Name: Soe Thet Naing

Neptun ID: x0flxy

Assignment: 3

Task No. 2

Task Description

Snake

We have a rattlesnake in a desert, and our snake is initially two units long (head and rattler). We have to collect with our snake the foods on the level, that appears randomly. Only one food piece is placed randomly at a time on the level (on a field, where there is no snake). The snake starts off from the center of the level in a random direction. The player can control the movement of the snake's head with keyboard buttons. If the snake eats a food piece, then its length grow by one unit.

It makes the game harder that there are rocks in the desert. If the snake collides with a rock, then the game ends. We also lose the game, if the snake goes into itself, or into the boundary of the game level.
In these situations show a popup message box, where the player can type his name and save it together with the amount of food eaten to 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.

Description of each Method

SnakePanel.java

setToRadomDirection(): Initializes the snake's movement direction at random at the start of each level.

restart(): Reboots the game to its initial state.

startGame(): Activates the game's main loop.

paintComponent(Graphics g): Renders the game's visual elements on the interface

doesCollideWithSnake(): Evaluate if a snakes collides with a rock or an apple

doesCollideWithRocks(): Assesses if there is a collision between the snake or the apple and any obstacles.

generateObstacles(): Construct obstacles within the game area.

draw(Graphics g): Illustrate the game's component on the screen.

newApple(): Position a new apple in the random location.

move(): Propels the snake's movement in the game space.

checkApple(): Inspects for apple consumption by the snake and adjusts the score and game level

checkCollision(): Determine if the snake has impacted itself, the game boundaries or the obstacles.

gameOver(): Concludes the gameplay and presents an input dialogue.

actionPerformed(ActionEvent e): Handles events during the game's operation.

keyPressed(KeyEvent e): Registers keyboard inputs for controlling the snake's direction.

MenuGUI.java

paintComponent(Graphics g): draw image of the given picture

HighScore.java

getName(): returns the player name

getScore(): returns score of the player

compareByScore(SnakeScore hs1, SnakeScore hs2): compares two players scores and returns the best score.

getHighScores(): firstly executes sql query and adds players into list and returns them sorted.

insertScore(String name, int score): inserts player which is given as a parameter into database.

putHighScore(String pName, int score): gets sorted lists of players and compare its size with the max number of players.

sortByScore(ArrayList<SnakeScore snakescore>): sorts list of players by their score

deleteScores(int score): deletes score and modifies the table

getColumnNamesArray(): returns ID, Name, Score of the player

getDataTable(): creates table of the players in matrix format

SnakeGameGUI.java

addEventListeners(JMenuItem exit, JMenuItem restart, JButton newGameButton, JButton scoresButton):

exit .addActionListener(new ActionListener()): exits the game

restart.addActionListener(new ActionListener()): starts the new game(menu).

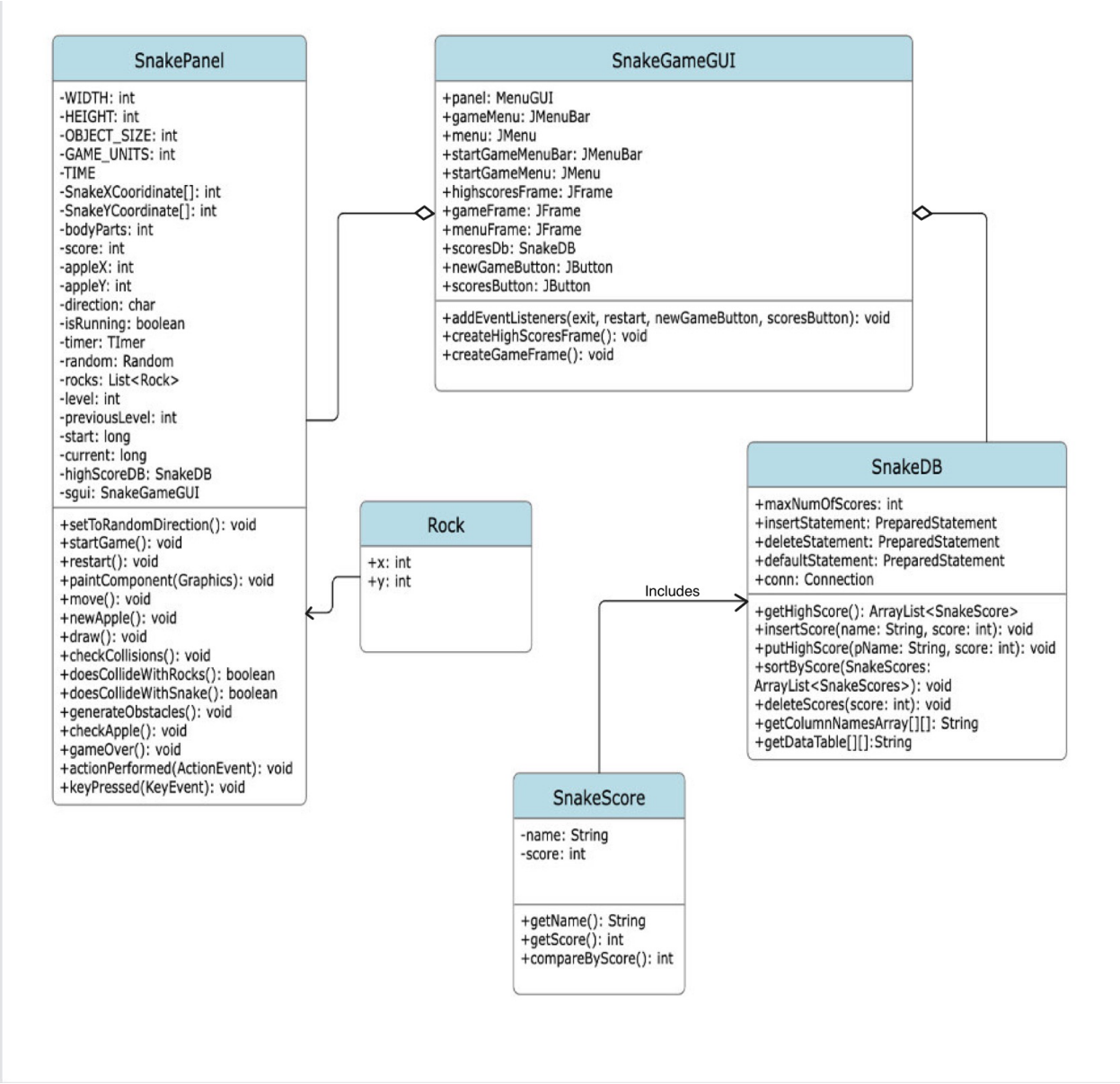
newGameButton.addActionListener(new AcitonListener()): starts the game(button).

scoresButton.addActionListener(new ActionListener()): opens score frame and shows best players.

createHighScoresFrame(): creates well-designed form for the high scores table.

createGameFrame(): creates a frame for playing.

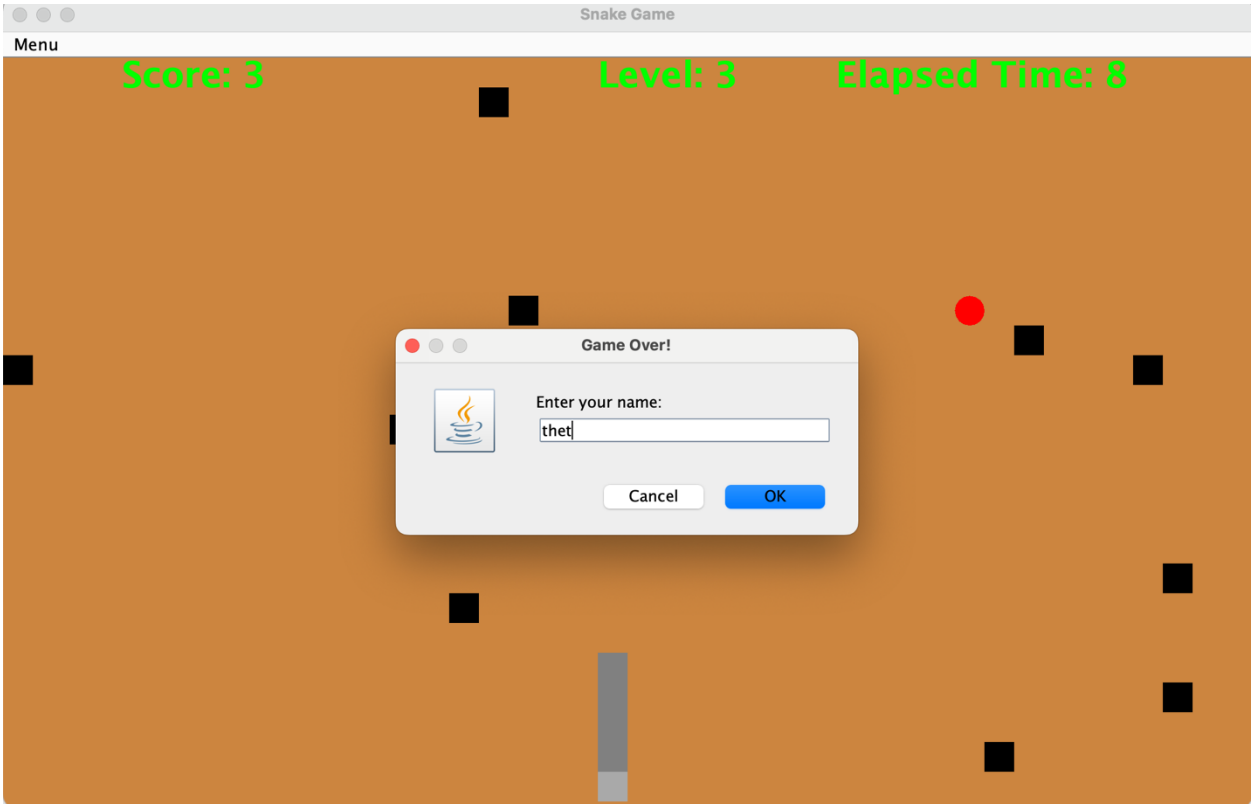
UML Class Diagram



Test Cases

We will test if the scores for a single player whether the scores will be overridden if the next scores is more than the previous one and remain the same if not.

We will store the scores for a player named “thet” which is 3.

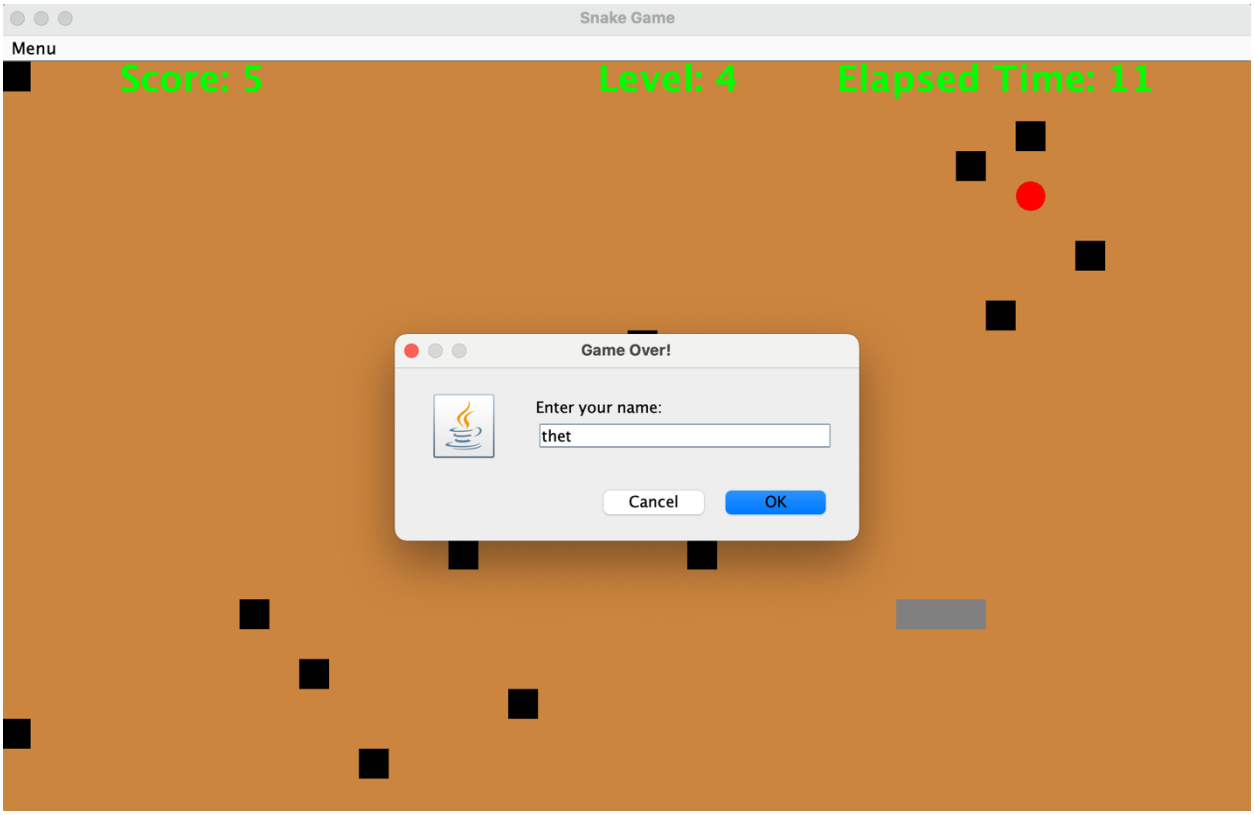


We can see the scores table as follows.

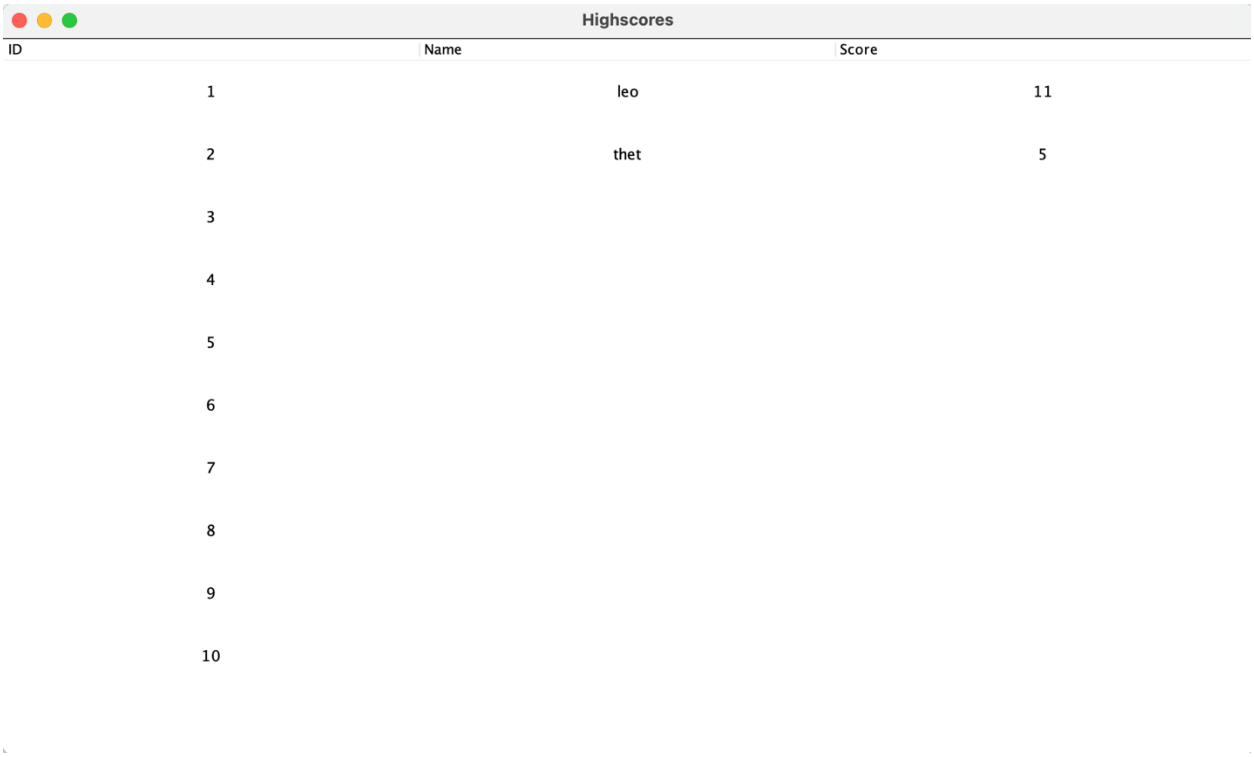
Highscores

ID	Name	Score
1	leo	11
2	thet	3
3		
4		
5		
6		
7		
8		
9		
10		

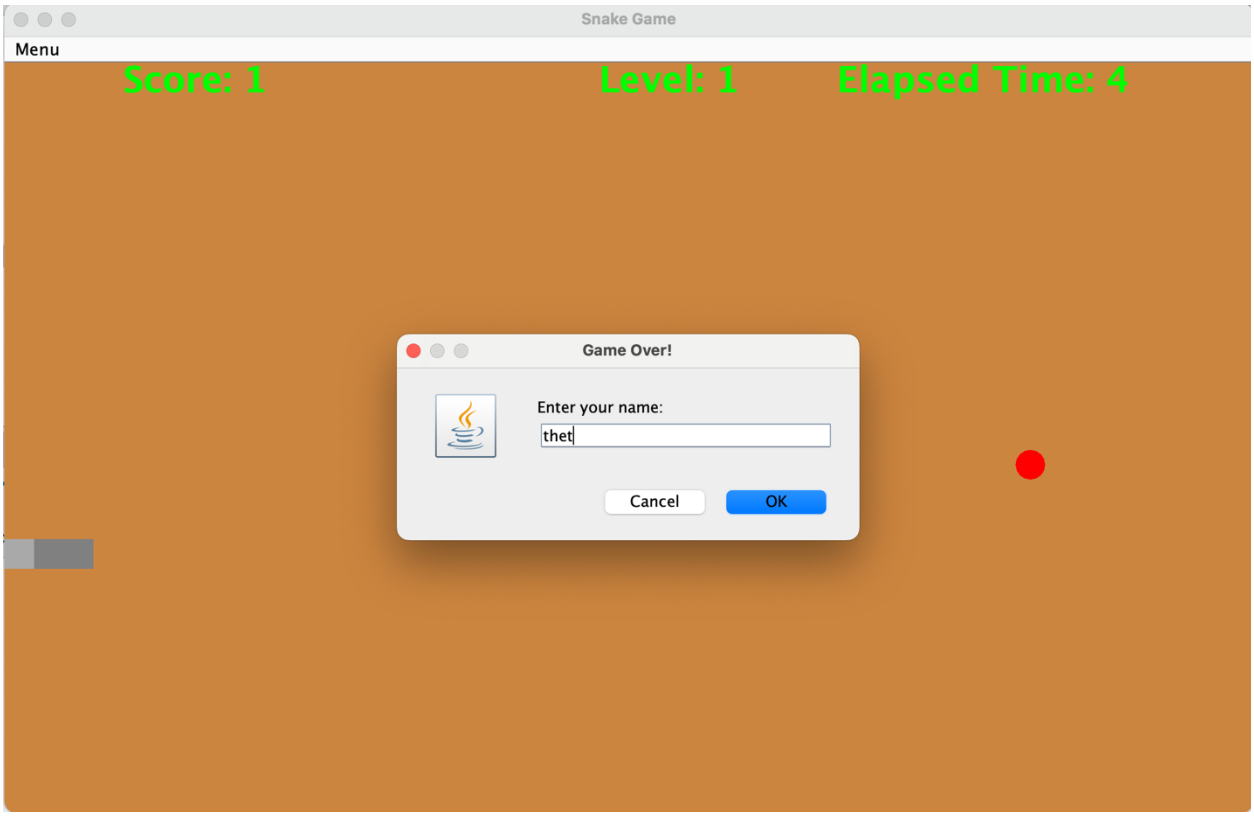
After that, the same player will play the game again and got 5 which should be overridden.



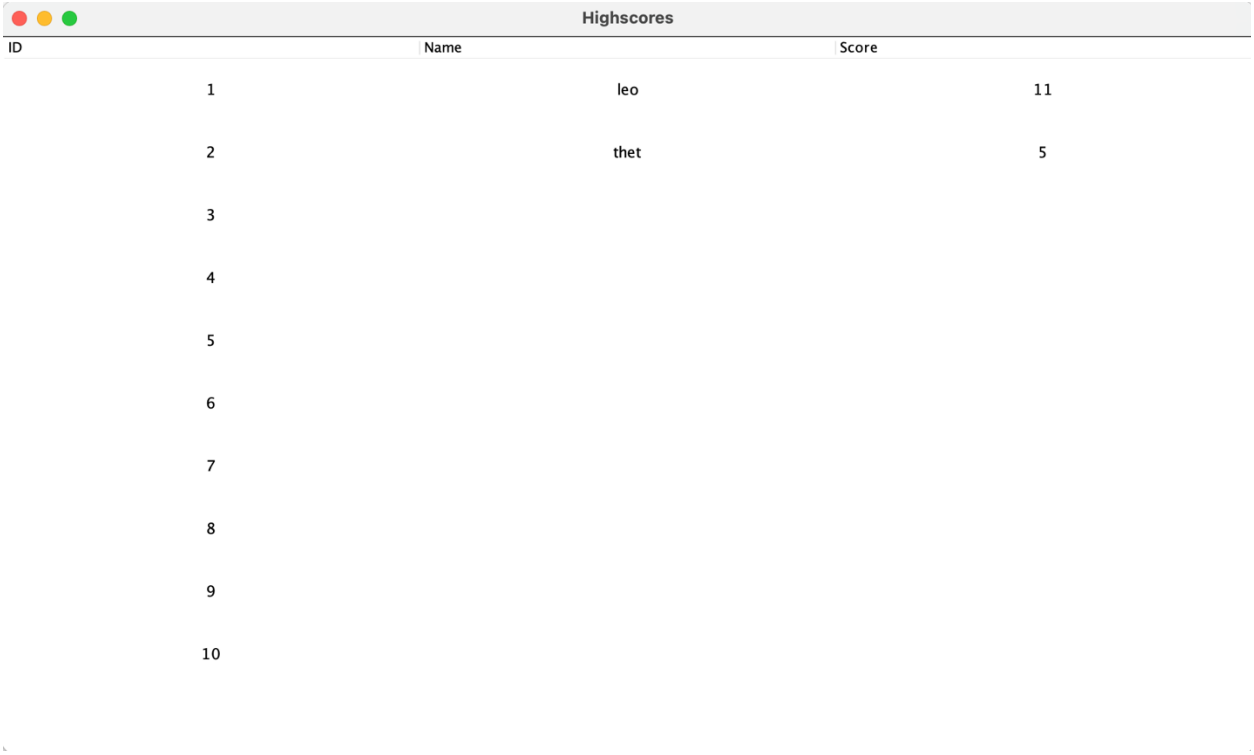
We can see that in the table, the scores are overridden.



After that, the same player will play the game again but got the lower scores this time.

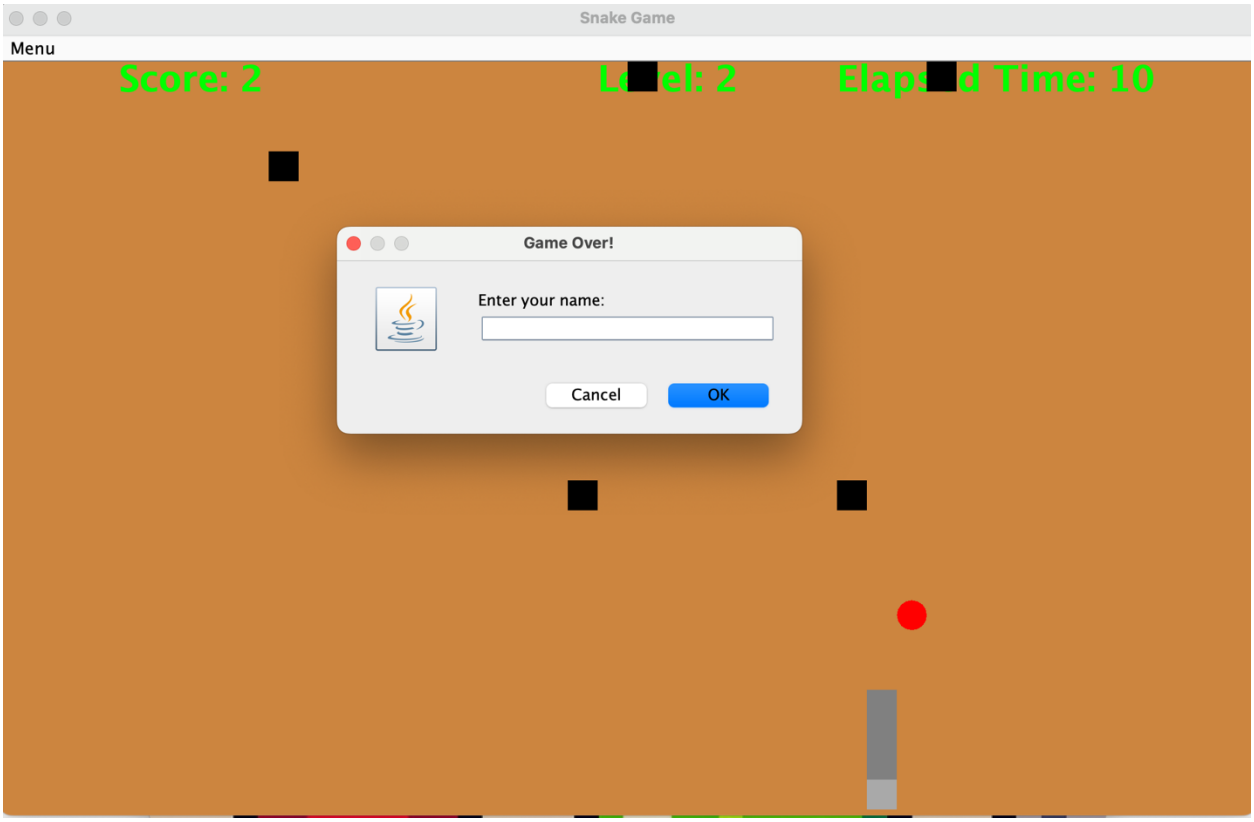


We can see that the scores remain the same in the table.

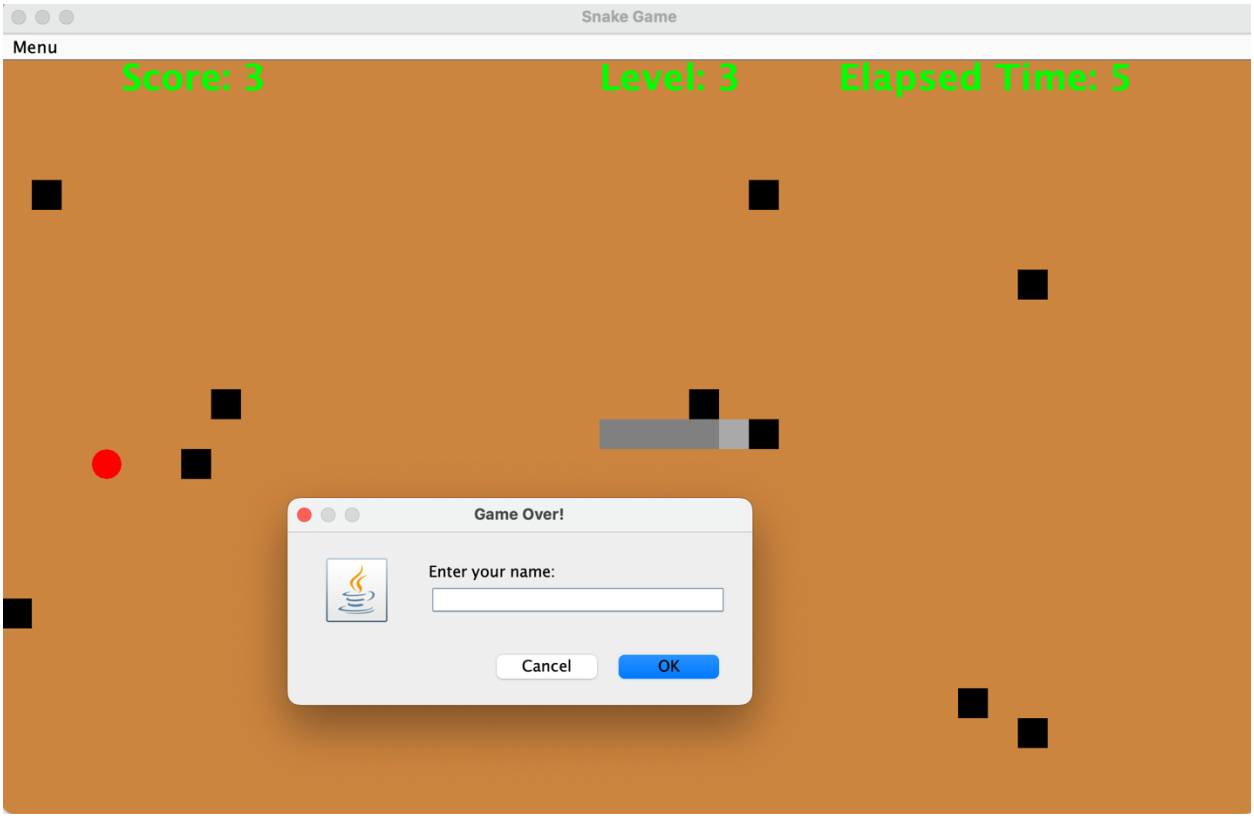


Now, we will test the game over stages.

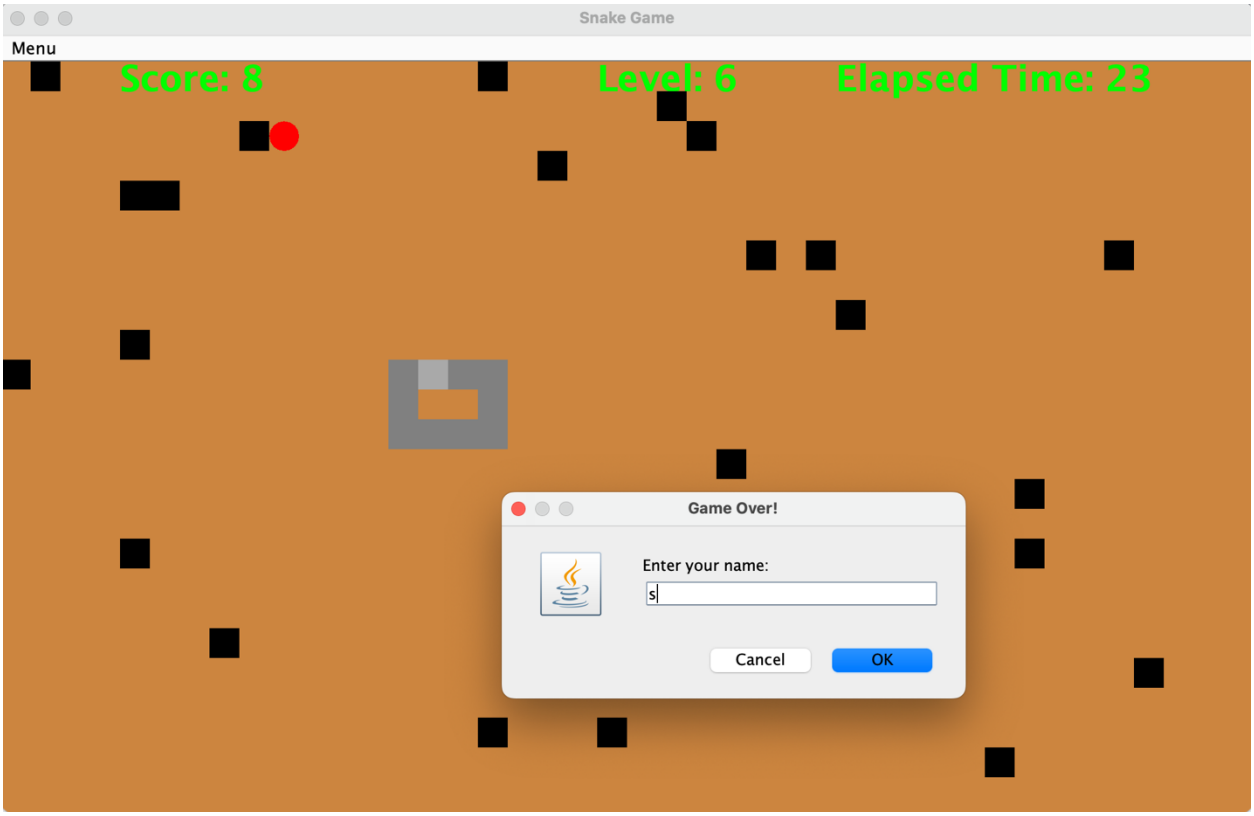
The following stage is against the frame. The message pops up.



Now, we will test another condition which is against the obstacle.



Finally, we will test the game over stage against its own body.



Now, we will test the end of the game. The game will end once its score reaches 11 as the picture below.