CmpE 160 Assignment 2
THIS IS THE ONLY LEVEL

Contact:
Volkan Bora Seki (volkan.seki@std.bogazici.edu.tr)
Deniz Baran Aksoy (deniz.aksoy@bogazici.edu.tr)

Due: April 7th 5 a.m. in the morning

Overview

In this assignment, you are required to implement an object-oriented version of the nostalgic game `"This Is the Only Level"` using **Java** and the **StdDraw** graphics library. At first glance, the game appears straightforward: the player, controlling an elephant character, must navigate a single level and reach the exit door. However, the challenge lies in the fact that this "only level" must be played repeatedly, with each stage introducing a unique and unexpected twist that requires the player to adapt their strategy. A game instance is shown in Figure 1 below. The primary objective is to successfully pass through the exit pipe to progress to the next stage. You can watch the gameplay from here to get a better understanding.

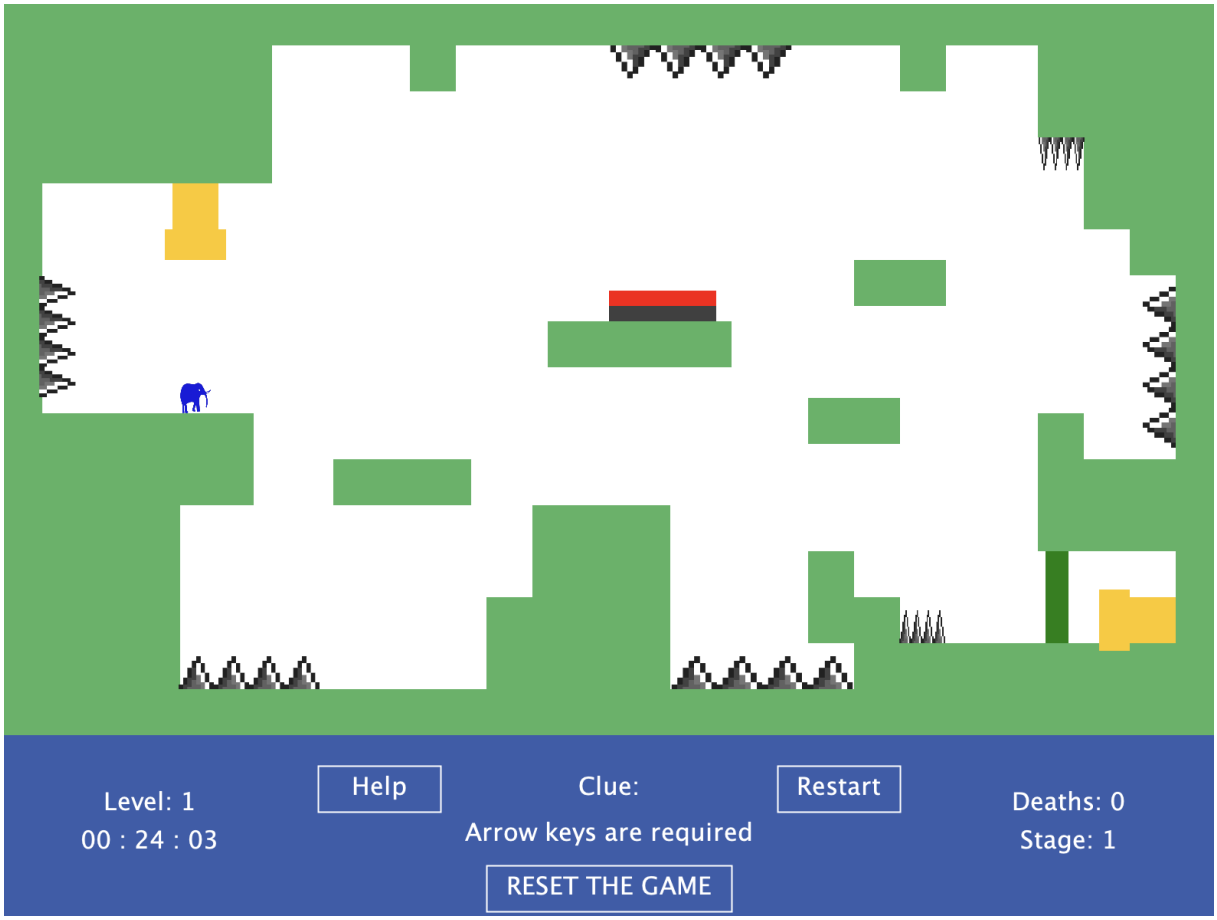

Figure 1: Game instance

Game Environment

The game environment can be seen in Figure 2 and consists of the following elements:

- **Player:** A 20x20 pixel square representing the elephant character.
- **Obstacles:** Solid blocks that restrict player movement.
- **Spikes:** If the player touches a spike, they restart from the spawn point.
- **Button:** Some levels require pressing a button to unlock the door.
- **Door:** A door blocks the player from reaching the exit pipe. It opens when a player clicks the button. The door opens by sliding down.
- **Start and Exit Pipes:** The Player spawns inside the start pipe and advances to the next stage by passing through the exit pipe.
- **Clue & Help Button:** In each level, a **clue** is displayed to help the player understand the new mechanic, and a **help button**, which is activated by a mouse click, gives a direct solution for that specific stage instead of a clue.
- **Game Timer & Death Counter:** Tracks the player's total playtime and number of deaths.
- **Level Counter:** Tracks the level, which is only "Level 1".
- **Stage Counter:** Tracks the current stage number the player is on.
- **Reset the Game Button:** A button, activated by a mouse click, resets the game timer and death counter to 0 and sets the user's stage back to stage 1.
- **Restart Button:** A button that is activated by a mouse click does not change the current stage but respawns the player and increments the death counter by 1.

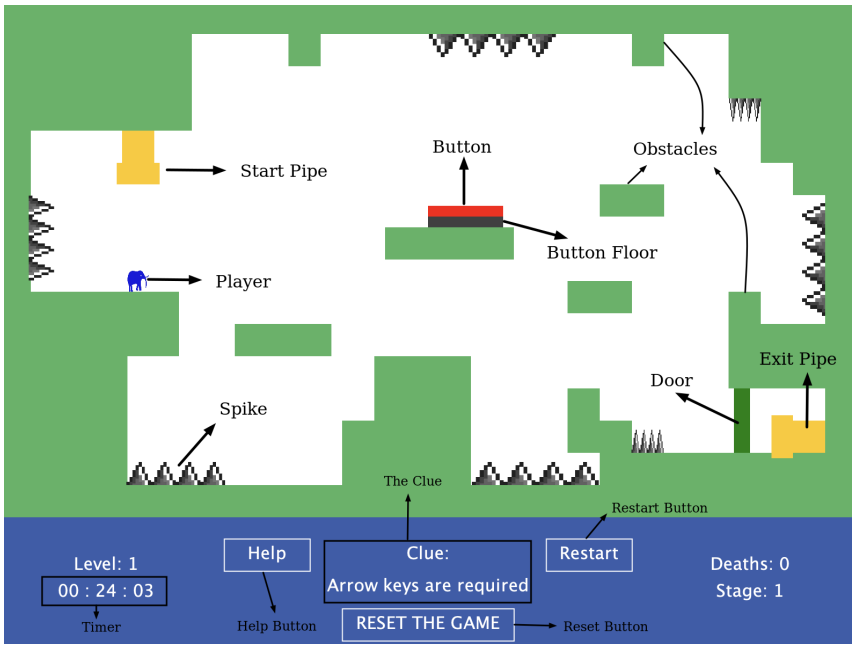

Figure 2: Game Environment.

- **Stage Passing Banner:** A green banner displays "`You passed the stage But is the level over?!`". Figure 3 illustrates this banner.
- **End Game Banner:** A green banner displays "`CONGRATULATIONS YOU FINISHED THE LEVEL PRESS 'A' TO PLAY AGAIN!`" and "You finished with `deathNumber` in `gameTime`", where `deathNumber` refers the total number of times the player died and `gameTime` refers the total playtime. Figure 4 illustrates this banner.

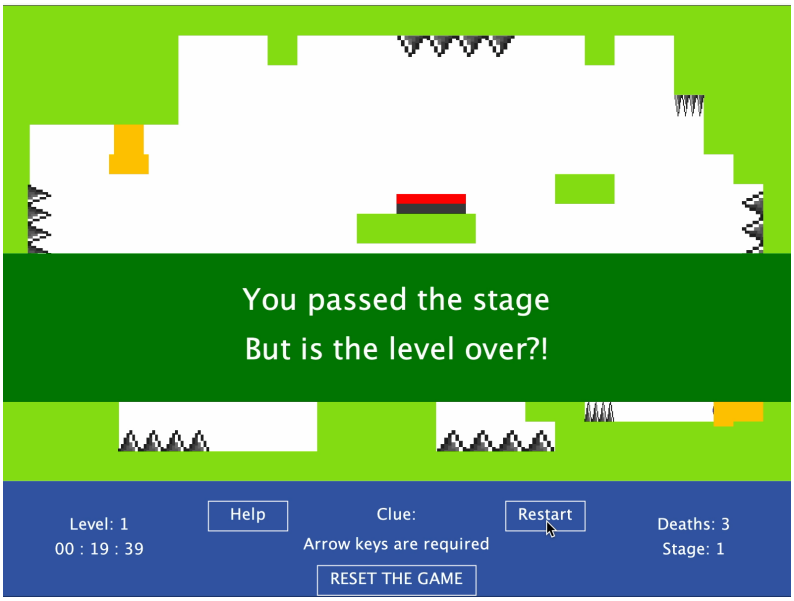

Figure 3: Passing Stage Screen.

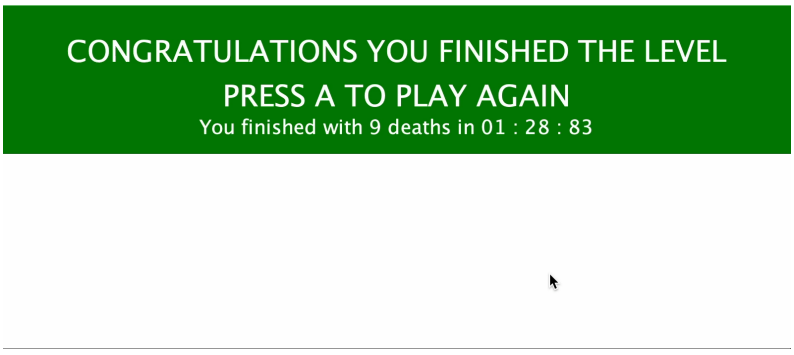

Figure 4: End Game Screen.

Game Mechanics

a) Player Control

The player should be able to control the elephant using keyboard input. Movement and interaction include.

- **Left arrow ⟵ and right arrow ⟶ keys:** Move the player left or right.
- **Up arrow ↑ key:** Makes the player jump (depending on the mechanics of the level). The player can only jump when it is on the ground.
- When the player changes the direction of movement, the elephant also needs to turn in the same direction.

b) Obstacle Drawing:

In each stage, obstacles are drawn with **random** colors.

c) Interacting with Door:

The door blocks the player from reaching the exit pipe until it is opened.

d) Interacting with Buttons:

Some levels require pressing a button before accessing the door. When the elephant steps on the button, its top part instantly drops and disappears.

e) Collision Detection:

The player cannot pass through obstacles or spikes. Spikes cause the player to start the stage from the beginning and increment the `death counter`.

f) Timer Updates:

Timer continuously updates, displaying the time in the format: `Minutes:Seconds:Milliseconds`. If any element is below ten, a leading zero is added. When the player passes the stage, the timer stops and continues as the player starts a new stage.

g) Passing a Stage:

After passing a stage, the `Stage Passing Banner` appears for 2 seconds, and the stage is updated. The timer stops for 2 seconds.

h) Passing the Level:

After the player passes all the stages and thus completes the level, `End Game Banner` remains on the screen until the player presses **A**. Then, the game starts from the beginning. If the player presses **Q**, the game ends.

Stages

This game has only one level, but there are different stages. You are expected to implement a total of five stages. Four of them are provided below, and for one stage, you should select a stage from the original game. You can examine the original game online from here.

Stage 1: Arrow Keys Are Required

This is the first stage. Left arrow key (←) makes the elephant move left, right arrow key (→) makes the elephant move right, and up (↑) makes the player jump. The player must press the button and reach the exit pipe to advance to the next stage.

Stage 2: Not Always Straight Forward

In this stage, the left and right arrow keys are reversed. The left arrow key makes the player move right, and the right arrow key makes the player go left. When the player moves left, the elephant should face right, and when the player moves right, the elephant should face left. Other than that, the player must press the button to open the door and reach the exit pipe.

Stage 3: A bit bouncy here

In this stage, you should disable the up button and make the player jump every instance it touches to the ground. Also, you should change the parameters in

- `gravity` = -2
- `velocityY`= 24

Stage 4: Never Gonna Give You Up

In this stage, controls are back to normal, but the door should not open until the button is pressed 5 times. If the player dies or restarts, the player must press the button 5 times again.

Game Parameters

The suggested game parameters are provided below in Code Box 1. However, any implementation that functions correctly and maintains the integrity of the original map is acceptable. You are not required to use given parameters, but you must adhere to the core mechanics of the original game.

Code 1 : Game Parameters Java `java`

```java
// keyCodes are KeyEvent.VK_RIGHT, KeyEvent.VK_A etc.
// Obstacles List (formant is int[] = {xLeftDown , yLeftDown, xRightUp, yRightUp}
private int[][] obstacles = {
new int[]{0, 120, 120, 270}, new int[]{0, 270, 168, 330},
new int[]{0, 330, 30, 480}, new int[]{0, 480, 180, 600},
new int[]{180, 570, 680, 600}, new int[]{270, 540, 300, 570},
new int[]{590, 540, 620, 570}, new int[]{680, 510, 800, 600},
new int[]{710, 450, 800, 510}, new int[]{740, 420, 800, 450},
new int[]{770, 300, 800, 420}, new int[]{680, 240, 800, 300},
new int[]{680, 300, 710, 330}, new int[]{770, 180, 800, 240},
new int[]{0, 120, 800, 150}, new int[]{560, 150, 800, 180},
new int[]{530, 180, 590, 210}, new int[]{530, 210, 560, 240},
new int[]{320, 150, 440, 210}, new int[]{350, 210, 440, 270},
new int[]{220, 270, 310, 300}, new int[]{360, 360, 480, 390},
new int[]{530, 310, 590, 340}, new int[]{560, 400, 620, 430}};

// Button Coordinates
private int[] button = new int[]{400, 390, 470, 410};

// Button Floor Coordinates
```

```
private int[] buttonFloor = new int[]{400, 390, 470, 400};

// Start Pipe Coordinates for Drawing
private int[][] startPipe = {new int[]{115, 450, 145, 480},
                             new int[]{110, 430, 150, 450}};
// Exit Pipe Coordinates for Drawing
private int[][] exitPipe = {new int[]{720, 175, 740, 215},
                            new int[]{740, 180, 770, 210}};
// Coordinates of spike areas
private int[][] spikes = {
new int[]{30, 333, 50, 423}, new int[]{121, 150, 207, 170},
new int[]{441, 150, 557, 170}, new int[]{591, 180, 621, 200},
new int[]{750, 301, 769, 419}, new int[]{680, 490, 710, 510},
new int[]{401, 550, 521, 570}};

// Timer Area (Blue Area at the Bottom)
private int[] timerArea = new int[]{0, 0, 800, 120};

// Door Coordinates
private int[] door = new int[]{685, 180, 700, 240};

// Given Stages
// First Stage
new Stage(-0.45, 3.65, 10, 0, KeyEvent.VK_RIGHT, KeyEvent.VK_LEFT, KeyEvent.VK_UP,
"Arrow keys are required", "Arrow keys move player, press button and enter the second
↪  pipe");
// Second Stage
new Stage(-0.45, 3.65, 10, 1, KeyEvent.VK_LEFT, KeyEvent.VK_RIGHT, KeyEvent.VK_UP,
"Not always straight forward", "Right and left buttons reversed");
// Third Stage
new Stage(-2, 3.65, 24, 2, KeyEvent.VK_RIGHT, KeyEvent.VK_LEFT, nullButton,
"A bit bouncy here", "You jump constantly");
// Fourth Stage
new Stage(-0.45, 3.65, 10, 3, KeyEvent.VK_RIGHT, KeyEvent.VK_LEFT, KeyEvent.VK_UP,
"Never gonna give you up", "Press button 5 times ");
// For drawing the game components
// The elephant's width and height are 20 for drawing
StdDraw.setCanvasSize(800, 600);
StdDraw.setXscale(0, 800);
StdDraw.setYscale(0, 600);
StdDraw.setPenColor(new Color(56, 93, 172)); // Color of the area
StdDraw.filledRectangle(400, 60, 400, 60); // Drawing timer area
StdDraw.setPenColor(StdDraw.WHITE);
StdDraw.text(250,85,"Help");
StdDraw.rectangle(250,85,40,15); // Help button
StdDraw.text(550,85,"Restart");
StdDraw.rectangle(550,85,40,15); // Restart button
StdDraw.text(400,20,"RESET THE GAME");
StdDraw.rectangle(400,20,80,15); // Reset button
StdDraw.text(700, 75, "Deaths: " + deathCounter);
StdDraw.text(700, 50, "Stage: " + stageNumber);
StdDraw.text(100, 50, timerText);
StdDraw.text(100,75, "Level: 1");
StdDraw.text(400, 85, "Clue:");
StdDraw.text(400, 55, clueText);
```

Class Descriptions

In this assignment, in addition to the Main class, four other classes must be implemented. These classes are `Stage, Player, Map` and `Game`. UML diagrams and explanations for each class are provided below. You are expected to implement the listed methods and properties but not restricted by them. You can define additional methods and properties as needed for each class.

a) `Stage`

`Stage` class represents a stage in the game. It provides data and methods that allow other classes to access, modify, or interact with `Stage` properties. These properties and methods are provided in Figure 5 below. `velocityX` and `velocityY` represent the player's velocity in the x and y directions through the stage. If the player wants to move in the x direction, the position of the player should be updated by `velocityX`. When the player jumps the player's velocity in the y direction starts from `velocityY` and gradually decreases due to `gravity` until the elephant collides with an obstacle. Color of the obstacles should be random in each stage.

Stage	
- stageNumber: int	Stage number (1-5)
- gravity: double	Gravity applied to the player
- velocityX: double	Player's velocity in x direction
- velocityY: double	Player's velocity in y direction
- rightCode: int	Key code for right movement
- leftCode: int	Key code for left movement
- upCode: int	Key code for up movement
- clue: String	Stage clue
- help: String	Help for stage
- color: Color	Color of the obstacles
+ Stage (gravity: double, velocityX: double, velocityY: double, stageNumber: int, rightCode: int, leftCode: int, upCode: int, clue: String, help: String)	Constructor of the class
+ getStageNumber(): int	Returns the stage number
+ getGravity(): double	Returns the gravity value
+ getVelocityX(): double	Returns the velocity in x direction
+ getVelocityY(): double	Returns the velocity in y direction
+ getKeyCodes(): int[]	Returns the key codes of the stage [rightCode, leftCode, upCode]
+ getClue(): String	Returns the clue for the stage
+ getHelp(): String	Returns the help for the stage
+ getColor(): Color	Returns the color of the obstacles

Figure 5: UML Diagram of the `Level` class.

b) `Player`

`Player` class represents the player in the game. It provides data and methods to modify the player's properties and interact with them. UML diagram of the class provided in Figure 6.

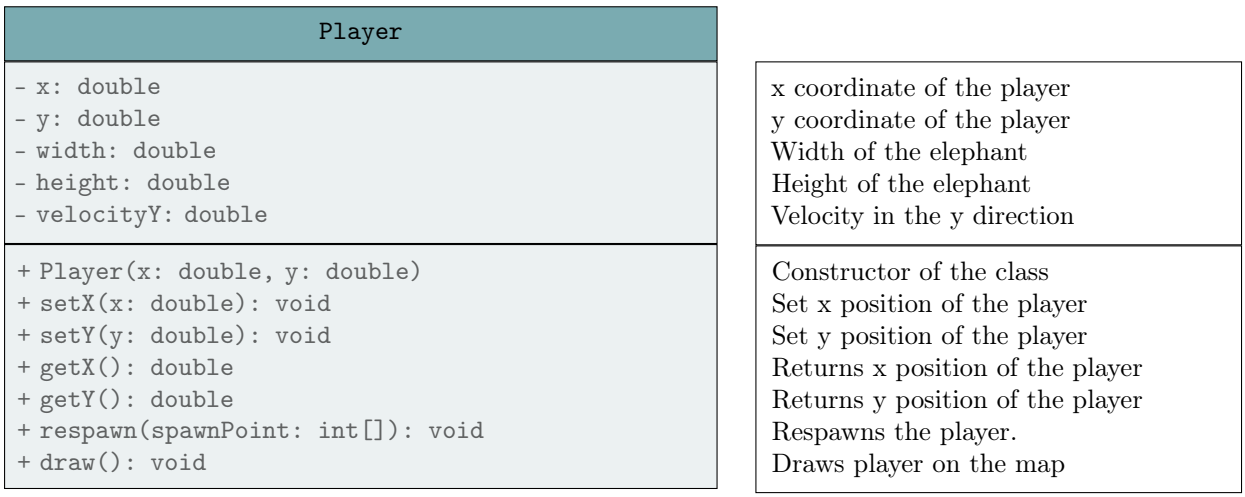

Figure 6: UML Diagram of the `Player` class.

c) `Map`

`Map` class represents the whole game environment with all of its elements. This class is also responsible for players' movement on the map and opening animation of the door. UML diagram of the class can be seen in Figure 7. Some properties of this class are given in Code Box 1.

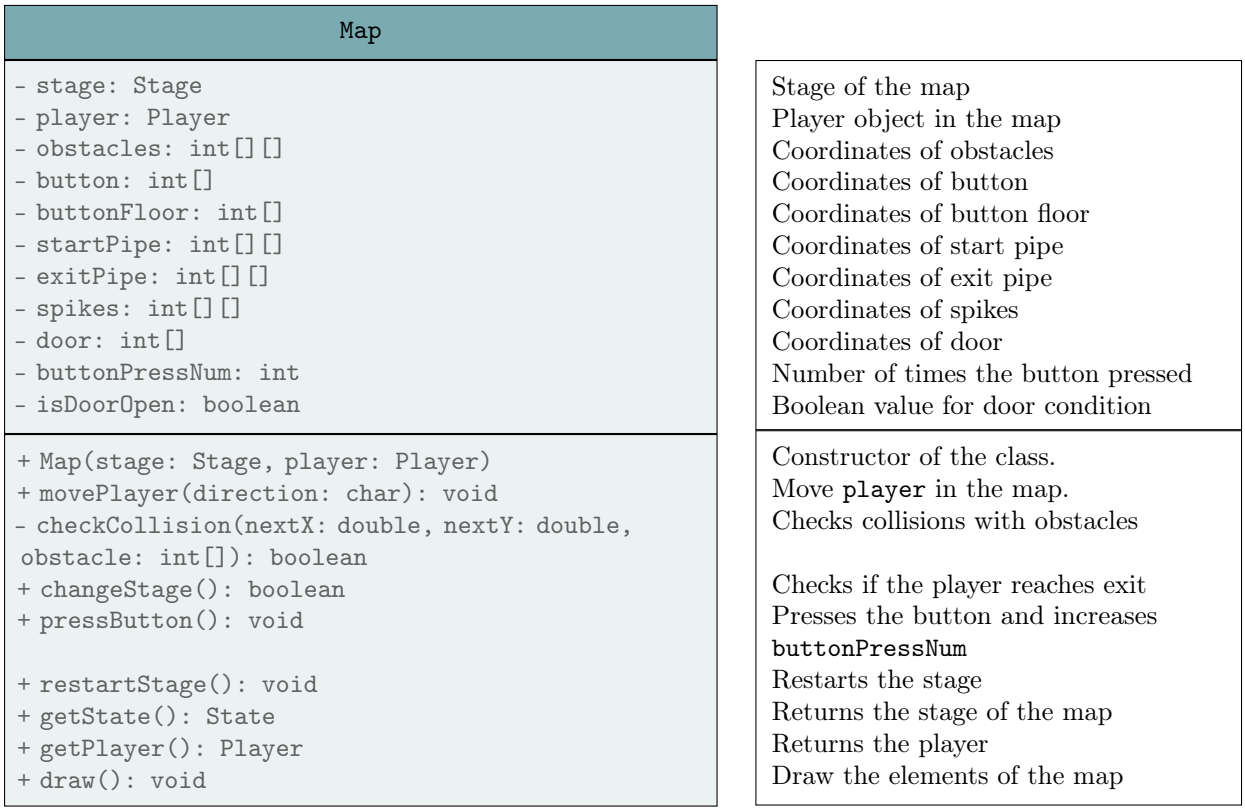

Figure 7: UML Diagram of the `Map` class.

d) `Game`

`Game` class is the main class where all interaction with the user happens. When `play` is called, the game begins, and the player can move the elephant to complete the `stages`. For each stage, `Game` instantiates a `Map` object with a new `Stage` from the `stages` list and `Player` moves in the map controlled by the user. If the player hits a spike, the `deathNumber` increases, and `Player` respawns at the `startPipe` to continue playing with keys. If all stages are passed, and the user presses (A), the game starts from the beginning as described before. UML diagram of the class explaining its properties and methods can be seen in Figure 8.

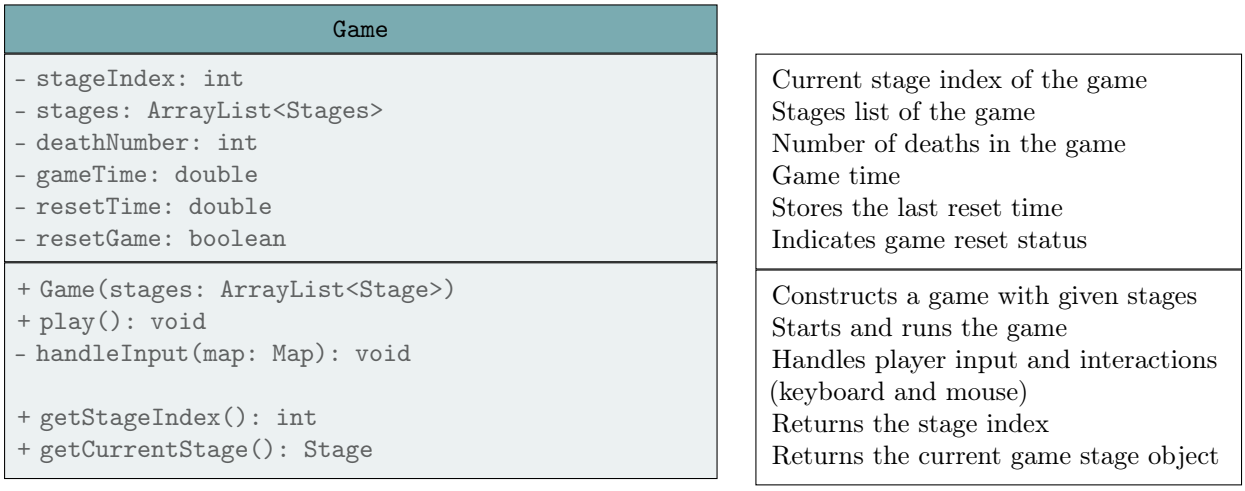

Figure 8: UML Diagram of the `Game` class.

Report Format

Your report should be concise and consist of the following sections:

- **Introduction:** Briefly explain the game and all stages. The introduction should not exceed half a page.
- **Implementation Details:** Provide a brief overview of your code with screenshots. Explain all the classes in detail and describe how they interact with each other.

Your report should **properly reference all figures, tables, and equations**. Figures and tables should be numbered and cited appropriately (as shown in Figure 1, etc.). Each figure must have a caption placed below it, clearly explaining its content. Each table must have a caption positioned above it. Lastly, all equations should be centered and numbered for reference.

Notes

- You **must** implement in Object Oriented way.
- You **must** use the **StdDraw graphics library**.
- In all assignments, do not use topics we have not covered in the lectures.

Submission Files

1. Java source code (**.java file**). Submission details are explained in greater detail in the Submission Guide section. Write clear and meaningful comments in your code.
2. Report (**.pdf format**) explaining your implementation, including screenshots of your gameplay. Explain the stage you designed in detail.
3. Provide **a YouTube link at the beginning of your report**: showing the game played with given stages and the stage you implemented.

Submission Guide

Submission Files

Submit a single compressed (`.zip`) file, named as `NameSurname.zip`, to Moodle. It should contain all source code files (under the `\code` directory), the report (in **PDF** format, under the `\report` directory), and all other files if needed (under the `\misc` directory). Name the main Java file as `NameSurname.java`. You are expected to upload a total of five Java files (`Stage.java`, `Player.java`, `Map.java`, `Game.java`, `NameSurname.java`). Name your report as `NameSurname.pdf`. Do not use Turkish characters in filenames, code, or comments. In each `.java` file, add your name and student number as a comment at the top.

Late Submission Policy

The maximum late submission duration is two days. Late submissions will be graded at 50% of the original grade.

Mandatory Submission

Submission of assignments is **mandatory**. If you do not submit an assignment, you **fail** the course.

Plagiarism

This leads to a grade of F, and YÖK regulations will be applied.