**Jamil Shahverdiyev**

CDWCRZ

cdwcrz@inf.elte.hu

# Documentation to 3rd assignment.

## Task

**Labyrinth**

Create the Labyrinth game, where objective of the player is to escape from this labyrinth. The player starts at the bottom left corner of the labyrinth. He has to get to the top right corner of the labyrinth as fast he can, avoiding a meeting with the evil dragon. The player can move only in four directions: left, right, up or down. There are several escape paths in all labyrinths. The dragon starts off from a randomly chosen position, and moves randomly in the labyrinth so that it choose a direction and goes in that direction until it reaches a wall. Then it chooses randomly a different direction. If the dragon gets to a neighboring field of the player, then the player dies. Because it is dark in the labyrinth, the player can see only the neighboring fields at a distance of 3 units. Record the number of how many labyrinths did the player solve, and if he loses his life, then save this number together with his name into the database. Create a menu item, which displays a highscore table of the players for the 10 best scores. Also, create a menu item which restarts the game. Take care that the player and the dragon cannot start off on walls

## Analysis

### Important things to consider:

1. Dragon moves randomly until it reaches the wall
2. Player spawns at bottom left corner of the left
3. It shall reach the top-right corner of the map in order to proceed
4. If Dragon is in the neighboring field of player, the player dies.

### Visual Structure:

LabyrinthMain class to start the game;

LabyrinthDisplay class for initializing the game, menu, and highscore

LabyrinthPanel class for handling the game rules, mechanics, for drawing the board and etc.

HighScoreFrame class responsible for initializing the HighScorePanel

HighScorePanel class responsible for creating the table and visualizing the data of the best players.

### Algorithms:

Losing condition:If the player is next to the dragon (isNextToPlayer(x,y) method)

Winning condition: If the player is on the correct row and col (top right corner)

Restarting the game: FinishGame() method and initialization of the game once more using InitializeGame()

Showing the leaderboard: Initializing the HighScoreFrame panel

**Remark: The levels are created manually by me. I created 10 levels by changing the map in each level. Finishing all levels will result in a panel where player will write his/her name. Therefore, there is no level algorithm. The only thing worth mentioning is that the dragon spawns randomly in the top rows in order to not collide with the player at the start of the game.**

**Implementation:**

- The movement of the player:

Player can move using well-known WASD which calls movePlayer method which takes the directions as parameters. The method then checks if the player can move in that direction and if yes, then it changes the player's location in matrix.

- The movement of the dragon:

Dragon starts at top row of the matrix but spawns at the random column. It chooses a direction: (Random import) top, left, right, bottom and moves until it reaches the wall. After that the dragon chooses another direction.

MoveRandomly() function

- Winning condition

There is an exit in top-right corner of the map, the function isWin() is called when the player is on the appropriate location, that is number 6 on the matrix (ground is denoted as 0, wall as 1) if the method returns true, then the player proceeds to the next level.

- Losing condition

The player loses if he is in the neighboring field of the dragon. There are two methods to check that: isNextToDragon and isNextToPlayer. These methods check if the player loses. If the player loses the game ends with the finishGame() method which calls the panel so the player can register his name. After that the highScore table shows up.

## Details about the classes:

**Dragon Class:**

- Constructor:

Dragon(int[][] matrix): Instantiates the dragon within the specified game matrix, randomly positioning it on an accessible cell.

- moveRandomly() Method:

Moves the dragon randomly within the game matrix. If it encounters a wall or reaches the matrix boundaries, it changes its direction.

- isNextToPlayer(Point playerPosition) Method:

Checks if the dragon is positioned adjacent to the player, considering the player's current position.

- reset() Method:

Resets the dragon's position by randomly placing it on the top row of the matrix, avoiding occupied cells.

**Highscore class**

- Constructor:

HighScore(String name, int score, int time): Initializes a high score entry with the specified player name, score, and time.

- getName() Method:

Returns the player's name associated with the high score.

- getScore() Method:

Returns the score achieved by the player in the high score entry.

- getTime() Method:

Returns the time taken by the player to achieve the high score.

**HighScorePanel class:**

- Constructor:

HighScorePanel(ArrayList<HighScore> highScores): Initializes a panel displaying high scores using a JTable with player name, score, and time columns.

- isCellEditable(int row, int column) Method Override:

Prevents cell editing in the JTable by overriding the default behavior.

**Labyrinth class:**

- Constructor:

Labyrinth(int[][] matrix)`: Instantiates the `Labyrinth` class, creating instances of the `Player` and `Dragon` classes by passing the provided matrix as a parameter.

**LabyrinthDisplay Class:**

- Constructor:

LabyrinthDisplay(int[][][] matrices): Initializes the game display, adds a labyrinth panel, sets up a menu bar with options for starting a new game and viewing the leaderboard.

- keyPressed(KeyEvent e) Method Override:

Passes key events to the labyrinth panel for handling player movements.

**LabyrinthPanel Class:**

- Constructor:

LabyrinthPanel(int[][][] matrix): Initializes the game panel, loads images for the player, dragon, wall, and ground, and initializes the game with the first level.

- resetGame() Method:

Resets the game to the initial state and level.

- finishGame() Method:

Stops the game timer, prompts for the player's name, saves the score to the leaderboard, and opens a high score frame.

- initializeLevel() Method:

Initializes a new game level, handling dragon movement, player-won conditions, and updates the panel.

- initializeGame() Method:

Initializes the game, including setting up a timer for elapsed time and starting the first level.

- getPreferredSize() Method:

Returns the preferred size of the panel based on the level dimensions and cell size.

- paintComponent(Graphics g) Method Override:

Paints the game components, including the labyrinth map, player, dragon, and fog of war effect.

- keyPressed(KeyEvent e) Method Override:

Passes key events to the labyrinth's player for movement handling.

## LabyrinthMain Class:

Instantiates the new labyrinthDisplay by taking the matrices.

## Player Class:

- Constructor:

Player(int[][] matrix): Initializes the player with a matrix and sets the initial position.

- keyPressed(KeyEvent e) Method:

Handles keyboard input to move the player in four directions (WASD).

- keyTyped(KeyEvent e) and keyReleased(KeyEvent e) Methods:

Unused event handlers.

- movePlayer(int x, int y) Method:

Moves the player in the specified direction (x, y) if the new position is within the matrix boundaries and not blocked by obstacles.

- reset() Method:

Resets the player's position to its initial state.

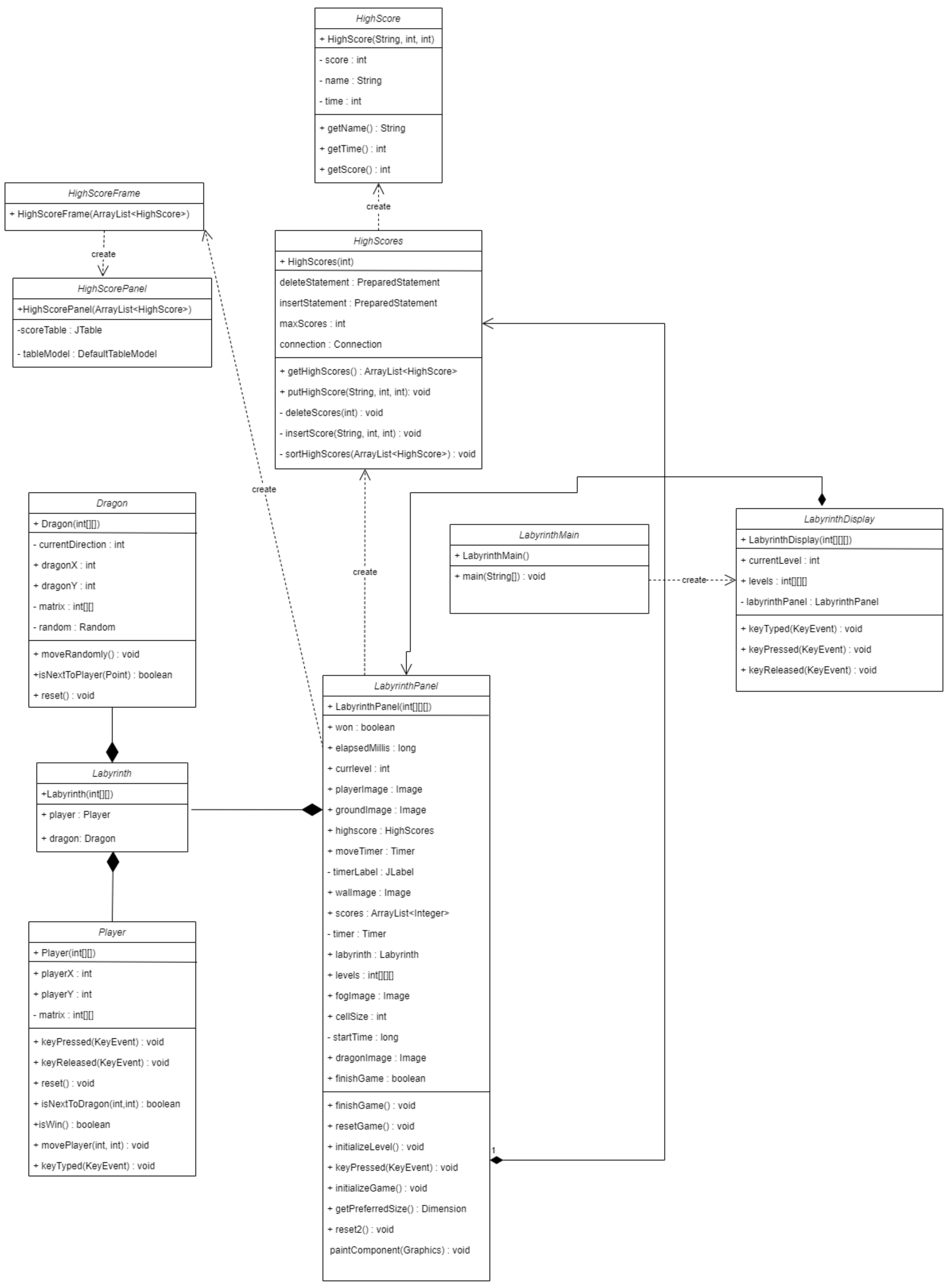
- isNextToDragon(int x, int y) Method:

Checks if the player is adjacent to the dragon based on the provided dragon's position (x, y).

- isWin() Method:

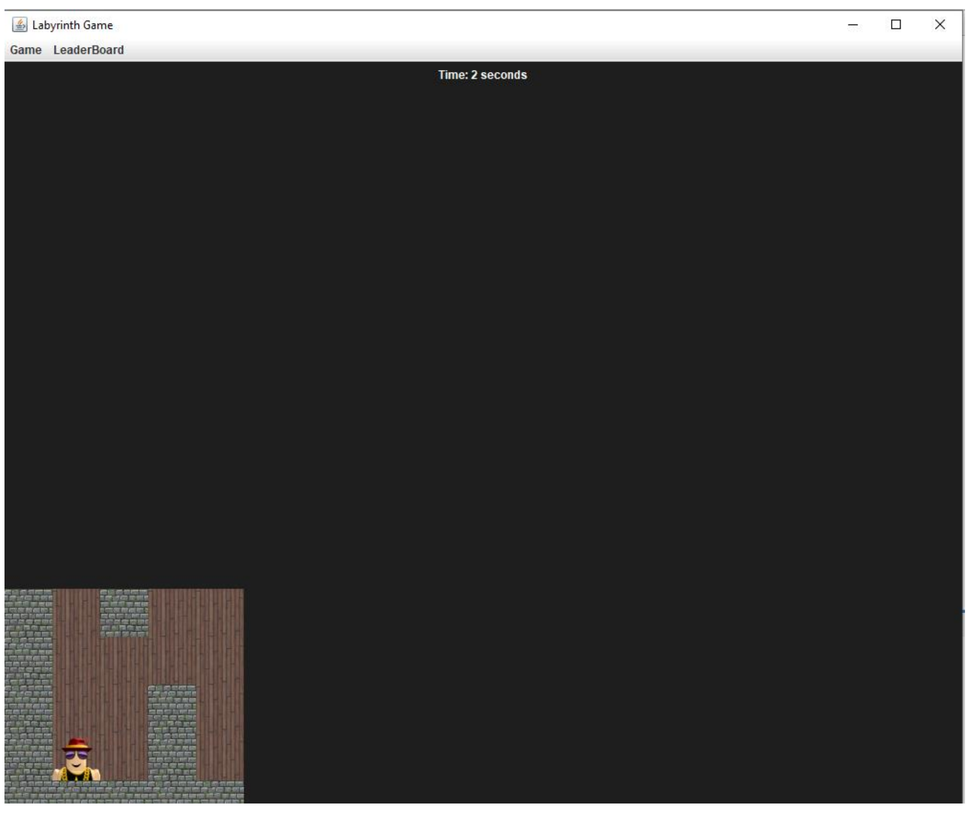
Checks if the player has won the level by reaching the exit (indicated by the value 6 in the matrix).

## UML CLASS DIAGRAM:

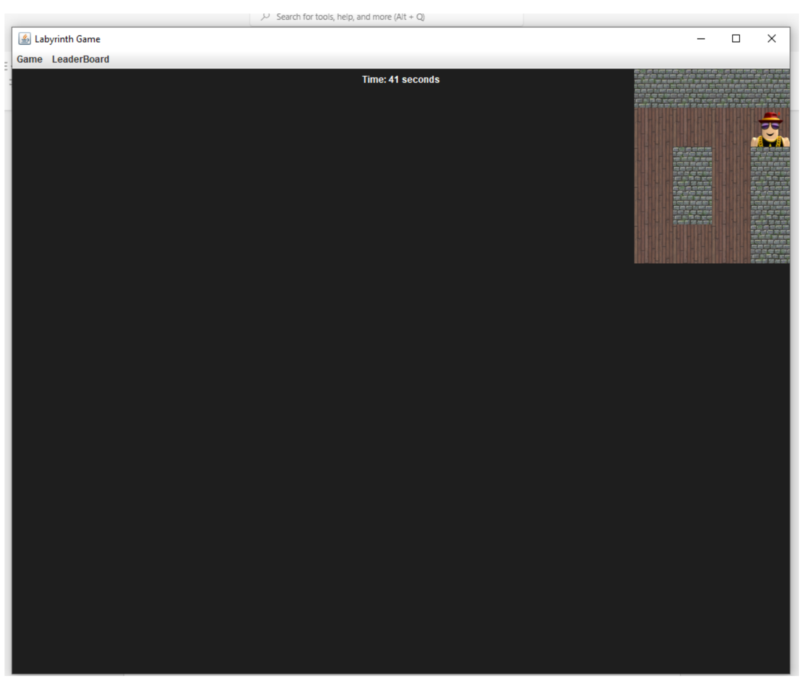


## Testing:

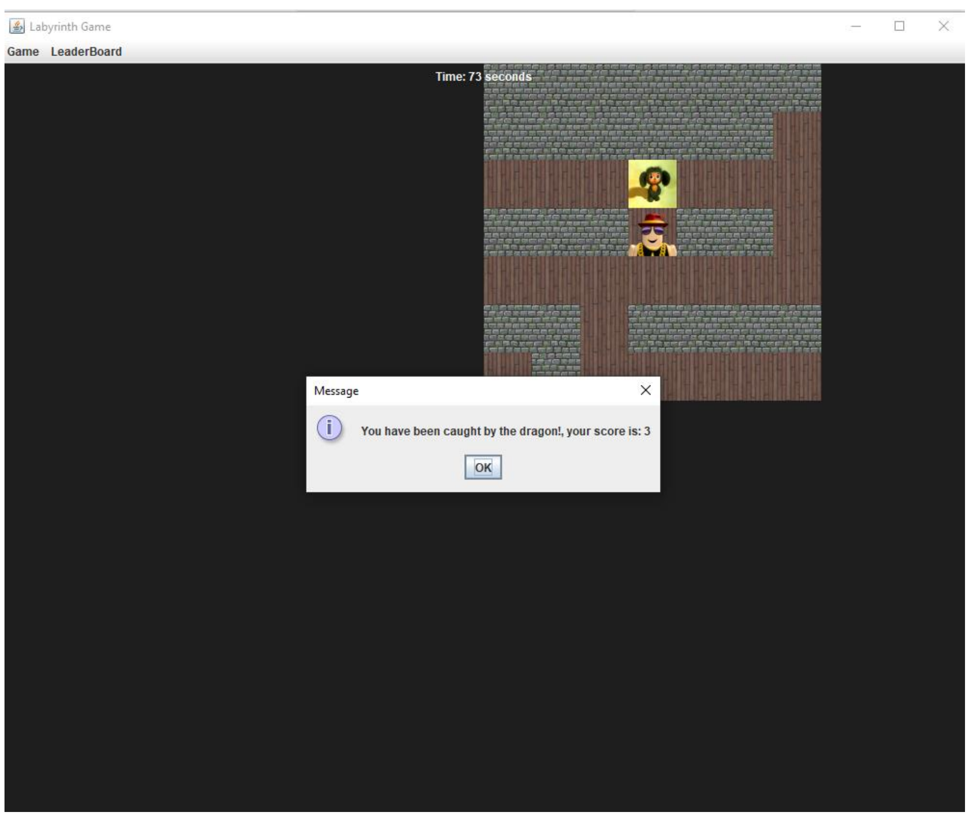
### Starting the game:



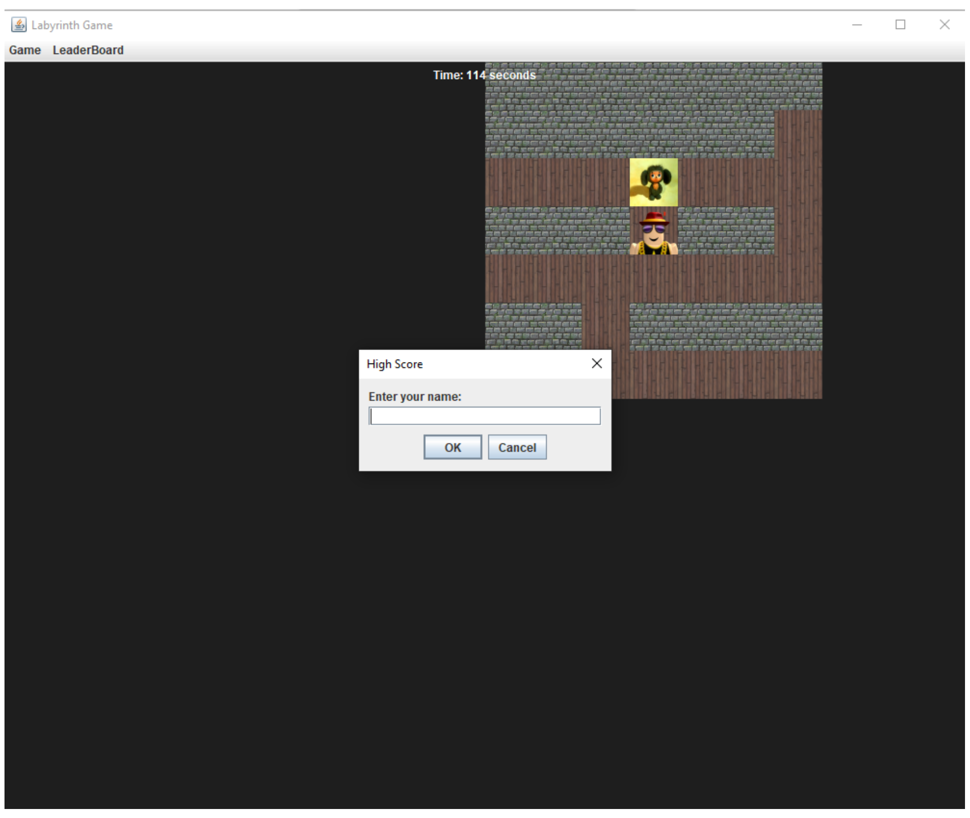
### Winning the level:



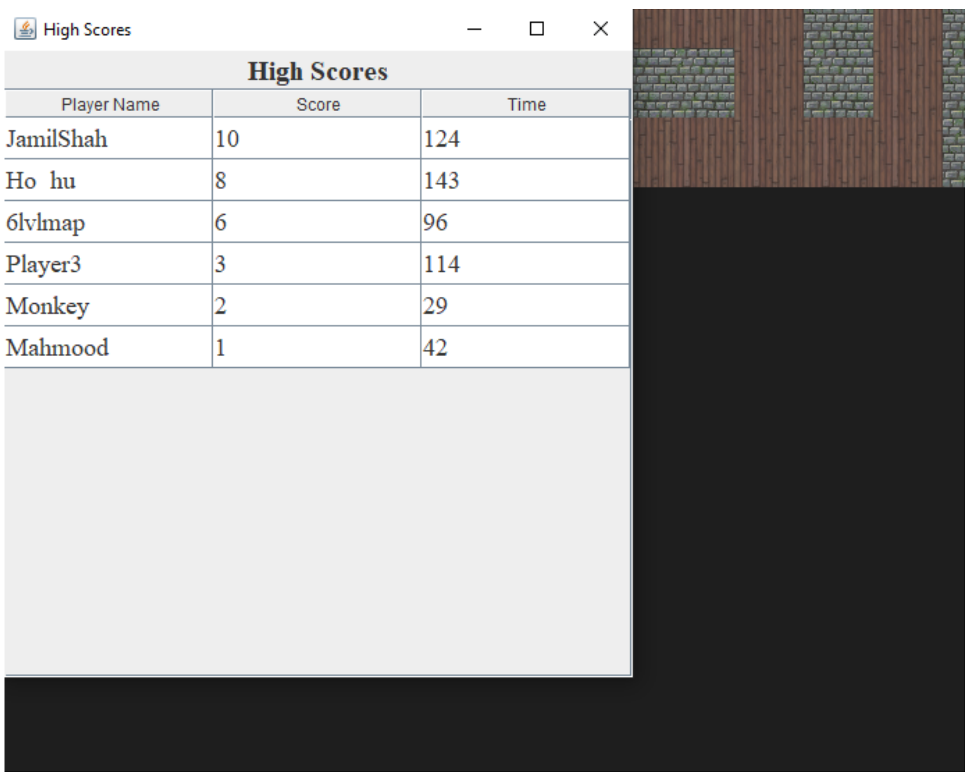
### Losing the game:



You have been caught by the dragon!, your score is: 3



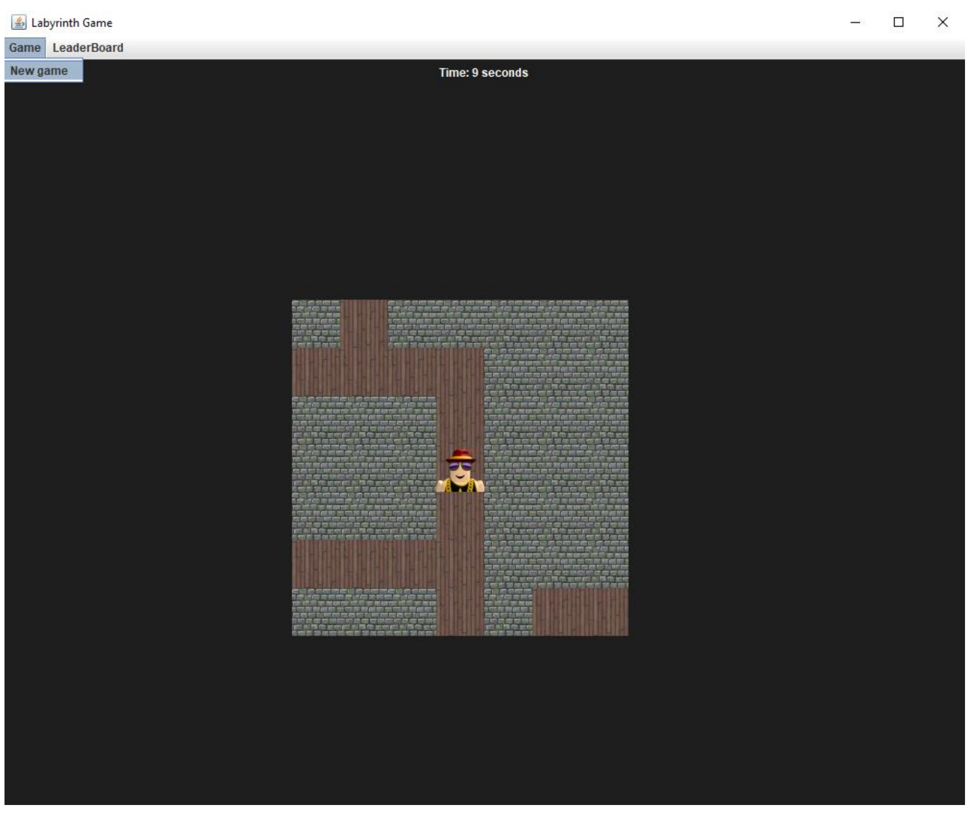
**Showing Highscores after losing or winning the game:**



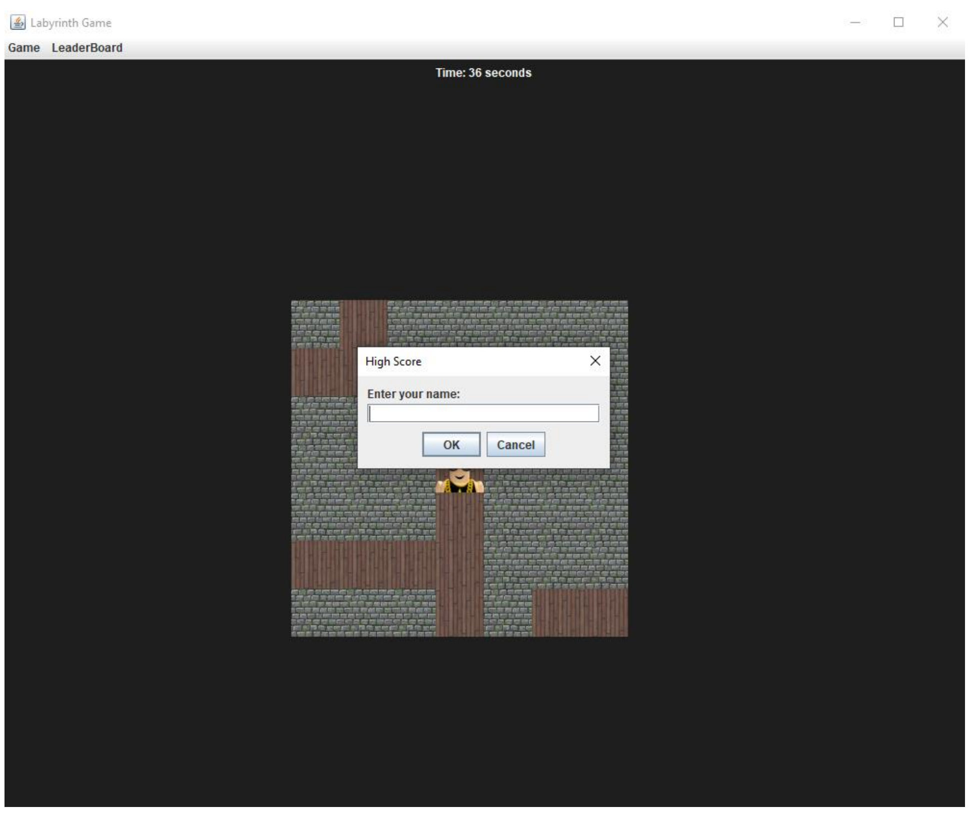
High Scores

| Player Name | Score | Time |
|---|---|---|
| JamilShah | 10 | 124 |
| Ho  hu | 8 | 143 |
| 6lvlmap | 6 | 96 |
| Player3 | 3 | 114 |
| Monkey | 2 | 29 |
| Mahmood | 1 | 42 |

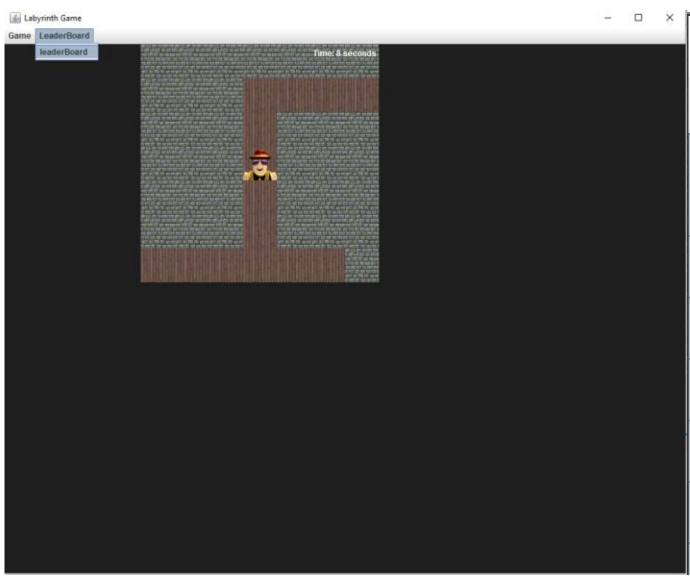
**Starting New Game in the middle of the current game**

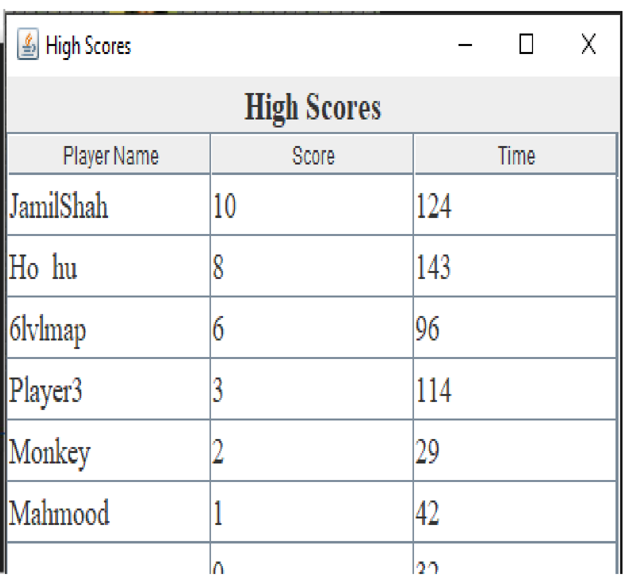


It asks for your name to store your current data:



**Leaderboards:**





High Scores

| Player Name | Score | Time |
|---|---|---|
| JamilShah | 10 | 124 |
| Ho  hu | 8 | 143 |
| 6lvlmap | 6 | 96 |
| Player3 | 3 | 114 |
| Monkey | 2 | 29 |
| Mahmood | 1 | 42 |
|  | 0 | 32 |