情報理工学域 メディア情報学プログラム
プログラミング演習 最終レポート

DX2 グループ 11 Gladiator

2411593 齋藤寛仁
2411603 佐々木晃誠
2411712 Puller Stefano Daniele

2026 年 2 月 18 日

1　概要説明

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

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

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

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

図 1: プレイ画面

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

(文責：佐々木)

2　設計方針

3　プログラムの説明

3.1　齋藤担当

3.2　佐々木担当

3.3　Puller 担当

4　実行例

5　考察

6　感想

6.1　齋藤

6.2　佐々木

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

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

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

(文責：佐々木)

6.3 Puller

7 付録 1：操作法マニュアル

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

図 2: タイトル画面

図 3: スタート画面

次に、プレイ中のキー操作について説明します。

表 1: プレイ中のキー操作

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

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

図 4: ゲームオーバー画面

(文責：佐々木)

8 付録２：プログラムリスト

Listing 1: Main.java

```java
package main;

import view.GamePanel;
import view.ResourceManager;

import javax.swing.*;

public class  Main {

    public static void main(String[] args) {
        // 1. Load resources BEFORE creating the window
        ResourceManager.loadImages();

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

Listing 2: GameModel.java

```java
package model;

import view.ResourceManager;

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

// --- GameModel (M) ---
public class GameModel {
    private ArrayList<GameObject> objects;
    private Player player;
    private boolean isGameOver = false;
```

```
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.
        PLAYER_WIDTH)/2, GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT -
        GameConstants.PLAYER_HEIGHT);
60          objects.add(player);
61
62          // Initialize Background
63          background = new Background();
64
65          isFiring = false;
66          shotTimer = 0;
67          arrowDamage = GameConstants.ARROW_DAMAGE;
68          arrowInterval = GameConstants.ARROW_INTERVAL;
69          state = GameState.PLAYING;
70          score = 0; //スコアをリセット
71
72          // 追加ライフ初期化:
73          lives = GameConstants.PLAYER_MAX_LIVES;
74          damageTimer = 0;
75
76          //Score progression resets
77          this.currentLevelIndex = 0;
78          this.nextTargetScore = GameConstants.LEVEL_MILESTONES[
        currentLevelIndex];
79          isBossActive = false;
80
81          // STAGE 1 SETUP: Add Harpy Spawner
82          activeSpawners.add(new EnemySpawner(Harpy.class, GameConstants.
        HARPY_SPAWN_INTERVAL, GameConstants.HARPY_SPAWN_VARIANCE));
83          this.currentStage = 1;
84
85
86          String tutorial = "WELCOME GLADIATOR!\n\n" +
87                  "Controls:\n" +
88                  "[KEY-ARROWS] Move\n" +
89                  "[SPACE] Shoot\n" +
90                  "[P] Pause\n\n" +
```

```
                "Defeat enemies\n" +
                "and survive.\n" +
                "Good Luck!";
        showMessage(tutorial);


    }

    public static void addScore(int points){
        score += points;
    }

    public void setFiring(boolean firing) {
        this.isFiring = firing;
    }

    public GameState getState() {
        return state;
    }

    public void setState(GameState s) {
        this.state = s;
    }

    public void update() {
        if (state != GameState.PLAYING) return;

        // Update background scrolling
        if(background!= null) {
            background.update();
        }

        checkLevelProgression();



        // --- 連射ロジック ---
        if (isFiring) {
            if (shotTimer == 0) {
                playerShoot();
```

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

```
171         for (GameObject obj : objects) {
172             obj.move();
173         }
174 
175         // 当たり判定
176         checkCollisions();
177 
178         // 削除
179         objects.removeIf(obj -> obj.isDead());
180     }
181 
182     /**
183      * Helper method to instantiate the correct enemy based on Class type.
184      */
185     private void spawnMinion(Class<? extends Minion> type) {
186         int x,y;
187         if (type == Harpy.class) {
188             x = rand.nextInt(GameConstants.WINDOW_WIDTH - GameConstants.
    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: " +
    currentLevelIndex);
223             nextTargetScore = GameConstants.LEVEL_MILESTONES[currentLevelIndex
    ];
224         }
225     }
226
227     private void applyLevelEffects(int levelIndex) {
228         switch (levelIndex) {
229             case 0: //START OF THE GAME
230                 System.out.println("Start of the Game");
231                 break;
232
233             case 1:
234                 System.out.println("Difficulty UP!");
235                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
    (0.8);
236                 break;
237
238             case 2:
239                 System.out.println("Difficulty UP!");
240                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
    (0.8);
241                 break;
242
```

```
243             case 3:
244                 showMessage("WARNING!\n\nBOSS DETECTED:\nAPOLLO\n\nPrepare for
     battle!");
245                 isBossActive = true;
246                 clearEverything();
247                 spawnApollo();
248                 healPlayer();
249                 break;
250             case 4:
251                 showMessage("STAGE 1 CLEARED!\n\nEntering the Heavens...\n\
    nArrow Damage doubled!\n\n Press [1] to use\nABILITY 1:\nAPOLLO'S SUN");
252                 if (background != null) {
253                     clearEverything();
254                     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                     // Cyclops Spawner
268                     activeSpawners.add(new EnemySpawner(Cyclops.class,
269                             GameConstants.CYCLOPS_SPAWN_INTERVAL,
270                             GameConstants.CYCLOPS_SPAWN_VARIANCE));
271
272                 }
273                 break;
274             case 5:
275                 System.out.println("Difficulty UP!");
276                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
    (0.9);
277                 break;
278             case 6:
```

```
279                 System.out.println("Difficulty UP!");
280                 for (EnemySpawner s : activeSpawners) s.increaseDifficulty
    (0.9);
281                 break;
282             case 7:
283                 showMessage("WARNING!\n\nBOSS DETECTED:\nZEUS\n\nPrepare for
    battle!");
284                 isBossActive = true;
285                 clearEverything();
286                 spawnZeus();
287                 healPlayer();
288                 break;
289             case 8:
290                 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                     // Cyclops Spawner
301                     activeSpawners.add(new EnemySpawner(Cyclops.class,
302                             GameConstants.CYCLOPS_SPAWN_INTERVAL,
303                             GameConstants.CYCLOPS_SPAWN_VARIANCE));
304                 }
305                 break;
306             case 9:
307
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
    instanceof Projectile);
328     }
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-
    GameConstants.ARROW_WIDTH)/2, player.getY() - GameConstants.ARROW_HEIGHT,
    arrowDamage);
335             newObjectsBuffer.add(a);
336         }
337     }
338
339     public void ability1(){
340         if (ability1Timer > 0) return;
341         ability1Timer = GameConstants.ABILITY1TIMER;
342         Sun sun = new Sun(
343                 player.getX() + GameConstants.PLAYER_WIDTH / 2,
344                 player.getY(),
345                 0, false, true
346         );          newObjectsBuffer.add(sun);
347         System.out.println("Player shoots sun");
348     }
349
```

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

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

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

```
468             return;
469         }
470 
471         // --- CASE 2: PROJECTILE vs LIVING ENTITY (Player or Enemy) ---
472         Projectile proj = null;
473         GameObject entity = null;
474 
475         // Identify which is which
476         if (a instanceof Projectile) { proj = (Projectile) a; entity = b; }
477         else if (b instanceof Projectile) { proj = (Projectile) b; entity = a;
    }
478 
479         if (proj != null) {
480             // Sub-case A: Projectile hits Player
481             if (entity instanceof Player) {
482                 if (proj.getAlignment() == Alignment.ENEMY) {
483                     playerTakesDamage();
484                     if (!proj.isPenetrating()) proj.setDead();
485                 }
486             }
487             // Sub-case B: Projectile hits Enemy (Harpy, Apollo, Golem, etc.)
488             else if (entity instanceof HostileEntity) {
489                 HostileEntity enemy = (HostileEntity) entity;
490 
491                 // Only damage if the projectile belongs to the Player
492                 if (proj.getAlignment() == Alignment.PLAYER) {
493                     enemy.takeDamage(proj.getDamage());
494 
495                     // SUN CASE: If the projectile is a Sun, it also loses HP
    upon contact
496                     if (proj instanceof BossProjectile) {
497                         ((BossProjectile) proj).reduceHealth(1);
498                     }
499 
500                     else if (!proj.isPenetrating()) {
501                         proj.setDead();
502                     }
503                 }
504             }
505             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      //HEIGTHS
8      public static final int HUD_HEIGHT = 50;
9      public static final int FIELD_HEIGHT = 800;
10     public static final int BOTTOM_HUD_HEIGHT = 100;
11     public static final int WINDOW_HEIGHT = HUD_HEIGHT + FIELD_HEIGHT +
       BOTTOM_HUD_HEIGHT;
12 
13     // ゲームの設定
14     public static final int FPS = 60;
15 
16     // 背景の設定
17     public static final double SCREEN_SPEED = 1.0;
18 
19     // の設定 PLAYER
20     public static final int PLAYER_WIDTH = 90;
21     public static final int PLAYER_HEIGHT = PLAYER_WIDTH*923/721; // set to
       the image height width ration
22     public static final int PLAYER_SPEED = 8;
23     public static final int PLAYER_MAX_LIVES = 3;
```

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

```
61      public static final int CYCLOPS_WIDTH = 150;
62      public static final int CYCLOPS_HEIGHT = CYCLOPS_WIDTH;
63      public static final double CYCLOPS_YSPEED = 2;
64      public static final int CYCLOPS_HP = 10;
65      public static final int CYCLOPS_SCORE_POINTS = 50;
66      public static final int CYCLOPS_SPAWN_INTERVAL = 400;
67      public static final int CYCLOPS_SPAWN_VARIANCE = CYCLOPS_SPAWN_INTERVAL /
        2;
68      public static final int CYCLOPS_MOVEMENT_TIMER = 20;
69      public static final int CYCLOPS_ATTACK_TIMER = 60;
70
71      // の設定 BOULDER
72      public static final int BOULDER_WIDTH = 100;
73      public static final int BOULDER_HEIGHT = BOULDER_WIDTH;
74      public static final double BOULDER_INITIAL_SPEED = 0;
75      public static final double BOULDER_GRAVITY = 0.3;
76      public static final int BOULDER_DAMAGE = 5;
77
78      // ************************************
79      // の設定 BOSS
80      // ************************************
81
82      // の設定 APOLLO
83      public static final int APOLLO_WIDTH = 200;
84      public static final int APOLLO_HEIGHT = APOLLO_WIDTH * 1556 / 2463;
85      public static final int APOLLO_SPEED1 = 4;
86      public static final int APOLLO_SPEED2 = APOLLO_SPEED1 * 3 / 2;
87      public static final int APOLLO_HP = 2; //50
88      public static final int APOLLO_SCORE_POINTS = 1000;
89
90      // の設定 SUN
91      public static final int SUN_WIDTH = 150;
92      public static final int SUN_HEIGHT = SUN_WIDTH;
93      public static final double SUN_SPEED1 = 6;
94      public static final double SUN_SPEED2 = SUN_SPEED1 * 3 / 2;
95      public static final int SUN_DAMAGE = 1;
96      public static final int SUN_HP = 20;
97
98      // の設定 ZEUS
99      public static final int ZEUS_SPEED = 4;
```

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

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

Listing 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;
```

```
        this.height = h;
        this.image = image;
    }

    public abstract void move();

    public abstract void draw(Graphics g);

    public Rectangle getBounds() {
        return new Rectangle(x, y, width, height);
    }

    public Shape getShape() {
        return new Rectangle2D.Float(x, y, width, height);
    }

    public boolean isDead() {
        return isDead;
    }

    public void setDead() {
        this.isDead = true;
    }

    public int getY() {
        return y;
    }

    public int getX() {
        return x;
    }
}
```

Listing 5: GameState.java

```
package model;

// ゲームの状態を定義
public enum GameState {
    TITLE, PLAYING, PAUSED, MESSAGE, GAMEOVER
```

```
}
```

Listing 6: Alignment.java

```
package model;

/**
 * Alignment Enum
 * Defines which "team" a game object belongs to.
 * Used to prevent friendly fire and determine collision logic.
 */
public enum Alignment {
    PLAYER, // Belongs to the Player (Targets Enemies)
    ENEMY  // Belongs to Enemies (Targets Player)
}
```

Listing 7: Background.java

```
package model;

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

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

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

    public Background() {
```

```
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              // 1. Draw Normal Tile at current Y
60              // Covers: y to y+400  x
```

```
            drawForceSize(g, currentY, false);

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

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

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

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

    private boolean isStage2Img(){
        return (this.image == ResourceManager.stage2Img);
```

```
    }
}
```

Listing 8: Projectile.java

```
package model;

import view.ResourceManager;

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

/**
 * Projectile Class
 * Abstract base class for all flying objects (Arrows, Feathers, Suns, Boulders).
 * It introduces the concept of "Power Level" to handle projectile-vs-projectile collisions.
 */
public abstract class Projectile extends GameObject {

    protected Alignment alignment;

    // Power Level determines priority when two projectiles collide:
    // 0 = Ephemeral (Destroyed by anything)
    // 1 = Light (Arrow, Feather)
    // 2 = Heavy (Boulder - Destroys Light)
    // 3 = Ultimate (Sun - Destroys Heavy and Light)
    protected int powerLevel;

    protected int damage;

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

    public Projectile(int x, int y, int w, int h, BufferedImage image, Alignment alignment, int powerLevel, int damage) {
        super(x, y, w, h, image);
        this.alignment = alignment;
```

```
        this.powerLevel = powerLevel;
        this.damage = damage;
        this.isPenetrating = false; // Default: destroys itself on impact
    }

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

Listing 9: Player.java

```
package model;

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

// --- クラス Player 自分 () ---
public class Player extends GameObject {
    private int velX = 0; // 横の移動速度
    private int velY = 0; // 縦の移動速度
    private int speed = GameConstants.PLAYER_SPEED;
    private int level = 1;

    public Player(int x, int y) {
        super(x, y, GameConstants.PLAYER_WIDTH, GameConstants.PLAYER_HEIGHT,
    ResourceManager.playerImg); // 40の四角
    x40
    }

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

        // 画面からはみ出さないように制限
        if (x < 0) x = 0;
```

```
26          if (x > GameConstants.WINDOW_WIDTH - width) x = GameConstants.
        WINDOW_WIDTH - width;
27          if (y < GameConstants.HUD_HEIGHT) y = GameConstants.HUD_HEIGHT;
28          if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
        ) y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height;
29      }
30
31      @Override
32      public void draw(Graphics g) {
33          if(image != null){
34              g.drawImage(image, x, y, width, height, null);
35          }
36          else{
37              // Fallback if image failed to load
38              g.setColor(Color.BLUE);
39              g.fillRect(x, y, width, height);
40          }
41      }
42
43      @Override
44      public Shape getShape() {
45          // we define the margin
46          float paddingX = width * 0.3f;  // remove 20 % width
47          float paddingY = height * 0.2f; // remove 10 % height
48
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
        Sun (Lvl 3)
12          // Damage: Standard Arrow Damage
13          super(x, y, GameConstants.ARROW_WIDTH, GameConstants.ARROW_HEIGHT,
        ResourceManager.arrowImg,
14                  Alignment.PLAYER, 1, arrowDamage);
15      }
16
17      @Override
18      public void move() {
19          y -= GameConstants.ARROW_SPEED;
20          if (y < -height) {
21              isDead = true;
22          }
23      }
24
25      @Override
26      public void draw(Graphics g) {
27          if (ResourceManager.arrowImg != null) {
28              g.drawImage(image, x, y, width, height, null);
```

```
        } else {
            g.setColor(Color.YELLOW);
            g.fillRect(x, y, width, height);
        }
    }
}
```

Listing 11: BossProjectile.java

```
package model;

import java.awt.image.BufferedImage;

public abstract class BossProjectile extends Projectile {

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

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

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

Listing 12: Boss.java

```
package model;

import java.awt.*;
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
       scorePoints, GameModel model) {
9          super(x, y, w, h, image, hp, scorePoints, model);
10     }
11
12     @Override
13     public void takeDamage(int dmg) {
14         super.takeDamage(dmg);
15
16         if (this.isDead){
17             model.bossDefeated();
18         }
19     }
20 }
```

Listing 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 "
       Rage Mode"
17
18     public Apollo(GameModel model) {
19         // Call the parent constructor (Boss -> HostileEntity)
```

```
20          // Parameters: x, y, width, height, HP, Score Points
21          super(
22                  (GameConstants.WINDOW_WIDTH - GameConstants.APOLLO_WIDTH) / 2,
     // Start in the middle
23                  GameConstants.HUD_HEIGHT, // Start below HUD
24                  GameConstants.APOLLO_WIDTH,
25                  GameConstants.APOLLO_HEIGHT,
26                  ResourceManager.apolloImg,
27                  GameConstants.APOLLO_HP,
28                  GameConstants.APOLLO_SCORE_POINTS, // Score awarded when
    defeated
29                  model
30          );
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
     Red/Fast
46              shootSun();
47          }
48      }
49
50      private void shootSun(){
51          Sun s = new Sun(x, y, speedX, secondPhase, false);
52          model.spawnEnemyProjectile(s);
53      }
54
55      @Override
56      public void takeDamage(int dmg) {
```

```
57          // 1. Apply damage using the parent class logic (reduces HP, checks
        death, adds score)
58          super.takeDamage(dmg);
59
60          // 2. Specific Logic for Apollo: Check for Second Phase (Rage Mode)
61          // If HP drops below 50% and we are not yet in the second phase...
62          if (hp <= maxHp / 2 && !secondPhase) {
63              image = ResourceManager.apolloImg2; // Change sprite to Red Apollo
64              speedX = (speedX > 0) ? GameConstants.APOLLO_SPEED2 : -
        GameConstants.APOLLO_SPEED2; // Double the movement speed
65              secondPhase = true; // Activate the flag
66              System.out.println("Apollo entering Phase 2!");
67          }
68      }
69
70      @Override
71      public void draw(Graphics g) {
72          BufferedImage imgToDraw = (flashTimer > 0) ? ResourceManager.
        apolloHitImg : image;
73
74          if (image != null) {
75              // Directional Flipping Logic
76              if (speedX > 0) {
77                  // Moving RIGHT: Draw normally
78                  g.drawImage(imgToDraw, x, y, width, height, null);
79              } else {
80                  // Moving LEFT: Flip the image horizontally
81                  // We draw starting at (x + width) and use a negative width to
         flip
82                  g.drawImage(imgToDraw, x + width, y, -width, height, null);
83              }
84          } else {
85              // Fallback: Draw an Orange rectangle if images fail to load
86              g.setColor(Color.ORANGE);
87              g.fillRect(x, y, width, height);
88          }
89      }
90  }
```

Listing 14: Sun.java

```java
package model;

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

public class Sun extends BossProjectile {

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

    private double initialSize;

    public Sun(int summonerX, int summonerY, int ApolloSpeedX, boolean
    isSecondPhase, boolean friendly) {
        // Call parent constructor
        // HP = 1 (doesn't matter), Score = 0
        super(0,
                0,
                GameConstants.SUN_WIDTH,
                GameConstants.SUN_HEIGHT,
                isSecondPhase ? ResourceManager.sunImg2 : ResourceManager.
    sunImg,
                friendly ? Alignment.PLAYER : Alignment.ENEMY,
                3,
                GameConstants.SUN_DAMAGE);

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

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

        this.initialSize = GameConstants.SUN_WIDTH;

        velX = 1;
```

```
38          velY = 1;
39          double currentSpeed;
40          double angleRadians;
41
42          if (isSecondPhase) {
43              currentSpeed = GameConstants.SUN_SPEED2;
44          } else {
45              currentSpeed = GameConstants.SUN_SPEED1;
46          }
47
48          if (!friendly){
49              // sun comes from apollo
50              preciseX = (ApolloSpeedX > 0) ? (summonerX + GameConstants.
    APOLLO_WIDTH + width / 2) : (summonerX - width / 2);
51              preciseY = summonerY + height;
52              x = (int) preciseX;
53              y = (int) preciseY;
54              // --- Phase Logic ---
55              // --- Trajectory Calculation ---
56              double minAngle = 20.0;
57              double maxAngle = 70;
58              angleRadians = Math.toRadians(minAngle + Math.random() * (maxAngle
     - minAngle));
59          } else {
60              // sun comes from ability
61              preciseX = summonerX;
62              preciseY = summonerY - height / 2;
63
64              // 2. right boarder check
65              if (preciseX > GameConstants.WINDOW_WIDTH - width / 2) {
66                  preciseX = GameConstants.WINDOW_WIDTH - width / 2;
67              }
68
69              // 3. left boarder check
70              if (preciseX < width / 2) {
71                  preciseX = width / 2;
72              }
73
74              x = (int) preciseX;
75              y = (int) preciseY;
```

```
            angleRadians = Math.toRadians(360-30);
            velX = (Math.random() < 0.5) ? 1 : -1;
        }

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

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

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

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

        double currentRadius = getCurrentSize() / 2.0;

        // Wall Bounce
        if (x - currentRadius < 0) {
            x = (int)currentRadius;
            preciseX = x;
            velX = -velX;
        }
        if (x + currentRadius > GameConstants.WINDOW_WIDTH) {
            x = (int)(GameConstants.WINDOW_WIDTH - currentRadius);
            preciseX = x;
            velX = -velX;
        }

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

```
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

```java
package model;

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

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

    public Zeus (GameModel model) {
        super ( (GameConstants.WINDOW_HEIGHT - GameConstants.ZEUS_WIDTH) / 2,
                GameConstants.HUD_HEIGHT,
                GameConstants.ZEUS_WIDTH,
                GameConstants.ZUES_HEIGHT,
                ResourceManager.zeusImg,
                GameConstants.ZEUS_HP,
                GameConstants.ZEUS_SCORE_POINTS,
                model
        );
        maxShootTimer = GameConstants.ZEUS_SHOOT_TIMER;
        resetShootTimer();
        maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE;
        setAbility1Timer();
        ability1Position = 1;
        hitImg = ResourceManager.zeusHitImg;
        ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
```

```
        ability2Started = false;
    }

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

                // Bounce logic: If it hits the screen edges
                if (x <= 0 ) {
                    x = 0;
                    speedX = -speedX; // Reverse direction
                    ability1Timer--;
                } else if (x >= GameConstants.WINDOW_WIDTH - width) {
                    x = GameConstants.WINDOW_WIDTH - width;
                    speedX = -speedX; // Reverse direction
                    ability1Timer--;
                }
                if(shootTimer <= 0){
                    shootLighting();
                    resetShootTimer();
                }
                shootTimer--;
            } else if (ability1Pause <= 0){
                ability1();
            } else {
                ability1Pause--;
            }
        } else {
            if (!ability2Started) {
                if (Math.random() < 0.5) {
                    x = 0;
                    speedX = GameConstants.ZEUS_SPEED2;
                } else {
                    x = GameConstants.WINDOW_WIDTH - width;
                    speedX = -GameConstants.ZEUS_SPEED2;
                }
```

```
                Random random = new Random();
                ability2Repetitions = random.nextInt(3) + 1;
                ability2Started = true;
            }
            x += speedX;

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

            if (x <= 0 ) {
                x = 0;
                speedX = -speedX; // Reverse direction
                ability2Repetitions--;
            } else if (x >= GameConstants.WINDOW_WIDTH - width) {
                x = GameConstants.WINDOW_WIDTH - width;
                speedX = -speedX; // Reverse direction
                ability2Repetitions--;
            }
            if (ability2Repetitions == 0){
                ability2Timer = GameConstants.ZEUS_ABILITY2_TIMER;
                ability2Started = false;
                resetShootTimer();
            }

        }

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

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

```

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

```
156             maxAbility1Pause = GameConstants.ZEUS_ABILITY1_PAUSE2;
157         }
158     }
159 
160     private void setAbility1Timer(){
161         Random random = new Random();
162         ability1Timer = random.nextInt(4) + 1;;
163         ability1Phase = false;
164     }
165 
166     private void resetShootTimer(){
167         shootTimer = maxShootTimer;
168     }
169 
170     @Override
171     public void draw(Graphics g) {
172         BufferedImage imgToDraw = (flashTimer > 0) ? hitImg : image;
173 
174         if (image != null) {
175             if (speedX > 0) {
176                 g.drawImage(imgToDraw, x, y, width, height, null);
177             } else {
178                 g.drawImage(imgToDraw, x + width, y, -width, height, null);
179             }
180         } else {
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,
        boolean friendly, boolean ability2Started) {
11          super(0,
12                  0,
13                  GameConstants.LIGHTING_WIDTH,
14                  GameConstants.LIGHTING_HEIGHT,
15                  isSecondPhase ? ResourceManager.lightingImg2 : ResourceManager
        .lightingImg,
16                  friendly ? Alignment.PLAYER : Alignment.ENEMY,
17                  3,
18                  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 :
         x + width;
41              } else {
42                  this.x = (ZeusSpeedX > 0) ? x : x + GameConstants.ZEUS_WIDTH -
         width;
43              }
```

```
            this.y = y + height;
        }
    }

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

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

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

    @Override
    public Shape getShape() {
        return new Ellipse2D.Float(x, y, width, height);
    }
}
```

Listing 17: HostileEntity.java

```
package model;

import java.awt.image.BufferedImage;

/**
 * HostileEntity
 * Abstract class representing any entity that is hostile to the player.
 * Handles HP, damage logic, invincibility, and score points.
```

```
 */
public abstract class HostileEntity extends GameObject {

    protected int hp;
    protected int maxHp;
    protected int flashTimer = 0; // For hit effect
    protected boolean isInvincible = false; // If true, entity takes no damage
    protected int scorePoints; // Points awarded when destroyed
    protected GameModel model;

    public HostileEntity(int x, int y, int w, int h, BufferedImage image, int
    hp, int scorePoints, GameModel model) {
        super(x, y, w, h, image);
        this.hp = hp;
        this.maxHp = hp;
        this.scorePoints = scorePoints;
        this.model = model;
    }

    /**
     * Handles taking damage from player projectiles.
     * @param dmg Amount of damage to take
     */
    public void takeDamage(int dmg) {
        // If dead or invincible, do nothing
        if (isDead || isInvincible) return;

        this.hp -= dmg;
        this.flashTimer = GameConstants.FLASH_TIMER; // Set flash effect

        // Check for death
        if (this.hp <= 0) {
            this.isDead = true;
            GameModel.addScore(this.scorePoints); // Award points
        }
    }

    @Override
    public void move() {
        if (flashTimer > 0) {
```

```
            flashTimer--;
        }
    }

    public int getFlashTimer() {
        return flashTimer;
    }
}
```

Listing 18: EnemySpawner.java

```
package model;

import java.util.Random;

/**
 * EnemySpawner Class
 * Manages the spawn timing for a specific type of enemy.
 * Allows independent spawn rates for different enemies (e.g., Harpies vs
    Golems).
 */
public class EnemySpawner {

    private Class<? extends Minion> enemyType; // The class of the enemy to
    spawn
    private int baseInterval;   // Frames between spawns (average)
    private int variance;       // Random variation in frames
    private int timer;          // Current countdown timer
    private Random rand = new Random();

    public EnemySpawner(Class<? extends Minion> enemyType, int baseInterval,
    int variance) {
        this.enemyType = enemyType;
        this.baseInterval = baseInterval;
        this.variance = variance;
        resetTimer(); // Start the timer immediately
    }

    /**
     * Updates the timer.
```

```
     * @return true if it's time to spawn an enemy, false otherwise.
     */
    public boolean update() {
        timer--;
        if (timer <= 0) {
            resetTimer();
            return true;
        }
        return false;
    }

    private void resetTimer() {
        // Calculate next spawn time: base +/- random variance
        int var = (variance > 0) ? rand.nextInt(variance * 2 + 1) - variance :
     0;
        this.timer = baseInterval + var;
        if (this.timer < 30) this.timer = 30; // Minimum safety limit (0.5 sec
    )
    }

    /**
     * Increases difficulty by reducing the spawn interval.
     * @param multiplier Factor to multiply interval (e.g., 0.8 for 20% faster
    ).
     */
    public void increaseDifficulty(double multiplier) {
        this.baseInterval = (int)(this.baseInterval * multiplier);
        this.variance = (int)(this.variance * multiplier);
        // Prevent it from becoming too fast (e.g., limit to 60 frames)
        if (this.baseInterval < 60) this.baseInterval = 60;
    }

    public Class<? extends Minion> getEnemyType() {
        return enemyType;
    }
}
```

Listing 19: Boulder.java

```
package model;
```

```java

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

/**
 * Boulder Class
 * A heavy projectile dropped by the StoneGolem.
 * It has Power Level 2, meaning it destroys Arrows/Feathers but is destroyed by the Sun.
 */
public class Boulder extends Projectile {

    private double preciseY;
    private double velY;
    private double gravity = GameConstants.BOULDER_GRAVITY; // Accelerates downwards

    public Boulder(int x, int y) {
        super(x, y,
                GameConstants.BOULDER_WIDTH,
                GameConstants.BOULDER_HEIGHT,
                ResourceManager.boulderImg,
                Alignment.ENEMY,
                2,
                GameConstants.BOULDER_DAMAGE
        );

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

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

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

Listing 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 /
     2;
52                  }
```

```
53              }
54              this.image = wingsClosed ? ResourceManager.cyclopsImg2 :
    ResourceManager.cyclopsImg;
55          }
56          preciseY += velY;
57          y = (int) preciseY;
58
59          if(y > (GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT/2 -
    height)){
60              reachedMidScreen = true;
61          }
62
63          // CHECK IF THERE IS BOULDER OR IF IT IS DEAD
64          if (myBoulder == null || myBoulder.isDead()) {
65              // REDUCE TIMER
66              if(attackTimer > 0){
67                  attackTimer--;
68              }
69              // ONCE THE TIMER IS 0 SPAWN BOULDER AND RESET TIMER
70              if (attackTimer == 0){
71                  throwBoulder();
72                  attackTimer = GameConstants.CYCLOPS_ATTACK_TIMER;
73              }
74          }
75          movementTimer--;
76      }
77
78      @Override
79      public void draw(Graphics g) {
80          BufferedImage imgToDraw;
81
82          if (image == ResourceManager.cyclopsImg){
83              imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg :
    image;
84          } else {
85              imgToDraw = (flashTimer > 0) ? ResourceManager.cyclopsHitImg2 :
    image;
86          }
87
88          if (image != null) {
```

```
            g.drawImage(imgToDraw, x, y, width, height, null);
        } else {
            g.setColor(Color.RED);
            g.fillRect(x, y, width, height);
        }
    }

    private void throwBoulder(){
        Boulder b = new Boulder (x + (width - GameConstants.BOULDER_WIDTH)/2,
    y + height);
        this.myBoulder = b;
        model.spawnEnemyProjectile(b);
    }
}
```

Listing 21: Feather.java

```
package model;

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

public class Feather extends Projectile {

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

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

```
    }

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

Listing 22: Harpy.java

```
package model;

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

public class Harpy extends Minion {

    private int velX;
    private int velY;
    private int fireTimer;
    private boolean isInScreen;

    public Harpy(int x, int y, GameModel model) {
        // Pass params to parent: x, y, width, height, HP, Score Points
        super(x, y,
                GameConstants.HARPY_WIDTH,
                GameConstants.HARPY_HEIGHT,
                ResourceManager.harpyImg,
                GameConstants.HARPY_HP,
                GameConstants.HARPY_SCORE_POINTS,
                model);

        // --- MOVEMENT SETUP ---
        this.velY = GameConstants.HARPY_YSPEED;
```

```
26          this.velX = GameConstants.HARPY_XSPEED;
27          this.isInScreen = false;
28
29          // Randomize direction
30          if (Math.random() < 0.5) {
31              this.velX = -this.velX;
32          }
33          resetFireTimer();
34      }
35
36      @Override
37      public void move() {
38          super.move();
39          x += velX;
40          y += velY;
41
42          // Bounce Logic
43          if (x < 0) {
44              x = 0;
45              velX = -velX;
46          }
47          if (x > GameConstants.WINDOW_WIDTH - width) {
48              x = GameConstants.WINDOW_WIDTH - width;
49              velX = -velX;
50          }
51          if (y < GameConstants.HUD_HEIGHT && isInScreen) {
52              y = GameConstants.HUD_HEIGHT;
53              velY = -velY;
54          }
55          if (y > GameConstants.HUD_HEIGHT) {
56              isInScreen = true;
57          }
58          if (y > GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
    ) {
59              y = GameConstants.FIELD_HEIGHT + GameConstants.HUD_HEIGHT - height
    ;
60              velY = -velY;
61          }
62
63          if(fireTimer > 0) {
```

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

```
}
```

Listing 23: Minion.java

```
package model;

import java.awt.image.BufferedImage;

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

Listing 24: GamePanel.java

```
package view;

import model.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.*;
import java.awt.image.BufferedImage;

// --- Main Class (View and Controller simplified) ---
public class GamePanel extends JPanel implements KeyListener {
    private GameModel model;
    private Timer timer;

    // Variables to track button states
    private boolean leftPressed = false;
    private boolean rightPressed = false;
    private boolean upPressed = false;
```

```
19      private boolean downPressed = false;
20
21      private long startTime; // Game start time
22      private long endTime;   // Game end time
23
24      public GamePanel() {
25          model = new GameModel(); // Initial state is TITLE
26
27          // Update preferred size to include the new BOTTOM_HUD_HEIGHT
28          this.setPreferredSize(new Dimension(GameConstants.WINDOW_WIDTH,
    GameConstants.WINDOW_HEIGHT));
29          this.setBackground(Color.BLACK);
30          this.setFocusable(true);
31          this.addKeyListener(this);
32
33          // Game Loop Timer
34          timer = new Timer(1000/GameConstants.FPS, e -> {
35              model.update();
36              repaint();
37          });
38          timer.start();
39      }
40
41      @Override
42      protected void paintComponent(Graphics g) {
43          super.paintComponent(g);
44
45          // Switch drawing based on game state
46          GameState state = model.getState();
47
48          if (state == GameState.TITLE) {
49              drawTitleScreen(g);
50          } else if (state == GameState.PLAYING || state == GameState.PAUSED ||
    state == GameState.MESSAGE) {
51              drawGameScreen(g);
52              if (state == GameState.PAUSED){
53                  drawPauseScreen(g);
54              }
55              else if (state == GameState.MESSAGE) {
56                  drawMessageScreen(g);
```

```
57              }
58
59          } else if (state == GameState.GAMEOVER) {
60              drawGameScreen(g); // Draw game screen in background
61              drawGameOverScreen(g);
62          }
63      }
64
65      // Helper method to set font easily
66      private void setPixelFont(Graphics g, float size) {
67          if (ResourceManager.pixelFont != null) {
68              g.setFont(ResourceManager.pixelFont.deriveFont(size));
69          } else {
70              g.setFont(new Font("Arial", Font.BOLD, (int)size));
71          }
72      }
73
74      // Main Game Drawing Method
75      private void drawGameScreen(Graphics g) {
76
77          // 1. Draw Background
78          if (model.getBackground() != null) {
79              model.getBackground().draw(g);
80          } else {
81              // Fallback: Black background if image is missing
82              g.setColor(Color.BLACK);
83              // Fill only the game field area
84              g.fillRect(0, GameConstants.HUD_HEIGHT, GameConstants.WINDOW_WIDTH
    , GameConstants.FIELD_HEIGHT);
85          }
86
87          // 2. Draw Game Objects (Player, Enemies, Projectiles)
88          for (GameObject obj : model.getObjects()) {
89              // Invincibility flashing logic for Player
90              if (obj instanceof Player && model.isInvincible()) {
91                  // Toggle visibility every 100ms
92                  if (System.currentTimeMillis() % 200 < 100) {
93                      continue;
94                  }
95              }
```

```
            obj.draw(g);
        }

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

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

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

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

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

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

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

        // C. Hearts / Lives (Right aligned)
        int maxLives = GameConstants.PLAYER_MAX_LIVES;
        int currentLives = model.getLives();
        int heartSize = 32;
        int spacing = 8;
```

```
133         int startX = GameConstants.WINDOW_WIDTH - 20 - (maxLives * (heartSize
    + spacing));
134         int heartY = (GameConstants.HUD_HEIGHT - heartSize) / 2;
135 
136         for (int i = 0; i < maxLives; i++) {
137             // Determine which icon to draw (Full or Empty)
138             BufferedImage icon = (i < currentLives) ? ResourceManager.
    heartFullImg : ResourceManager.heartEmptyImg;
139 
140             if (icon != null) {
141                 g.drawImage(icon, startX + (i * (heartSize + spacing)), heartY
    , heartSize, heartSize, null);
142             } else {
143                 // Fallback drawing if images are missing
144                 g.setColor(i < currentLives ? Color.RED : Color.GRAY);
145                 g.fillOval(startX + (i * (heartSize + spacing)), heartY,
    heartSize, heartSize);
146             }
147         }
148     }
149 
150     // Helper method to draw the Bottom HUD (Ability Slots)
151     // Helper method to draw the Bottom HUD (Ability Slots)
152     private void drawBottomHUD(Graphics g) {
153         // Calculate start Y position (below the game field)
154         int startY = GameConstants.HUD_HEIGHT + GameConstants.FIELD_HEIGHT;
155         int height = GameConstants.BOTTOM_HUD_HEIGHT;
156 
157         // 1. Background Bar
158         g.setColor(new Color(50, 50, 80));
159         g.fillRect(0, startY, GameConstants.WINDOW_WIDTH, height);
160 
161         // 2. Separator Line
162         g.setColor(Color.WHITE);
163         g.drawLine(0, startY, GameConstants.WINDOW_WIDTH, startY);
164 
165         // 3. Draw Slots
166         int slotSize = 60;
167         int gap = 40;
168         int totalWidth = (3 * slotSize) + (2 * gap);
```

```
169         int startX = (GameConstants.WINDOW_WIDTH - totalWidth) / 2;
170         int slotY = startY + (height - slotSize) / 2 - 9;
171 
172         setPixelFont(g, 14f);
173 
174         for (int i = 0; i < 3; i++) {
175             int x = startX + (i * (slotSize + gap));
176 
177             // A. Draw Slot Background (Black)
178             g.setColor(Color.BLACK);
179             g.fillRect(x, slotY, slotSize, slotSize);
180 
181             // Check if this is the first slot (Index 0) AND if Ability 1 is
    unlocked
182             if (i == 0 && model.isAbilityUnclocked(1)) {
183 
184                 // 1. Draw the Icon (The Sun)
185                 if (ResourceManager.sunImg != null) {
186                     g.drawImage(ResourceManager.sunImg, x, slotY, slotSize,
    slotSize, null);
187                 }
188 
189                 // 2. Draw Cooldown Overlay (The fading effect)
190                 int timer = model.getAbilityNthTimer(1);
191                 int maxTime = GameConstants.ABILITY1TIMER; // Make sure this
    is set correctly in Constants!
192 
193                 if (timer > 0) {
194                     // Calculate percentage of time remaining (0.0 to 1.0)
195                     float ratio = (float) timer / maxTime;
196 
197                     // Calculate height of the dark overlay based on the ratio
198                     // If ratio is 1.0 (just used), height is full (60).
199                     // If ratio is 0.5, height is half (30), covering the top
    half.
200                     // This creates the effect of the color "filling up from
    bottom".
201                     int overlayHeight = (int) (slotSize * ratio);
202 
203                     // Set color to semi-transparent black
```

```
                g.setColor(new Color(0, 0, 0, 180)); // 180 is the alpha (transparency)

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

                // Optional: Draw the text timer on top if you want
                 g.setColor(Color.WHITE);
                String keyNum = String.valueOf(timer/60 + 1);
                int numWidth = g.getFontMetrics().stringWidth(keyNum);
                g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY + 37);
            }
        }

        // Check if this is the first slot (Index 0) AND if Ability 1 is unlocked
        if (i == 1 && model.isAbilityUnclocked(2)) {

            // 1. Draw the Icon (The Sun)
            if (ResourceManager.lightingImg != null) {
                g.drawImage(ResourceManager.lightingImg, x, slotY, slotSize, slotSize, null);
            }

            // 2. Draw Cooldown Overlay (The fading effect)
            int timer = model.getAbilityNthTimer(2);
            int maxTime = GameConstants.ABILITY2TIMER; // Make sure this is set correctly in Constants!

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

                // Calculate height of the dark overlay based on the ratio
                // If ratio is 1.0 (just used), height is full (60).
                // If ratio is 0.5, height is half (30), covering the top half.
                // This creates the effect of the color "filling up from bottom".
```

```
                int overlayHeight = (int) (slotSize * ratio);

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

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

                // Optional: Draw the text timer on top if you want
                g.setColor(Color.WHITE);
                String keyNum = String.valueOf(timer/60 + 1);
                int numWidth = g.getFontMetrics().stringWidth(keyNum);
                g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY + 37);
            }
        }
        // ----------------------------------------

        // B. Draw Slot Border
        g.setColor(Color.GRAY);
        g.drawRect(x, slotY, slotSize, slotSize);

        // C. Draw Key Number (1, 2, 3)
        g.setColor(Color.WHITE);
        String keyNum = String.valueOf(i + 1);
        int numWidth = g.getFontMetrics().stringWidth(keyNum);
        g.drawString(keyNum, x + (slotSize - numWidth) / 2, slotY + slotSize + 25);
    }
}

// Draw Title Screen
private void drawTitleScreen(Graphics g) {
    g.setColor(Color.WHITE);

    // Title: Big Pixel Font
    setPixelFont(g, 28f);
    String title = "GLADIATOR GAME"; // Cambia il nome se vuoi
    int titleWidth = g.getFontMetrics().stringWidth(title);
```

```
274         g.drawString(title, (GameConstants.WINDOW_WIDTH - titleWidth)/2, 250);
275
276         // Subtitle: Smaller
277         setPixelFont(g, 15f);
278         String msg = "Press SPACE to Start";
279         int msgWidth = g.getFontMetrics().stringWidth(msg);
280         g.drawString(msg, (GameConstants.WINDOW_WIDTH - msgWidth)/2, 350);
281     }
282
283     private void drawPauseScreen(Graphics g) {
284         // 1. Semi-transparent black overlay
285         g.setColor(new Color(0, 0, 0, 150)); // 150 = Alpha (Transparency)
286         g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
    WINDOW_HEIGHT);
287
288         // 2. "PAUSE" Text
289         g.setColor(Color.WHITE);
290         setPixelFont(g, 40f); // Large font
291         String pauseText = "PAUSE";
292         int pauseWidth = g.getFontMetrics().stringWidth(pauseText);
293         // Center text
294         g.drawString(pauseText, (GameConstants.WINDOW_WIDTH - pauseWidth) / 2,
     GameConstants.WINDOW_HEIGHT / 2 - 100);
295
296         // 3. Instruction Text
297         setPixelFont(g, 20f); // Smaller font
298         String resumeText = "Press [P] to Resume";
299         int resumeWidth = g.getFontMetrics().stringWidth(resumeText);
300         g.drawString(resumeText, (GameConstants.WINDOW_WIDTH - resumeWidth) /
    2, GameConstants.WINDOW_HEIGHT / 2 - 50);
301     }
302
303     private void drawMessageScreen(Graphics g) {
304         // 1. Semi-transparent black background for the whole screen (dimming)
305         g.setColor(new Color(0, 0, 0, 100));
306         g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
    WINDOW_HEIGHT);
307
308         // 2. The Message Box Dimensions
309         int boxWidth = 500;
```

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

```
349         int promptWidth = g.getFontMetrics().stringWidth(prompt);
350         g.drawString(prompt, (GameConstants.WINDOW_WIDTH - promptWidth) / 2,
    boxY + boxHeight - 20);
351     }
352
353     // Draw Game Over Screen
354     private void drawGameOverScreen(Graphics g) {
355         // Semi-transparent overlay
356         g.setColor(new Color(0, 0, 0, 150));
357         g.fillRect(0, 0, GameConstants.WINDOW_WIDTH, GameConstants.
    WINDOW_HEIGHT);
358
359         // Game Over Text
360         g.setColor(Color.RED);
361         setPixelFont(g, 35f); // Big Red Text
362         String GO = "GAME OVER";
363         int goWidth = g.getFontMetrics().stringWidth(GO);
364         g.drawString(GO, (GameConstants.WINDOW_WIDTH - goWidth)/2, 250);
365
366         g.setColor(Color.WHITE);
367         setPixelFont(g, 20f);
368
369         String scoreMsg = "Final Score: " + model.getScore();
370         int scoreWidth = g.getFontMetrics().stringWidth(scoreMsg);
371         g.drawString(scoreMsg, (GameConstants.WINDOW_WIDTH - scoreWidth)/2,
    320);
372
373         // ... Time e Quit/Continue (usa la stessa logica per centrare) ...
374         String cont = "[C] Continue";
375         String quit = "[Q] Quit";
376
377         g.drawString(cont, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
    stringWidth(cont))/2, 400);
378         g.drawString(quit, (GameConstants.WINDOW_WIDTH - g.getFontMetrics().
    stringWidth(quit))/2, 440);
379     }
380
381     // Update Player Velocity based on key states
382     private void updatePlayerVelocity() {
383         Player p = model.getPlayer();
```

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

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

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

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

Listing 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      // BOSSES
37      //******************
38
39      // APOLLO
40      public static BufferedImage apolloImg;
41      public static BufferedImage apolloImg2;
42      public static BufferedImage apolloHitImg;
43      // APOLLO's SUN
44      public static BufferedImage sunImg;
45      public static BufferedImage sunImg2;
46      // ZEUS
47      public static BufferedImage zeusImg;
48      public static BufferedImage zeusImg2;
49      public static BufferedImage zeusHitImg;
50      public static BufferedImage zeusHitImg2;
51      // ZEUS's LIGHTNING
52      public static BufferedImage lightingImg;
53      public static BufferedImage lightingImg2;
54
55      //******************
56      // HUD
57      //******************
58
59      // STAGES
60      public static BufferedImage stage1Img;
61      public static BufferedImage stage2Img;
62      public static BufferedImage stage3Img;
63      // HEART
64      public static BufferedImage heartFullImg;
65      public static BufferedImage heartEmptyImg;
66
67      // PIXEL FONT
68      public static Font pixelFont;
69
70      /**
71       * "resフォルダからすべてのリソースを読み込む"
72       */
73      public static void loadImages() {
74          try {
75              System.out.println("Loading resources...");
```

```
76
77              // PLAYER
78              playerImg = loadTexture("res/player.png");
79              arrowImg   = loadTexture("res/arrow.png");
80
81              //******************
82              // MINIONS
83              //******************
84              // HARPY
85              harpyImg = loadTexture("res/enemy.png");
86              harpyHitImg = createWhiteSilhouette(harpyImg);
87              // HARPY's FEATHER
88              featherImg = loadTexture("res/feather.png");
89              // CYCLOPS
90              cyclopsImg = loadTexture("res/cyclops_openedwings.png");
91              cyclopsImg2 = loadTexture("res/cyclops_closedwings.png");
92              cyclopsHitImg = createWhiteSilhouette(cyclopsImg);
93              cyclopsHitImg2 = createWhiteSilhouette(cyclopsImg2);
94              // CYCLOPS's BOULDER
95              boulderImg = loadTexture("res/boulder.png");
96
97              //******************
98              // BOSSES
99              //******************
100
101             // APOLLO
102             apolloImg = loadTexture("res/Apollo.png");
103             apolloImg2 = loadTexture("res/ApolloRed.png");
104             apolloHitImg = createWhiteSilhouette(apolloImg);
105             // APOLLO's SUN
106             sunImg = loadTexture("res/sun.png");
107             sunImg2 = loadTexture("res/sunRed.png");
108             // ZEUS
109             zeusImg = loadTexture("res/Zeus.png");
110             zeusImg2 = loadTexture("res/ZeusAngry.png");
111             zeusHitImg = createWhiteSilhouette(zeusImg);
112             zeusHitImg2 = createWhiteSilhouette(zeusImg2);
113             // ZEUS's LIGHTNING
114             lightingImg = loadTexture("res/lighting.png");
115             lightingImg2 = loadTexture("res/lightingAngry.png");
```

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

```
154     // 画像を安全に読み込むためのヘルパーメソッド
155     private static BufferedImage loadTexture(String path) throws IOException {
156         // クラスパスからリソースを探す
157         java.net.URL url = ResourceManager.class.getClassLoader().getResource(
        path);
158 
159         if (url == null) {
160             // もし getResource で見つからない場合（フォルダ構成の違いなど)、
161             // 通常のファイルパスとして読み込みを試みる（フォールバック処理）
162             try {
163                 return ImageIO.read(new File(path));
164             } catch (IOException ex) {
165                 throw new IOException("Image not found: " + path);
166             }
167         }
168         return ImageIO.read(url);
169     }
170 
171     // ダメージ演出用に、透明度を維持したまま「真っ白なシルエット」を作成するメソッド
172     private static BufferedImage createWhiteSilhouette(BufferedImage original)
         {
173         // 元の画像と同じサイズで、空の画像を作成
174         BufferedImage whiteImg = new BufferedImage(
175                 original.getWidth(),
176                 original.getHeight(),
177                 BufferedImage.TYPE_INT_ARGB
178         );
179 
180         // すべてのピクセルを走査する
181         for (int x = 0; x < original.getWidth(); x++) {
182             for (int y = 0; y < original.getHeight(); y++) {
183                 int p = original.getRGB(x, y);
184 
185                 // アルファ値（透明度）を取得
186                 int a = (p >> 24) & 0xff;
187 
188                 // 透明ではない部分（キャラクター部分）だけを「真っ白」に塗りつぶす
189                 if (a > 0) {
190                     // ARGB: アルファ値 + R(255) + G(255) + B(255)
191                     int whiteColor = (a << 24) | (255 << 16) | (255 << 8) |
```

```
    255;
192                     whiteImg.setRGB(x, y, whiteColor);
193                 }
194             }
195         }
196         return whiteImg;
197     }
198 
199     private static BufferedImage createBlackSilhouette(BufferedImage original)
     {
200         BufferedImage blackImg = new BufferedImage(original.getWidth(),
    original.getHeight(), BufferedImage.TYPE_INT_ARGB);
201         for (int x = 0; x < original.getWidth(); x++) {
202             for (int y = 0; y < original.getHeight(); y++) {
203                 int p = original.getRGB(x, y);
204                 int a = (p >> 24) & 0xff; // Get Alpha
205 
206                 // If the pixel is not transparent, make it BLACK
207                 if (a > 0) {
208                     // ARGB: Alpha + R(0) + G(0) + B(0)
209                     int blackColor = (a << 24) | (0 << 16) | (0 << 8) | 0;
210                     blackImg.setRGB(x, y, blackColor);
211                 }
212             }
213         }
214         return blackImg;
215     }
216 }
```

(文責：佐々木)