int a = (p >> 24) & 0xff;
188
189             // 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
190             if (a > 0) {
191                 // ARGB: アルファ値 + R(255) + G(255) + B(255)
192                 int whiteColor = (a << 24) | (255 << 16) | (255 << 8) |
193                 255;
194                 whiteImg.setRGB(x, y, whiteColor);
195             }
196         }
197     }
198     return whiteImg;
199 }
200
201 private static BufferedImage createBlackSilhouette(BufferedImage original)
202 {
203     BufferedImage blackImg = new BufferedImage(original.getWidth(),
204         original.getHeight(), BufferedImage.TYPE_INT_ARGB);
205     for (int x = 0; x < original.getWidth(); x++) {
206         for (int y = 0; y < original.getHeight(); y++) {

```

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

(文責：佐々木)