

<https://www.youtube.com/watch?v=2shVUFrcUsM>

## Introduction:

In this game, I tried to make the flash game “This is The Only Level” using StdDraw library in Java. The player controls the elephant character on the screen and tries to reach the exit pipe in the bottom right corner of the screen. However, there is a door blocking access to the exit pipe. The door can simply be opened by pressing the red button in the first stage. However, after the first stage, there are different stages that have the same layout and appearance(besides colors) as the first stage, but each has its unexpected twists that requires the player to play differently. In stage 2, the player moves right when pressing the left key, and moves left when pressing the right key. In stage 3, the player can’t control their jumps. They jump constantly. Gravity and velocity of the jumps are also higher in this level. In stage 4, the player has to press the button 5 times. In stage 5, everything except the player and the start and exit pipes is invisible.

## Implementation Details:

The initial game is as seen in Figure 1. The player, obstacles, the button, start and exit pipes, the door, spikes, and obstacles are on the screen. There is also some information and buttons at the bottom of the screen. There is a level text(which only displays level 1), the timer, a death counter, stage text, and a clue that helps the player understand the stage. The player can restart the stage they are in by pressing the restart button. This increments the death counter by 1. RESET THE GAME button resets the entire game. When pressed, this button restarts the timer and the death counter and sets the stage to 1. The help button reveals the twist of the level.



Figure 1: Game Instance

There are five classes in this game called AhmetMeteAtay.java, Game.java, Stage.java, Player.java, and Map.java.

AhmetMeteAtay.java is the main class. The stages and the game object is created in this class.

Game class has ArrayList **stages** that contains all the stages in the game as an attribute. This class handles the input from the player by using the **map.movePlayer()** and **map.getPlayer().setDirection()** methods. The whole game process runs in the **play()** method of this class.

Stage class has all the unique properties of a stage which gives the unique mechanic. Color of the stage is also determined randomly in the constructor.

Player class draws the player with the direction according to the last pressed arrow key. Player's vertical velocity is stored here in order to determine the position according to gravity and this velocity.

Map class draws the player all game objects(obstacles, spikes, door, pipes, button) on the screen in the `draw()` method. This class also handles player movement in the `movePlayer()` method which takes the parameter direction from the Game class' `handleInput()` method. Vertical movement of the player is handled by the `updatePlayerHeight()` method.

## Player Movement:

The player moves by using arrow keys. These arrow key inputs are first detected by the method `handleInput()` in the Game class.

```
private void handleInput(Map map){
    //gets the input for player movement
    if (StdDraw.isKeyPressed(map.getStage().getKeyCodes()[0])){
        map.movePlayer('r');
    }
    if (StdDraw.isKeyPressed(map.getStage().getKeyCodes()[1])){
        map.movePlayer('l');
    }
    if (StdDraw.isKeyPressed(map.getStage().getKeyCodes()[2])
    ||map.getStage().getKeyCodes()[2]==-1){
        map.movePlayer('u');
    }
    if (StdDraw.isKeyPressed(KeyEvent.VK_LEFT)){
        map.getPlayer().setDirection('l');
    }
    if (StdDraw.isKeyPressed(KeyEvent.VK_RIGHT)){
        map.getPlayer().setDirection('r');
    }
}
```

The input keys are an attribute of a Stage class. This allows the game to have different keys of movement for every stage. The direction of the player image, however, is the same for every stage. The elephant looks right when the right key is pressed and left when the left key is pressed. This was done in order to give a confusing effect for stage 2 where left and right keys are reversed.

When moving sideways, the `map.moveplayer()` method simply moves the player in the given direction if there is no obstacle on the way.

When jumping, it only adds a vertical velocity to the player.

```

else if (isOnSurface(player.getX(), player.getY(), obstacles)){
    player.setVelocityY(stage.getVelocityY());
}

```

Vertical movement and velocity adjustments according to gravity are done in the `updatePlayerHeight()` method in Map class. When there is a vertical collision, the highest point of the object the player collides with is stored.

```

if (obstacle[3] > highestCollisionY) {
    if (player.getY() > obstacle[3]) {
        highestCollisionY = obstacle[3]; //top height
    }
    else{
        highestCollisionY = obstacle[1]-
player.getHeight(); //bottom height
    }
}

```

If the player is above the obstacle, the stored y value is the height of the top line of the obstacle, if the player is under the obstacle, the stored y value is the height of the bottom line of the obstacle minus the player's height. This is where the player's feet should stand after collision.

```

if (collisionDetected) {
    // Place player exactly on top of the highest obstacle hit.
    // Assumes player Y is the center, and Y increases downwards.
    // Player's bottom edge should be at highestCollisionY.
    player.setY(highestCollisionY + player.getHeight() / 2.0); //
Adjust Y to be just above the obstacle
    player.setVelocityY(0); // Stop vertical movement

```

The player's height is set to these values after collision.

## Deaths:

Whether the player is inside the spikes is checked in `movePlayer()` and `updatePlayerHeight()` methods and `restartStage()` method is called if the player touches the spikes

```

if (checkCollision(player.getX() + stage.getVelocityX(),
player.getY(), spike)) {
    restartStage();
}

```

```

public void restartStage() {
    isDead=true;
    player.setX(initialPos[0]);
    player.setY(initialPos[1]);
    player.setVelocityY(0);
    buttonPressNum=0;
    isDoorOpen=false;
    door[3]=doorTopPos;
}

```

The player returns to the start position, death count increments by one, the door closes, and button press count is restarted when `restartStage()` method is called.

## Custom Stage:

In my custom stage, I made the stage “Do you remember?” from the original game. In this stage, nothing except the player and the pipes is visible, but they still act as normal. This level can be seen in Figure 2.

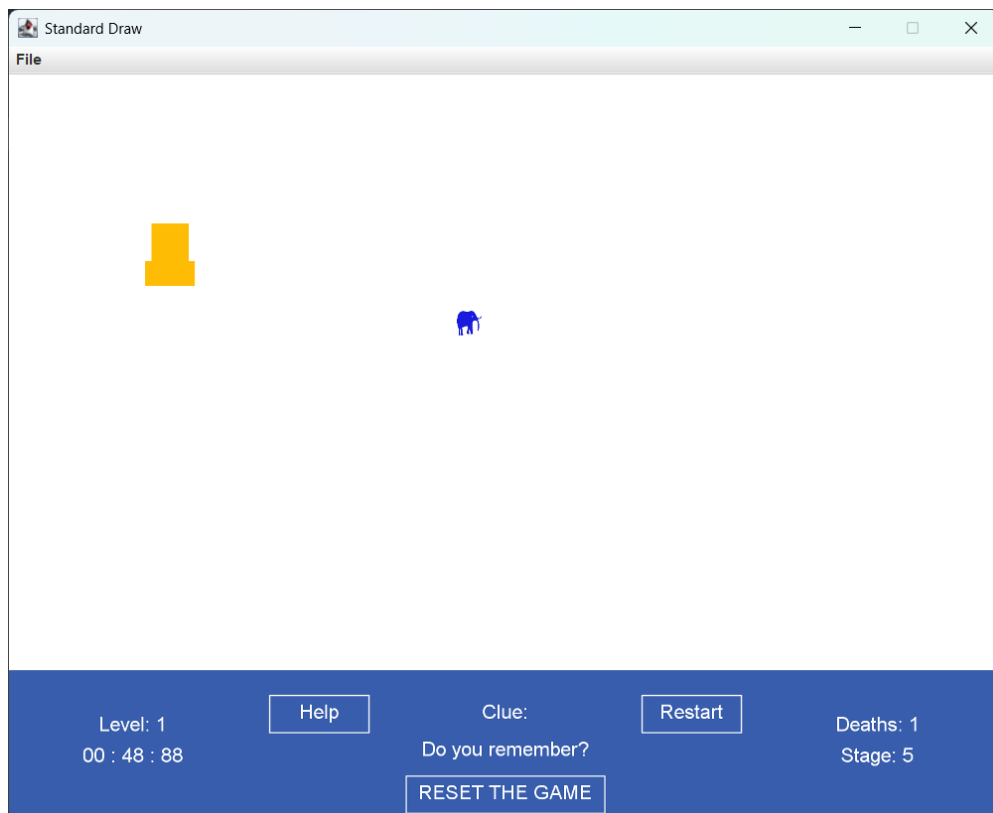


Figure 2: Custom Stage

## Finishing the Game:

```
while(stageIndex<stages.size())
```

The game ends when all the stages are done. This while loop in `play()` method which basically runs the game is exited. After the while loop, an end-game screen is displayed by the `play()` method. In this screen, the player can either quit the game by pressing Q or play the game again by pressing A.

```
while (true){  
    if (StdDraw.isKeyPressed(KeyEvent.VK_A)) {  
        resetGame();  
        break;  
    }  
    if (StdDraw.isKeyPressed(KeyEvent.VK_Q)) {  
        System.exit(0);  
    }  
}  
play();
```

After the end game screen is displayed, there is a while loop to get exit or restart input from the player. If the game is restarted, `resetGame()` method which resets the death count, stage number, and time. The loop is exited after this and `play()` method is called again, restarting the game.