CMPE 160 ASSİGNMENT 2 REPORT

Youtube Link: https://youtu.be/cq1saX8ti2Y

Introduction:

In this assignment I made a game similar to that called “This Is The Only Level”. The main purpose of the game is passing the all stages and finally passing the “one level”. To do so, in each stage player should find a way to open the door and pass through the exit pipe. While doing so player should also avoid the spikes since when player touches them it dies and stage restarts. In the game there are 5 stages which are slightly different than each other.

Stage1: Player can move with arrow keys and should open the door by pressing the button and reach the exit pipe.

Stage2: Same logic as in stage 1, but the arrow key controls are reversed, requiring the player to adapt to the new control scheme.

Stage3: Player is constantly jumping in this stage without needing player to press up arrow key and also jump strength and gravity are increased.

Stage4: Player should press the button 5 times to open the door other logics are kept same.

Stage5: Gravity is lovered in this stage and jumping mechanic is slightly changed so that when player jumps it goes up until it collides with some object, by so it seems like player is floating in space.

Game Environment:

Elephant: Player controlled figure.

Obstacles: rectangular blocks obstructing player to pass through.

Spikes: Deadly traps that restart the stage if touched.

Pipes: Entry and exit points of the level.

Button: A mechanism that must be activated to open the door.

Door: Rectangular shaped block blocking the player to pass through and should be opened to let the player pass.

UI Area: Includes some buttons activated by player and some counters.

- **Clue & Help Button**: In each stage a corresponding clue is displayed to help the player to understand the stage mechanics. And a help button which is activated by a mouse click and displaying a more detailed explanation of stage mechanics.
- **Game Timer & Death Counter**: Tracking and displaying the total game elapsed time in the game and total count of player’s deaths.
- **Level Counter & Stage Counter**: Tracking and displaying level and stage number.
- **Reset Button**: A button activated by a mouse click and let the player start the game again from the very beginning.
- **Restart Button**: A button activated by a mouse click and restarting the stage and increasing the death count by 1.

Game Mechanics:

1- Player Control:

Player can control the elephant using arrow keys respectively. But in some stages arrow keys does not work like expected so player should understand those tricks in each stage.

2- Obstacle Drawing:

Handled in Map class using Random class to draw obstacles in a random color in each stage.

3- Collision Detection:

Handled in Map class. Ensuring that the player can not pass through the any objects other than pipes and buttons. Generally used equation;

```
playerMaxX > objMinX && playerMinX < objMaxX &&
playerMaxY > objMinY && playerMinY < objMaxY                (1)
```

4- Passing a Stage and a Level:

When player handled opening the door and passed through the exit pipe, stageNumber increases by one and so stage changes and Stage Passing Banner is shown. If all the stages are passed it means that the “level” is completed so End Game Banner is shown until player presses “A” or “Q”. If “A” is pressed game starts again and if “Q” is pressed game ends.

Screenshots In the Game:

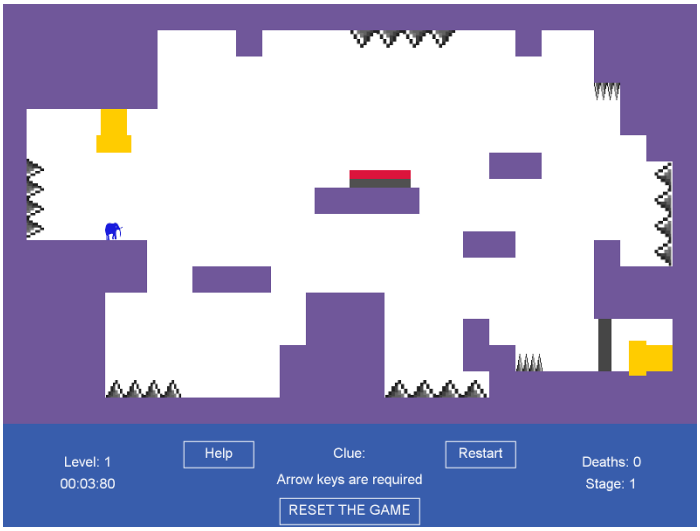


Figure 1: Initial Game

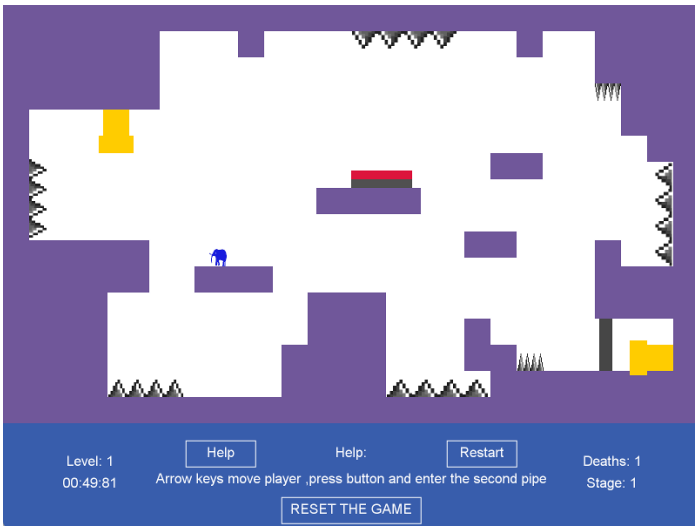


Figure 2: After Help Button is Clicked

Figure 3: Stage Passing Banner Screen

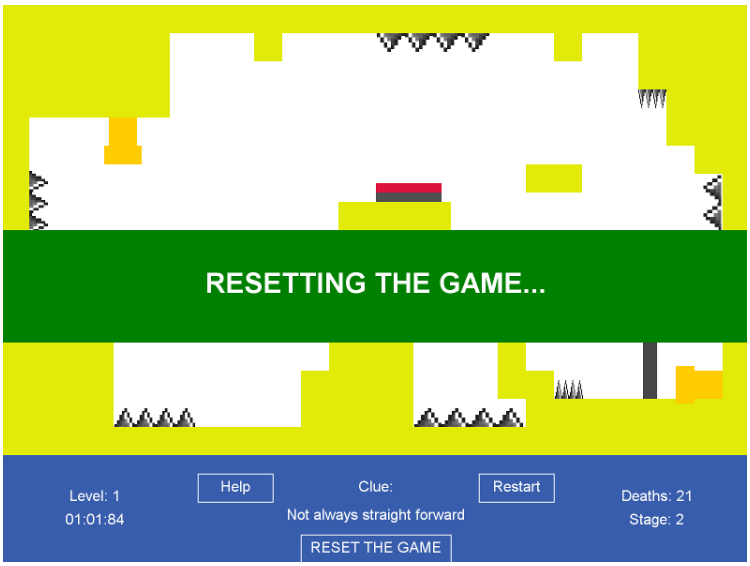


Figure 4: Reset Game Banner Screen

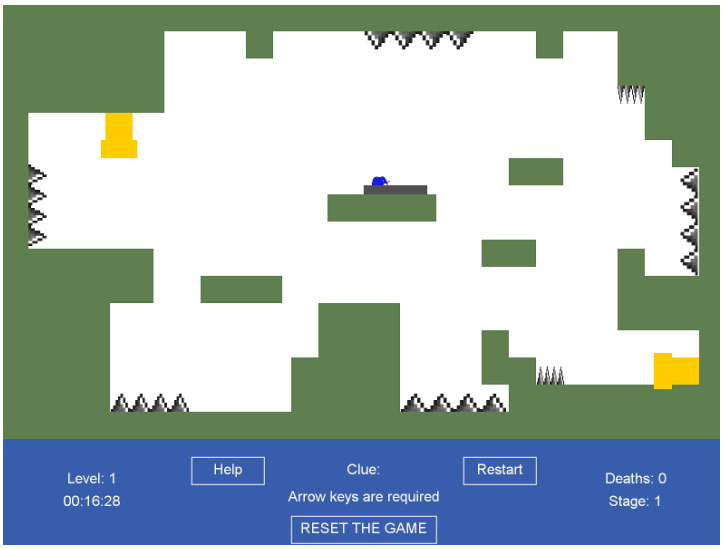


Figure 5: Button is Pressed

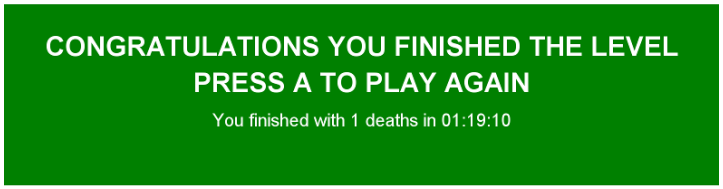
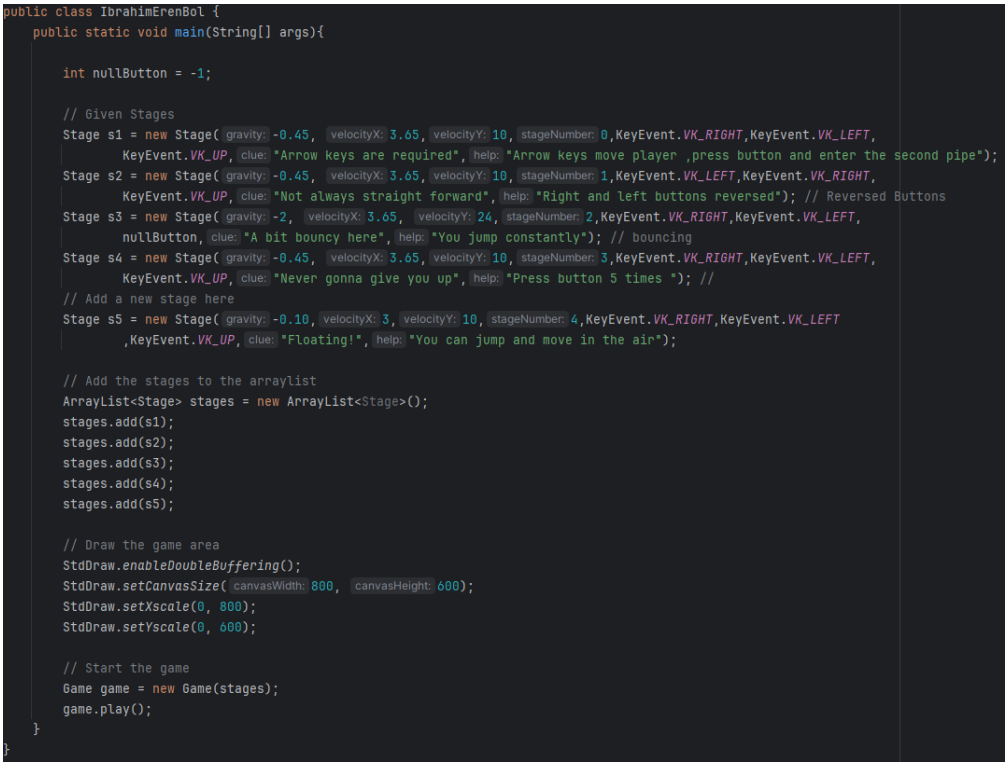


Figure 6: End Game Banner Screen

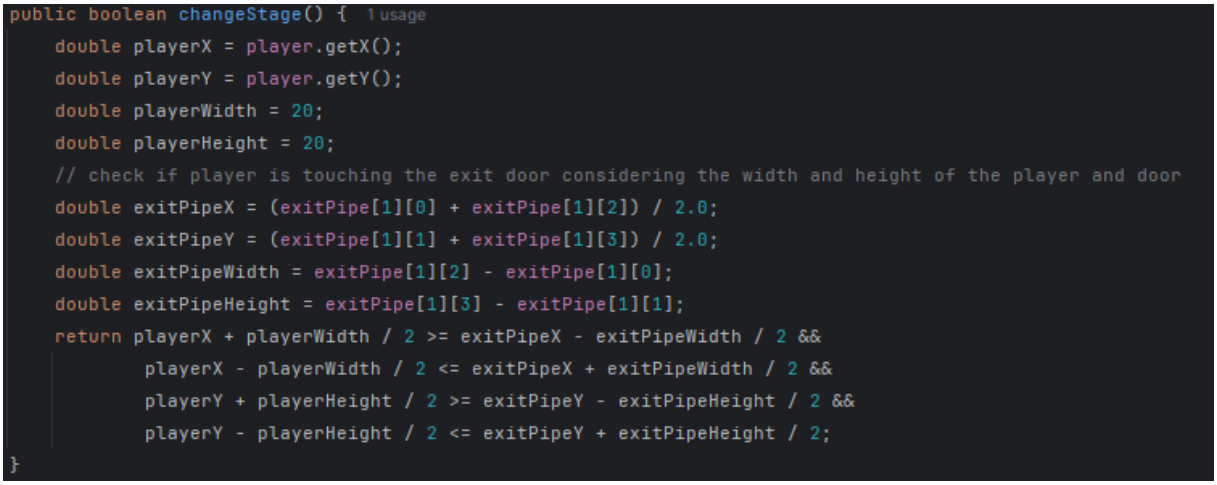
Some Code Snippets:

Code 1: Initializing and Starting the Game in Main class



```
public class IbrahimErenBol {
    public static void main(String[] args){

        int nullButton = -1;

        // Given Stages
        Stage s1 = new Stage( gravity: -0.45,  velocityX: 3.65, velocityY: 10, stageNumber: 0,KeyEvent.VK_RIGHT,KeyEvent.VK_LEFT,
                KeyEvent.VK_UP, clue: "Arrow keys are required", help: "Arrow keys move player ,press button and enter the second pipe");
        Stage s2 = new Stage( gravity: -0.45,  velocityX: 3.65, velocityY: 10, stageNumber: 1,KeyEvent.VK_LEFT,KeyEvent.VK_RIGHT,
                KeyEvent.VK_UP, clue: "Not always straight forward", help: "Right and left buttons reversed"); // Reversed Buttons
        Stage s3 = new Stage( gravity: -2,  velocityX: 3.65,  velocityY: 24, stageNumber: 2,KeyEvent.VK_RIGHT,KeyEvent.VK_LEFT,
                nullButton, clue: "A bit bouncy here", help: "You jump constantly"); // bouncing
        Stage s4 = new Stage( gravity: -0.45,  velocityX: 3.65, velocityY: 10, stageNumber: 3,KeyEvent.VK_RIGHT,KeyEvent.VK_LEFT,
                KeyEvent.VK_UP, clue: "Never gonna give you up", help: "Press button 5 times "); //
        // Add a new stage here
        Stage s5 = new Stage( gravity: -0.10, velocityX: 3, velocityY: 10, stageNumber: 4,KeyEvent.VK_RIGHT,KeyEvent.VK_LEFT
                ,KeyEvent.VK_UP, clue: "Floating!", help: "You can jump and move in the air");

        // Add the stages to the arraylist
        ArrayList<Stage> stages = new ArrayList<Stage>();
        stages.add(s1);
        stages.add(s2);
        stages.add(s3);
        stages.add(s4);
        stages.add(s5);

        // Draw the game area
        StdDraw.enableDoubleBuffering();
        StdDraw.setCanvasSize( canvasWidth: 800,  canvasHeight: 600);
        StdDraw.setXscale(0, 800);
        StdDraw.setYscale(0, 600);

        // Start the game
        Game game = new Game(stages);
        game.play();
    }
}
```

Code 2: Method for Changing the Stage



```
public boolean changeStage() {  1 usage
    double playerX = player.getX();
    double playerY = player.getY();
    double playerWidth = 20;
    double playerHeight = 20;
    // check if player is touching the exit door considering the width and height of the player and door
    double exitPipeX = (exitPipe[1][0] + exitPipe[1][2]) / 2.0;
    double exitPipeY = (exitPipe[1][1] + exitPipe[1][3]) / 2.0;
    double exitPipeWidth = exitPipe[1][2] - exitPipe[1][0];
    double exitPipeHeight = exitPipe[1][3] - exitPipe[1][1];
    return playerX + playerWidth / 2 >= exitPipeX - exitPipeWidth / 2 &&
            playerX - playerWidth / 2 <= exitPipeX + exitPipeWidth / 2 &&
            playerY + playerHeight / 2 >= exitPipeY - exitPipeHeight / 2 &&
            playerY - playerHeight / 2 <= exitPipeY + exitPipeHeight / 2;
}
```

Code 3: A Part of Drawing the Game Environment Method

```
public void draw() {  1 usage
    // Drawing obstacles
    StdDraw.setPenColor(stage.getColor());
    for(int[] obstacle : obstacles) {
        drawRectangle(obstacle);
    }

    // Drawing button considering if it is being pressed
    if(!isButtonBeingPressed){
        StdDraw.setPenColor(BUTTON_TOP);
        drawRectangle(button);
    }
    StdDraw.setPenColor(BUTTON_FLOOR);
    drawRectangle(buttonFloor);

    // Drawing the pipes
    StdDraw.setPenColor(PIPE);
    drawRectangle(startPipe[0]);
    drawRectangle(startPipe[1]);
    drawRectangle(exitPipe[0]);
    drawRectangle(exitPipe[1]);

    // Drawing the door
    if (this.doorCurrentHeight > 0) {
        StdDraw.setPenColor(DOOR);
        drawRectangle(new int[]{door[0], door[1], door[2], (int) (door[1] + this.doorCurrentHeight)});
    }
```

Code 4: Placing Player Considering Collison

```
// Determine if the player is colliding with an obstacle horizontally
int[] collidingObstacleH = checkCollisionDetailed(nextX, currentY, playerWidth, playerHeight, this.obstacles);

if (collidingObstacleH == null) {  // No horizontal collision detected
    if(checkPotentialDoorCollision(nextX,currentY)){ // Checking if the player is colliding with the door
        if(player.getDirection().equals("left"))  movePlayer( direction: 'L'); // Letting player move left while touching to door
        else if(player.getDirection().equals("right") ) {     // Restricting player to move right while touching to door
            finalX = currentX;
            player.setVelocityX(0);
        }
    } else movePlayer( direction: 'R');
} else {  // Horizontal collision detected
    if (currentVelX > 0) { // Player was moving right so set player to the left side of the obstacle
        finalX = collidingObstacleH[0] - playerWidth / 2.0;
    } else if (currentVelX < 0) { // Player was moving left so set player to the right side of the obstacle
        finalX = collidingObstacleH[2] + playerWidth / 2.0;
    }
    player.setVelocityX(0);
}

// Determine if the player is colliding with an obstacle vertically
player.setOnGround(false);
int[] collidingObstacleV = checkCollisionDetailed(finalX, nextY, playerWidth, playerHeight, this.obstacles);

if (collidingObstacleV == null) { // No vertical collision detected
    movePlayer( direction: 'U'); // Move player vertically
}
else { // Vertical collision detected
    if (currentVelY < 0) {   // Player was moving down so set player to the top side of the obstacle
        finalY = collidingObstacleV[3] + playerHeight / 2.0;
        player.setOnGround(true); // Setting the player on ground after putting it onto the obstacle
    } else if (currentVelY > 0) { // Player was moving up so set player to the bottom side of the obstacle
        finalY = collidingObstacleV[1] - playerHeight / 2.0;
    }
    player.setVelocityY(0);
}
// Finally set the player's position to the final calculated position
player.setX(finalX);
player.setY(finalY);
```

Code 5: All the Banner Displaying Methods

```
// Method to display the stage passing banner when the player passes the stage
private void displayStagePassingBanner() {  1 usage
    StdDraw.setPenColor(StdDraw.GREEN); // Set the color banner area color
    StdDraw.filledRectangle( x: 400,  y: 300,  halfWidth: 400,  halfHeight: 60); // Banner area
    StdDraw.setPenColor(StdDraw.WHITE);
    StdDraw.setFont(new Font( name: "Arial",Font.BOLD,  size: 30));
    StdDraw.text( x: 400,  y: 320,  text: "You passed the stage");
    StdDraw.text( x: 400,  y: 280,  text: "But is the level over?!");
    StdDraw.setFont();
    StdDraw.show();
    StdDraw.pause( t: 2000); // pause for 2 seconds
    totalPausedTime += 2000;
    StdDraw.clear();
}

// Method to display the reset game banner when the player resets the game
private void displayResetGameBanner() {  1 usage
    StdDraw.setPenColor(StdDraw.GREEN); // Set the color banner area color
    StdDraw.filledRectangle( x: 400,  y: 300,  halfWidth: 400,  halfHeight: 60); // Banner area
    StdDraw.setPenColor(StdDraw.WHITE);
    StdDraw.setFont(new Font( name: "Arial",Font.BOLD,  size: 30));
    StdDraw.text( x: 400,  y: 300,  text: "RESETTING THE GAME...");
    StdDraw.setFont();
    StdDraw.show();
    StdDraw.pause( t: 2000); // pause for 2 seconds
    totalPausedTime = 0 ;
    StdDraw.clear();
}

// Method to display the game over banner when the player finishes the level
private void displayGameOverBanner(String finalTime) {  1 usage
    StdDraw.clear();
    StdDraw.setPenColor(StdDraw.GREEN);
    StdDraw.filledRectangle( x: 400,  y: 300,  halfWidth: 400,  halfHeight: 100);
    StdDraw.setPenColor(StdDraw.WHITE);
    StdDraw.setFont(new Font( name: "Arial",Font.BOLD,  size: 30));
    StdDraw.text( x: 400,  y: 350,  text: "CONGRATULATIONS YOU FINISHED THE LEVEL");
    StdDraw.text( x: 400,  y: 310,  text: "PRESS A TO PLAY AGAIN");
    StdDraw.setFont(new Font( name: "Arial",Font.PLAIN,  size: 20));
    StdDraw.text( x: 400,  y: 270,  text: "You finished with " + deathNumber + " deaths in " + finalTime);
    StdDraw.setFont();
    StdDraw.show();
}
```

Code 6: Handling the User Input method (mouse part only)

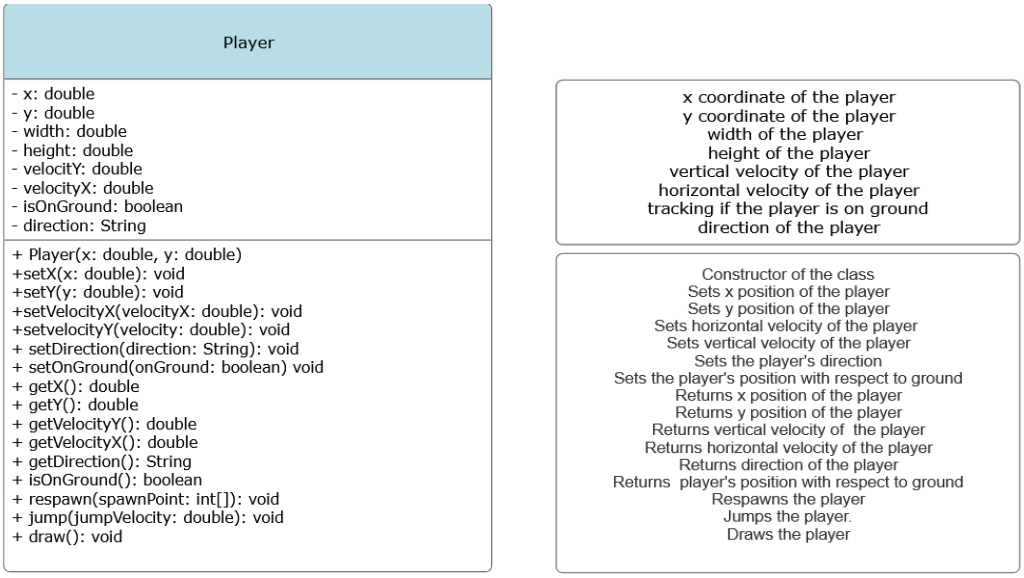
```
private void handleInput(Map map) {  1 usage
    Player player = map.getPlayer();
    // Making the player jump in stage 2 independent of player input
    if (getStageIndex() == 2) {
        player.jump(map.getStage().getVelocityY());
    }
    // Mouse input handling
    if (StdDraw.isMousePressed()) {
        double mouseX = StdDraw.mouseX();
        double mouseY = StdDraw.mouseY();
        // Help button
        if (mouseX >= 210 && mouseX <= 290 && mouseY >= 70 && mouseY <= 100) {
            showHelp = true;
        }
        // Restart button
        else if (mouseX >= 510 && mouseX <= 590 && mouseY >= 70 && mouseY <= 100) {
            map.restartStage();
            deathNumber++;
            showHelp = false;
            StdDraw.pause( t: 200);
            totalPausedTime += 200;
        }
        // Reset button
        else if (mouseX> 320 && mouseX < 480 && mouseY > 5 && mouseY < 35) {
            displayResetGameBanner();
            resetGame = true;
        }
    }
```

Class Descriptions:

In this project in addition to main class I implemented four additional classes which are Player, Stage, Map and Game.

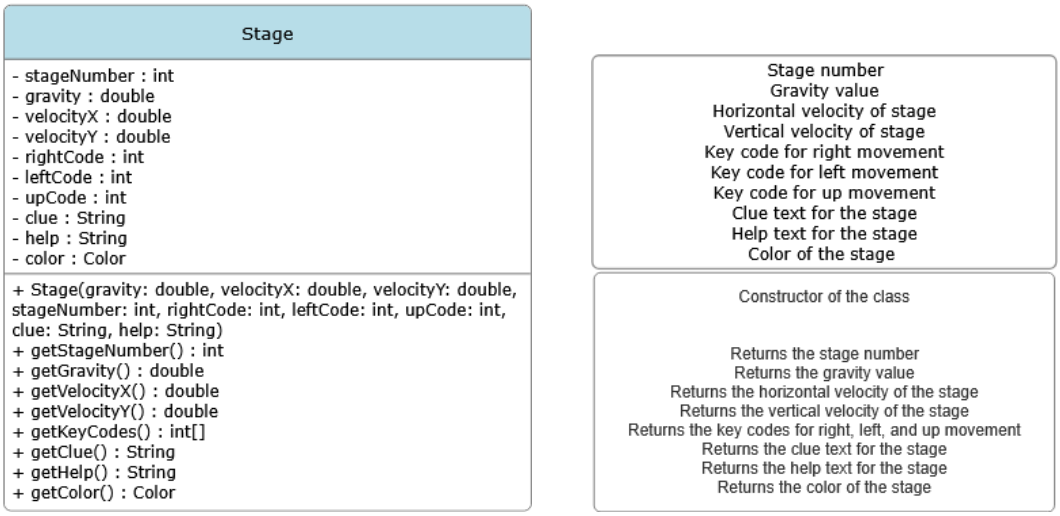
1- Player Class:

Class representing the player in the game. It is used for reaching some data about player and to arrange some of player's properties. Here is the UML diagram for Player class.



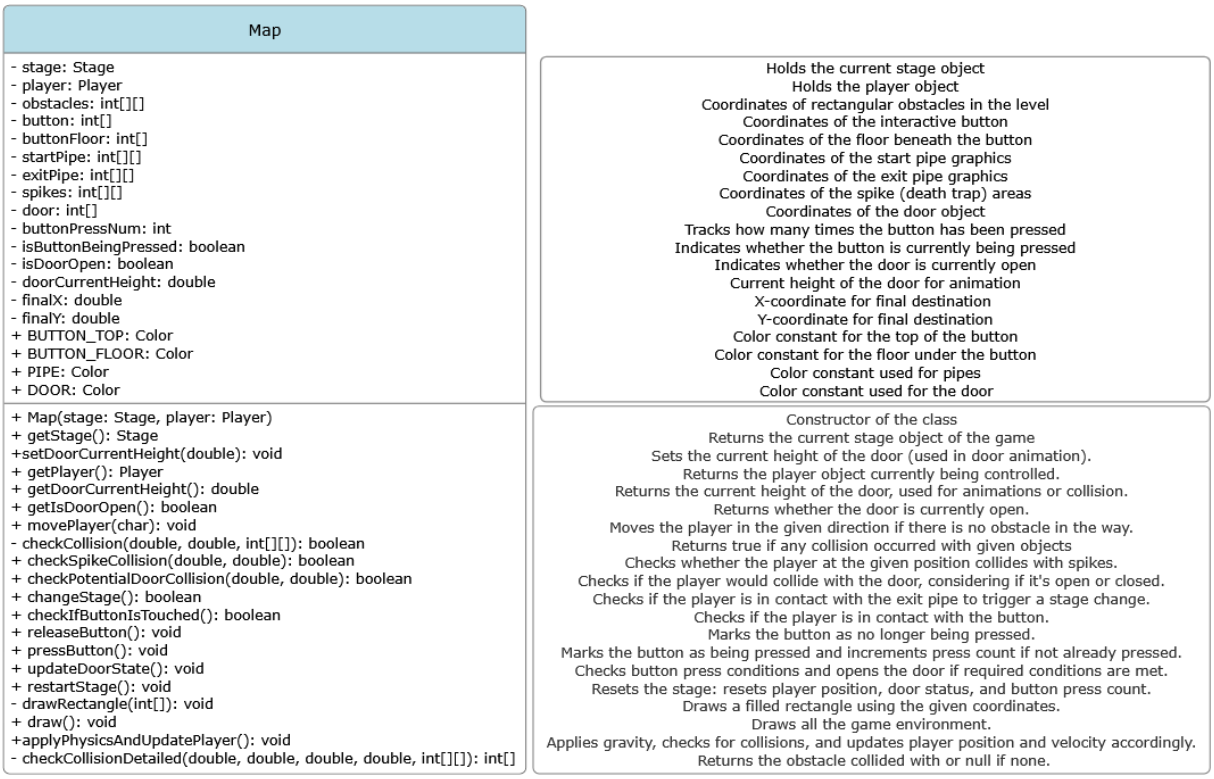
2- Stage Class:

Class representing the stages in the level and used for 5 stages totally. It is used to define stages by some properties and it provides those datas to other classes. Here is the UML diagram for Stage Class



3- Map Class:

Class representing the whole game environment and also responsible for player's movement in this environment while ensuring the movements are done correctly.



4- Game Class:

This is the main class of the project since all the user interactions and game flow is handled. A game object includes stages of the game and when the play method is invoked game starts so player could control the elephant and play the game and if the game finishes it exits the program.