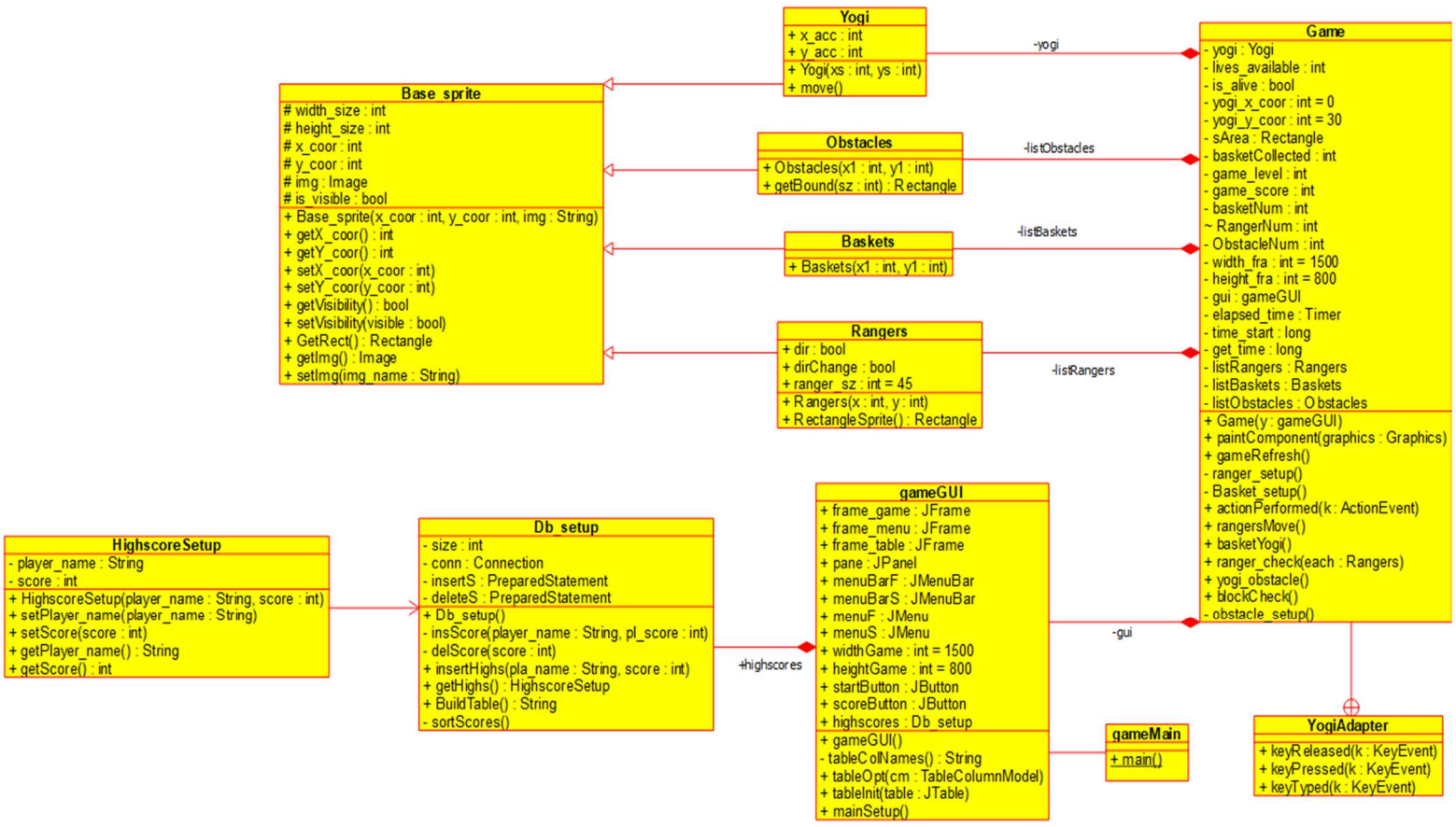
Munkhtenger Munkh-Aldar **3rd assignment/1st Task** **07 Dec 2022**
WSR292
WSR292@inf.elte.hu

Task – Yogi Bear

Yogi Bear wants to collect all the picnic baskets in the forest of the Yellowstone National Park. This park contains mountains and trees, that are obstacles for Yogi. Besided the obstacles, there are rangers, who make it harder for Yogi to collect the baskets. Rangers can move only horizontally or vertically in the park. If a ranger gets too close (one unit distance) to Yogi, then Yogi loses one life. (It is up to you to define the unit, but it should be at least that wide, as the sprite of Yogi.) If Yogi still has at least one life from the original three, then he spawns at the entrance of the park. During the adventures of Yogi, the game counts the number of picnic baskets, that Yogi collected. If all the baskets are collected, then load a new game level, or generate one. If Yogi loses all his lives, then show a popup messagebox, where the player can type his name and save it 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.

UML



Analysis

In order to implement the Yogi Bear game, the task should be separated into 3 parts:

- Model
- Persistence
- View

In the Model, the logic of the game should be implemented. Concrete objects of the game are Yogi, Obstacles, Rangers, and Baskets and all of them are sprites. Therefore, each of them can inherit from one class as all of them have coordinates, images, and visibility. Now that 5 classes should be implemented, the interaction of each class with each other should be handled in a separate class called Game. The Game class will be the engine of the game. All sprites would be generated from their classes but to make the game playable, the sprites should be moveable, and this generates encounters. The following interactions should be handled in Game class:

- Yogi tries to move outside of the border of the game.
- Yogi encounters a ranger.
- Yogi encounters an obstacle.
- Ranger encounters an obstacle.
- Moving ranger encounters the border of the game.
- Yogi encounters a basket.
- Yogi has no more life.
- All baskets are collected in a level.
- Initialize Yogi each time until Yogi has no more life.
- Obstacles generated randomly.
- Rangers generated randomly.
- After collecting every basket in a game, add more difficulty.
- Player reaches level 10… etc.

For the Persistence part, we need to handle the database and its queries of it. As the task requires that there should be a high score table of the players for the 10 best scores, using MySQL for storing the player names and scores would be convenient. For that, we need to implement a database setup class and a class for storing concrete player data. Auxiliary class that realizes 1 concrete player will store player name and player score. In the database setup class, we need to handle the following things:

- Execute a query to create a table in MySQL Workbench.
- Connecting Java project to the MySQL database.
- Insert player data into the database.
- Delete player data from the database.
- Sort 10 players' data in the database.

- Create a java matrix view by querying from the database.
- Delete the least-scored player from the database if there are more than 10 players.
- Putting high scores in sorted order after adding player data.

For the View part, we should take care of the game's design and layout. Multiple frames should be implemented as the game will have a main menu frame, game frame, and table frame. Also, in the game frame, there should be a menu option to restart the game. Once the player clicks on the restart option, the game should be restarted. In the main menu window, there should be a connecting option to change into the game or to the database. Adding 2 buttons in the main menu frame is the ideal choice. The following things should also be taken care of:

- Height and width of the game frame.
- Adding action listeners to the buttons.
- Generating a high score table with the matrix view of MySQL database.
- Adding the high score to the table frame.
- Taking care of the views of the table such as alignment, and visibility.
- Adding an option pane to the game.
- If the player dies or wins the game, show up option pane to get the player's name.
- Ability to restart the game.

Taking care of the things mentioned above can bring you a game where Yogi collects baskets and rangers move vertically and horizontally and some obstacles block Yogi. Once Yogi has no more life left, the player enters the name, and it is stored inside a MySQL table. If the player wants to check the table, they can check through the game by clicking on the high score button. Then the table view of the database will be queried and shows a sorted table through the game. If the player wants, they can exit the game or save their name by winning or dying. To make the game harder and long paced, Yogi will move very slowly with the keys of the keyboard.

Description of each method

Game Class

- ***Constructer(gameGUI):***
  Constructor for Game class that initializes the game to be playable.
- ***paintComponent (Graphics):***
  Function to paint the sprites and frames of the game initializes option panes and saves the name of the player into the database.
- ***gameRefresh():***
  Function that checks game level and the level is below 10, the game gets harder else it shows a pane to the winner.

- ***ranger_setup():***
  Function for generating and placing obstacles randomly.
- ***Basket_setup():***
  Function for generating and placing baskets randomly.
- ***actionPerformed(ActionEvent):***
  Function that handles the game's action when the player triggers any kind of event.
- ***yogiMove():***
  Function for handling Yogi's movement checks for the boundaries.
- ***rangersMove():***
  Function for handling Ranger's movement and direction change when frame boundaries encounter.
- ***basket_yogi():***
  Function for handling when yogi touches the basket.
- ***ranger_check(Rangers):***
  Function that handles Ranger coordinates direction by checking for its directions.
- ***yogi_obstacle():***
  Function that handles situations where Yogi runs into obstacles.
- ***blockCheck():***
  Function that handles when ranger encounters an obstacle and yogi touches ranger and yogi has no more life.
- ***obstacle_setup():***
  Function for generating and placing randomly.

YogiAdapter Class

- ***keyReleased(KeyEvent):***
  Function to check which key is released and stops Yogi's movement respectively.
- ***keyPressed(KeyEvent):***
  Function to check which key from W, A, S, D is pressed and commences yogi's movement respectively.
- ***keyTyped(KeyEvent):***
  Function to typing.

Base_sprite Class

- ***Constructor(x,y,img):***
  Constructor for Base_sprite class that sets coordinates, width, height, image, and visibility.
- ***getX_coor():***
  Get x-coordinate.

- ***getY_coor():***
  Get y-coordinate.
- ***setX_coor(x):***
  Set x-coordinate.
- ***setY_coor(y):***
  Set y-coordinate.
- ***getVisibility():***
  Get visibility of the sprite.
- ***setVisibility(visibility):***
  Set the visibility of the sprite.
- ***GetRect():***
  Getter of rectangle bound of the sprite.
- ***getImg():***
  Getter for the image of the sprite.
- ***setImg(img):***
  Setter for the image of the sprite.

Baskets Class

- ***Constructor(x,y)***:
  Constructor for Baskets class. It gets x and y coordinates to construct the basket.

Obstacles Class

- ***Constructor (x,y):***
  Constructor for Obstacles class that initializes Obstacle with given coordinates and image.
- ***obstacleRect():***
  Function for getting bounded rectangle obstacle.

Rangers Class

- ***Constructor (x,y):***
  Constructor for Rangers class that initializes Ranger with given coordinates and image.
- ***RectangleSprite():***
  Function for handling movement of Ranger when moved and returns the coordinate-altered rectangle.

Yogi Class

- ***Constructor (x,y):***
  Constructor for Yogi class that initializes Yogi with given coordinate and image.

Db_setup Class

- ***Constructor:***
  Constructor for the Db_setup class that makes a connection with the SQL server.
- ***insScore(player_name, pl_score)***:
  Function for inserting the player details into the database.
- ***delScore(score):***
  Function for deleting the player details from the database.
- ***insertHighs(player_name, pl_score)***:
  Function for inserting player details into the database when there are more than 10 players.
- ***getHighs():***
  Function for querying all table data from the database and returns the sorted players' data.
- ***BuildTable():***
  Function for constructing string matrix and putting each player's data in it.
- ***sortScores(<scores>):***
  Function for sorting scores by comparing them individually.

HighscoreSetup Class

- ***Constructor (player_name, score):***
  Constructor for HighscoreSetup class.
- ***getPlayer_name():***
  Getter for the player name.
- ***setScore():***
  Getter for the player score.
- ***setPlayer_name (player_name):***
  Setter for the player name.
- ***setScore(score):***
  Setter for the player score.

gameGUI Class

- ***Constructor (player_name, score):***
  Constructor for gameGUI class that constructs the main view of the game.
- ***tableColNames():***
  Returns each column name of the table.
- ***tableOpt (TableColumnModel):***
  Function for centering each cell data alignment the table.
- ***tableInit (JTable):***
  Function for dealing with the table view.
- ***mainSetup():***
  Function for initializing game on the frame and adding menubar and menu items.

gameMain Class

- ***main():***
  Main for running the game.

Implementation of the game

Generating baskets

By utilizing the function ***random***, each basket generates randomly throughout the frame but generates not near to the 4 boundaries of the frame. Therefore, the game is solvable by the player at any point.

Generating rangers

By utilizing the function ***random***, each ranger generates randomly throughout the frame and moves horizontally and vertically depending on their randomly selected direction.

Generating obstacles

By utilizing the function ***random***, each obstacle generates randomly throughout the frame, and depending on their random number between 1 and 2, the obstacle can be a tree or rock but their bounding rectangle will be the same.

Button for starting the game

If this button is clicked, a new game frame will be created and on that frame, the game will be started with the menu on the top left side.

Button for accessing the high score table

If this button is clicked, a new high-score table frame will be created and on that frame, the visually sorted representation of the MySQL database will be shown in a table.

Button for restarting the game

When the game is already started, there will be a menubar that has a menu item to restart the game. Once it is clicked, the game frame will be disposed of, and the new game frame appears again.

Keyboard listener for moving Yogi

By extending TAdapter Class, the methods ***keyReleased, keyPressed, and keyTyped*** will be overridden and these methods listen to when keys W, A, S, D are pressed Yogi moves one unit accordingly, and when these keys are released Yogi stops its movement.

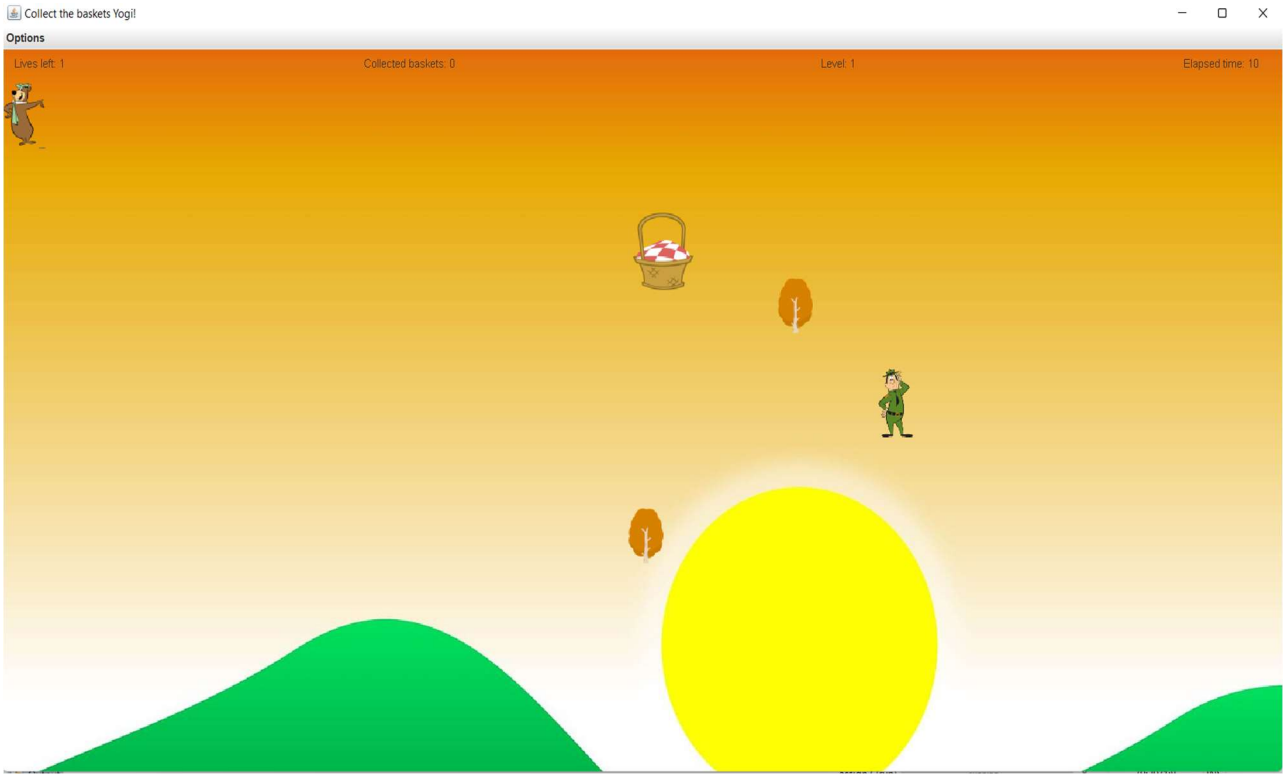
Action listener for the game

If any kind of action event triggers in the game, Yogi's available lives get checked first to stop the timer if Yogi has no more life. Following this, check for the keyboard listener to move Yogi in the game. Then, check for baskets if they are collected or not. If all of them in a level are collected, refresh the game with more baskets, rangers, and obstacles to make the game harder. Next, the rangers move horizontally and vertically and check if Yogi collected the basket to toggle the visibility of the basket.

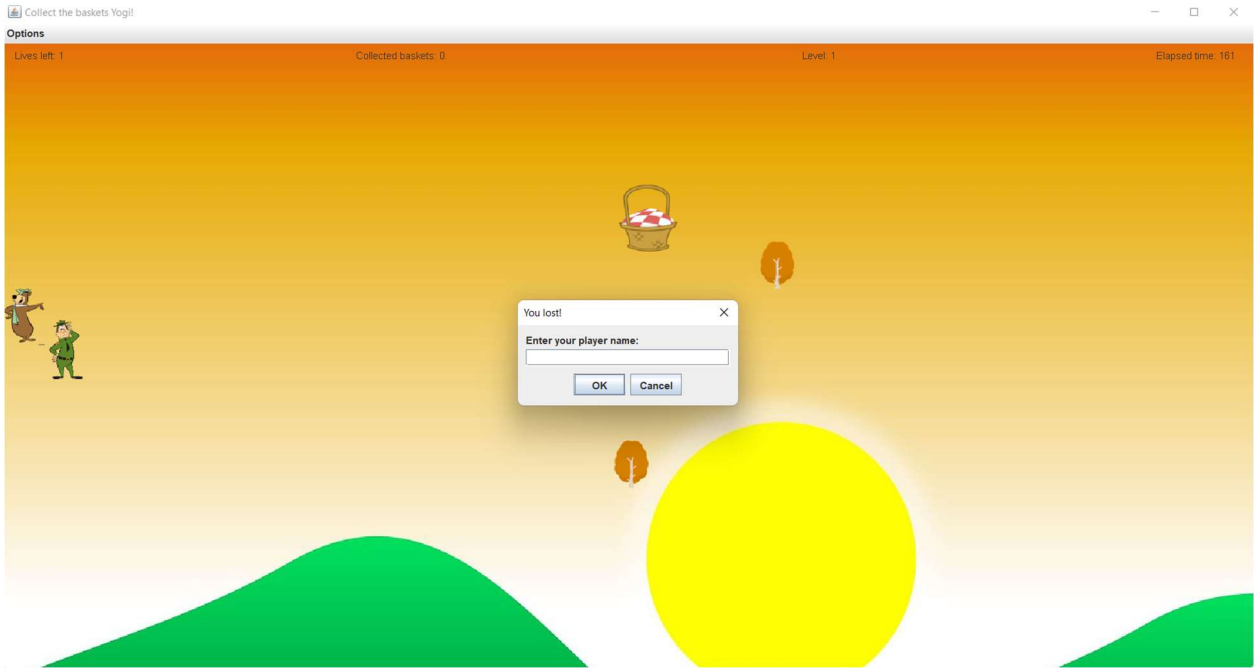
Then, check if Yogi encountered the obstacles or ranger encountered the obstacles. If yes, Yogi cannot get blocked by obstacles and the same goes for ranger. However, the ranger changes its direction when an obstacle is encountered. Lastly, repaint everything that is drawn with the help of ***paintComponent*** method to smoothly play the game.

Tests

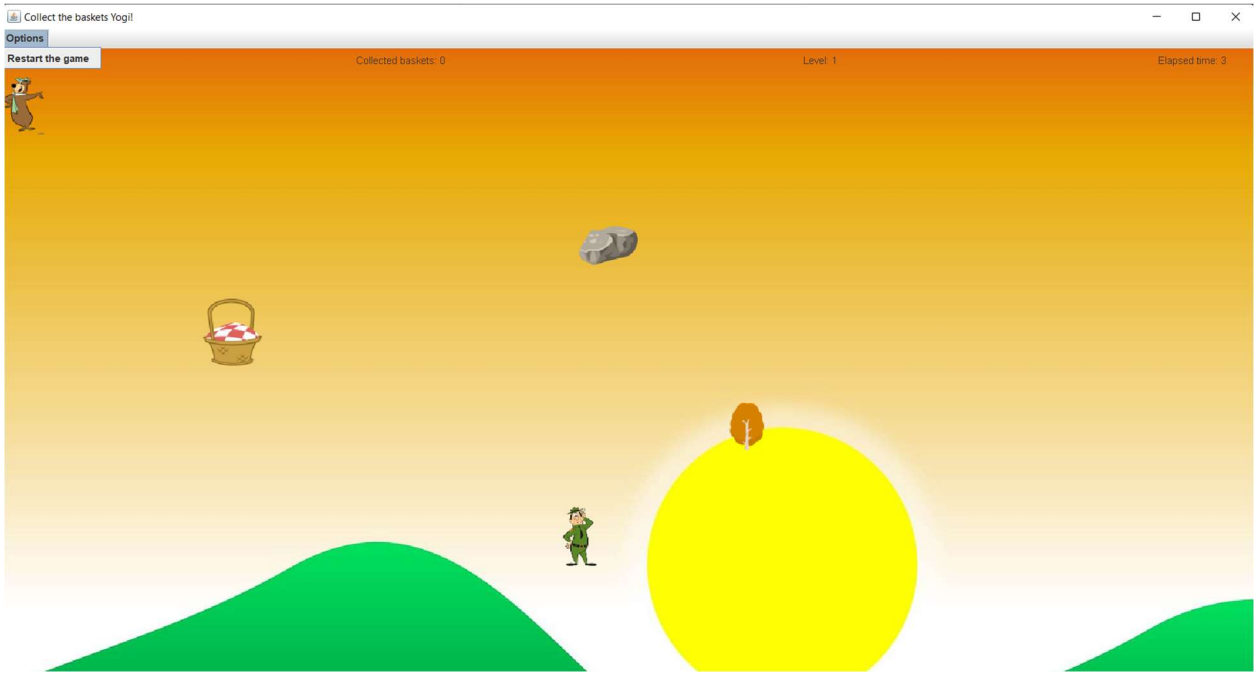
1. If Yogi encounters with the ranger, check for Yogi's available life and if it is more than 1, deduct 1 life and put Yogi in his starting position.



2. If Yogi encounters the ranger and Yogi has no more life left, the player has lost the game and option panel appears to get the players name to put it in the highscore table.



3. The restart menu item disposes the current game frame and creates a new game frame which restarts the game.

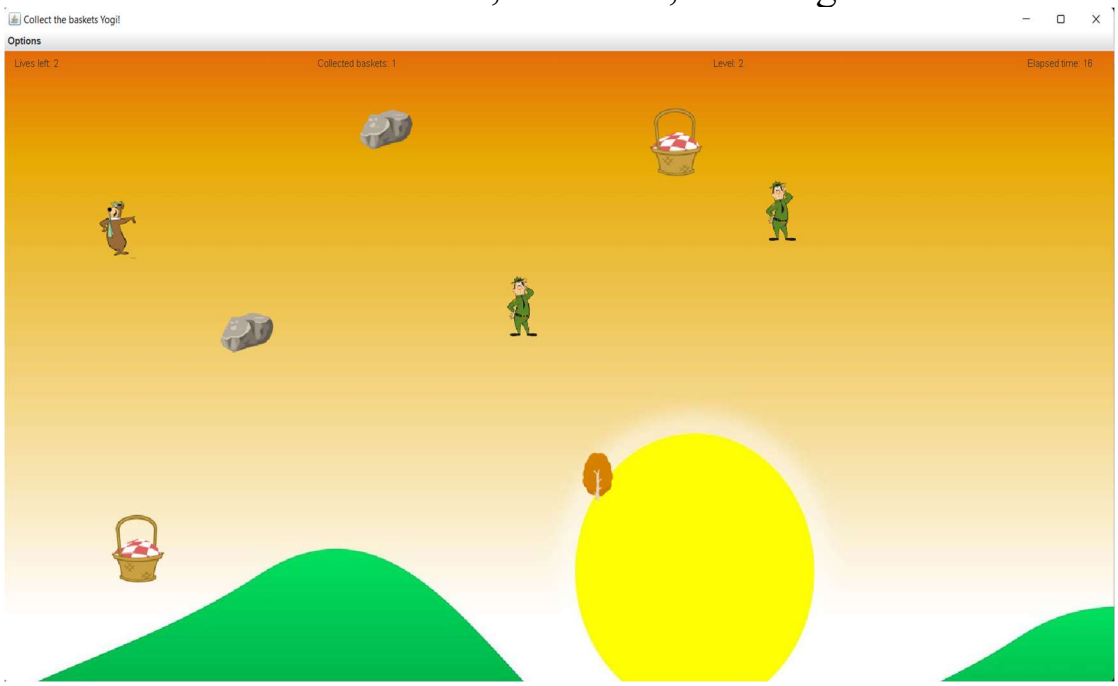


4. Check if the player's name and score is added to the highscore table in a sorted way as it is high score table.

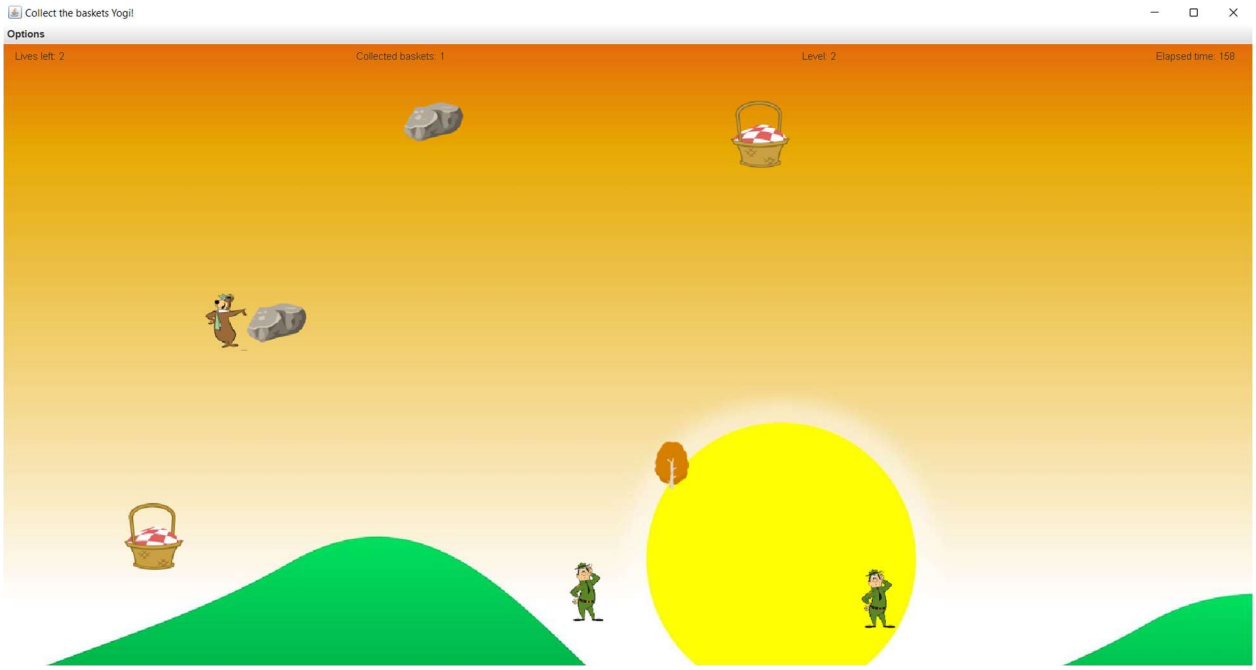
Highscore board

Rank	Player name	Baskets score
1	bb	24
2	a	11
3	2nd	8
4	hello	5
5	Munkhtenger	2

5. Check if Yogi collects all the basket in a level, the game starts a new level with more baskets, obstacles, and rangers.



6. Check if Yogi can't go through obstacles when they are encountered with each other.



7. Check if rangers can't go through obstacles when they are encountered with each other and changes direction.