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

7　付録 1：操作法マニュアル

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

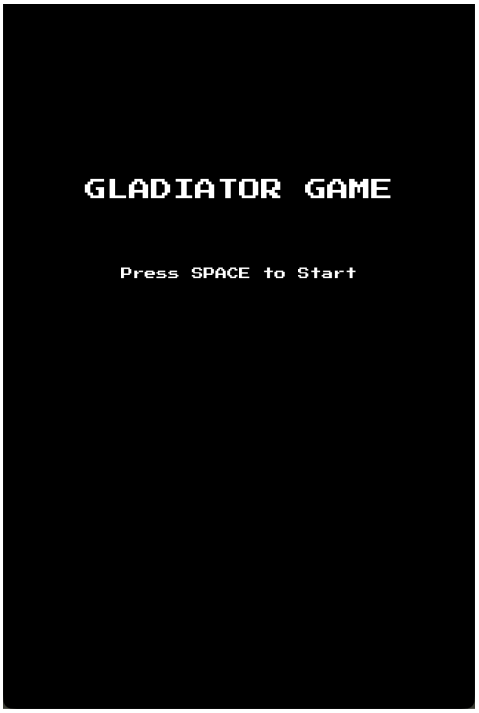

図 2: タイトル画面

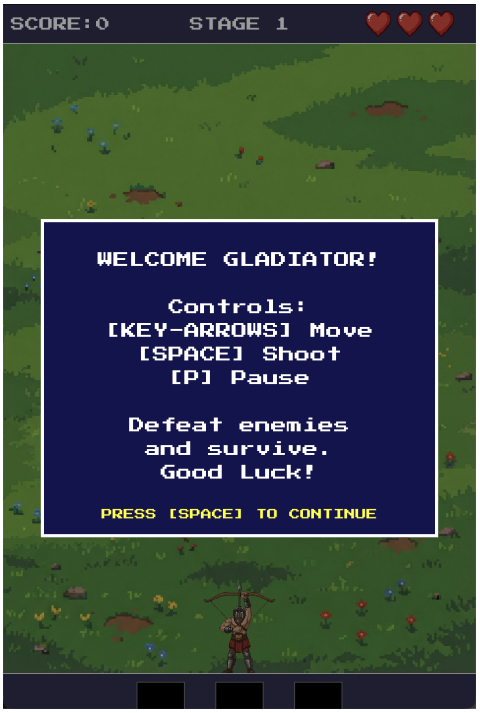

図 3: スタート画面

次に、プレイ中のキー操作について説明します。

表 1: プレイ中のキー操作

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

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

図 4: ゲームオーバー画面

(文責：佐々木)

8　付録２：プログラムリスト

Listing 1: Main.java

```
1  package main;
2
3  import view.GamePanel;
4  import view.ResourceManager;
5
6  import javax.swing.*;
7
8  public class  Main {
9
10     public static void main(String[] args) {
11         // 1. Load resources BEFORE creating the window
12         ResourceManager.loadImages();
13
```

```
        // 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

```
package model;

import view.ResourceManager;

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

// --- GameModel (M) ---
public class GameModel {
    private ArrayList<GameObject> objects;
    private Player player;
    private boolean isGameOver = false;
    private Random rand = new Random();
    private ArrayList<GameObject> newObjectsBuffer; // 弾を追加するための予約リスト（ループ中のエラー回避用）
    private GameState state; // 現在のゲーム状態
    private boolean isFiring; // スペースキーが押されているか
    private int shotTimer; // 連射間隔を制御するタイマー
    private int arrowDamage;
    private int arrowInterval;

    // Score progression
    private static int score = 0; //スコアの導入
    private int nextTargetScore;
    private int currentLevelIndex = 0;
    private boolean isBossActive = false; //ボスのフェーズの確認
```

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

```
67          arrowDamage = GameConstants.ARROW_DAMAGE;
68          arrowInterval = GameConstants.ARROW_INTERVAL;
69          state = GameState.PLAYING;
70          score = 0; //スコアをリセット
71
72          // 追加ライフ初期化:
73          lives = GameConstants.PLAYER_MAX_LIVES;
74          damageTimer = 0;
75
76          //Score progression resets
77          this.currentLevelIndex = 0;
78          this.nextTargetScore = GameConstants.LEVEL_MILESTONES[
    currentLevelIndex];
79          isBossActive = false;
80
81          // STAGE 1 SETUP: Add Harpy Spawner
82          activeSpawners.add(new EnemySpawner(Harpy.class, GameConstants.
    HARPY_SPAWN_INTERVAL, GameConstants.HARPY_SPAWN_VARIANCE));
83          this.currentStage = 1;
84
85
86          String tutorial = "WELCOME GLADIATOR!\n\n" +
87                  "Controls:\n" +
88                  "[KEY-ARROWS] Move\n" +
89                  "[SPACE] Shoot\n" +
90                  "[P] Pause\n\n" +
91                  "Defeat enemies\n" +
92                  "and survive.\n" +
93                  "Good Luck!";
94          showMessage(tutorial);
95
96
97      }
98
99      public static void addScore(int points){
100         score += points;
101     }
102
103     public void setFiring(boolean firing) {
104         this.isFiring = firing;
```

```
105     }
106
107     public GameState getState() {
108         return state;
109     }
110
111     public void setState(GameState s) {
112         this.state = s;
113     }
114
115     public void update() {
116         if (state != GameState.PLAYING) return;
117
118         // Update background scrolling
119         if(background!= null) {
120             background.update();
121         }
122
123         checkLevelProgression();
124
125
126
127         // --- 連射ロジック ---
128         if (isFiring) {
129             if (shotTimer == 0) {
130                 playerShoot();
131                 shotTimer = GameConstants.ARROW_INTERVAL;
132             }
133         }
134         if (shotTimer > 0) {
135             shotTimer--;
136         }
137
138         // 無敵時間の更新
139         if (damageTimer > 0) {
140             damageTimer--;
141         }
142
143         if (ability1Timer > 0) {
144             ability1Timer--;
```

```
145         }
146 
147         if (ability2Timer > 0) {
148             ability2Timer--;
149         }
150 
151         if (ability3Timer > 0) {
152             ability3Timer--;
153         }
154 
155         // --- NEW SPAWN LOGIC ---
156         // Iterate through all active spawners
157         if (!isBossActive) {
158             for (EnemySpawner spawner : activeSpawners) {
159                 // If spawner says "True", create that enemy
160                 if (spawner.update()) {
161                     spawnMinion(spawner.getEnemyType());
162                 }
163             }
164         }
165 
166         // オブジェクト追加
167         objects.addAll(newObjectsBuffer);
168         newObjectsBuffer.clear();
169 
170         // 移動
171         for (GameObject obj : objects) {
172             obj.move();
173         }
174 
175         // 当たり判定
176         checkCollisions();
177 
178         // 削除
179         objects.removeIf(obj -> obj.isDead());
180     }
181 
182     /**
183      * Helper method to instantiate the correct enemy based on Class type.
184      */
```

```
185     private void spawnMinion(Class<? extends Minion> type) {
186         int x,y;
187         if (type == Harpy.class) {
188             x = rand.nextInt(GameConstants.WINDOW_WIDTH - GameConstants.
    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

```java
package model;

public final class GameConstants {
    // 画面のサイズ
    public static final int WINDOW_WIDTH = 600;

    //HEIGTHS
    public static final int HUD_HEIGHT = 50;
    public static final int FIELD_HEIGHT = 800;
    public static final int BOTTOM_HUD_HEIGHT = 100;
    public static final int WINDOW_HEIGHT = HUD_HEIGHT + FIELD_HEIGHT +
    BOTTOM_HUD_HEIGHT;

    // ゲームの設定
    public static final int FPS = 60;

    // 背景の設定
    public static final double SCREEN_SPEED = 1.0;

    // の設定 PLAYER
    public static final int PLAYER_WIDTH = 90;
    public static final int PLAYER_HEIGHT = PLAYER_WIDTH*923/721; // set to
    the image height width ration
    public static final int PLAYER_SPEED = 8;
    public static final int PLAYER_MAX_LIVES = 3;

    //ABILITY TIMERS
    public static final int ABILITY1TIMER = FPS * 15; // 15 seconds
    public static final int ABILITY2TIMER = FPS * 8; // 10 seconds
    public static final int ABILITY3TIMER = FPS * 10; // 10 seconds

    // の設定 ARROW
    public static final int ARROW_WIDTH = 10;
    public static final int ARROW_HEIGHT = 70; // set to the image height
    width ration
    public static final int ARROW_SPEED = 30;
    public static final int ARROW_INTERVAL = 20;
    public static final int ARROW_INTERVAL2 = 15;
    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;
18         this.height = h;
19         this.image = image;
20     }
21
22     public abstract void move();
23
24     public abstract void draw(Graphics g);
25
26     public Rectangle getBounds() {
27         return new Rectangle(x, y, width, height);
28     }
29
30     public Shape getShape() {
31         return new Rectangle2D.Float(x, y, width, height);
```

```
32      }
33
34      public boolean isDead() {
35          return isDead;
36      }
37
38      public void setDead() {
39          this.isDead = true;
40      }
41
42      public int getY() {
43          return y;
44      }
45
46      public int getX() {
47          return x;
48      }
49  }
```

Listing 5: GameState.java

```
1   package model;
2
3   // ゲームの状態を定義
4   public enum GameState {
5       TITLE, PLAYING, PAUSED, MESSAGE, GAMEOVER
6   }
```

Listing 6: Alignment.java

```
1   package model;
2
3   /**
4    * Alignment Enum
5    * Defines which "team" a game object belongs to.
6    * Used to prevent friendly fire and determine collision logic.
7    */
8   public enum Alignment {
9       PLAYER, // Belongs to the Player (Targets Enemies)
10      ENEMY  // Belongs to Enemies (Targets Player)
```

```
}
```

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

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

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

    public void update() {
```

```
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
61              drawForceSize(g, currentY, false);
62
63              // 2. Draw Flipped Tile BELOW
64              // Covers: y+400 to y+800
65              drawForceSize(g, currentY + HEIGHT, true);
66
67              // 3. Draw Flipped Tile ABOVE
68              // Covers: y-400 to y. (Crucial for when y starts at 0 or is small
    )
69              drawForceSize(g, currentY - HEIGHT, true);
70
71              // 4. Draw Normal Tile ABOVE that
72              // Covers: y-800 to y-400. (Crucial for the loop wrap-around)
73              drawForceSize(g, currentY - (HEIGHT * 2), false);
```

```
74          } else {
75              g.drawImage(image, 0, GameConstants.HUD_HEIGHT, WIDTH, HEIGHT * 2,
        null);
76          }
77      }
78
79      /**
80       * Helper method to draw a tile either normally or vertically flipped.
81       * @param g Graphics context
82       * @param yPos The Y position to draw at
83       * @param isFlipped If true, draws the image upside down (mirrored)
84       */
85      private void drawForceSize(Graphics g, int yPos, boolean isFlipped) {
86          if (!isFlipped) {
87              // NORMAL: Force width 600, height 400
88              g.drawImage(image, 0, yPos, WIDTH, HEIGHT, null);
89          } else {
90              // FLIPPED: Force width 600, height 400 (but drawn upwards)
91              // Destination Y starts at bottom (yPos + HEIGHT)
92              // Height is negative (-HEIGHT) to flip it
93              g.drawImage(image, 0, yPos + HEIGHT, WIDTH, -HEIGHT, null);
94          }
95      }
96
97      private boolean isStage2Img(){
98          return (this.image == ResourceManager.stage2Img);
99      }
100 }
```

Listing 8: Projectile.java

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

```
 * 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; }
}
```

(文責：佐々木)