Made by: Ebrahim Al-Shargabi

2021. December 6.

Content

Task description

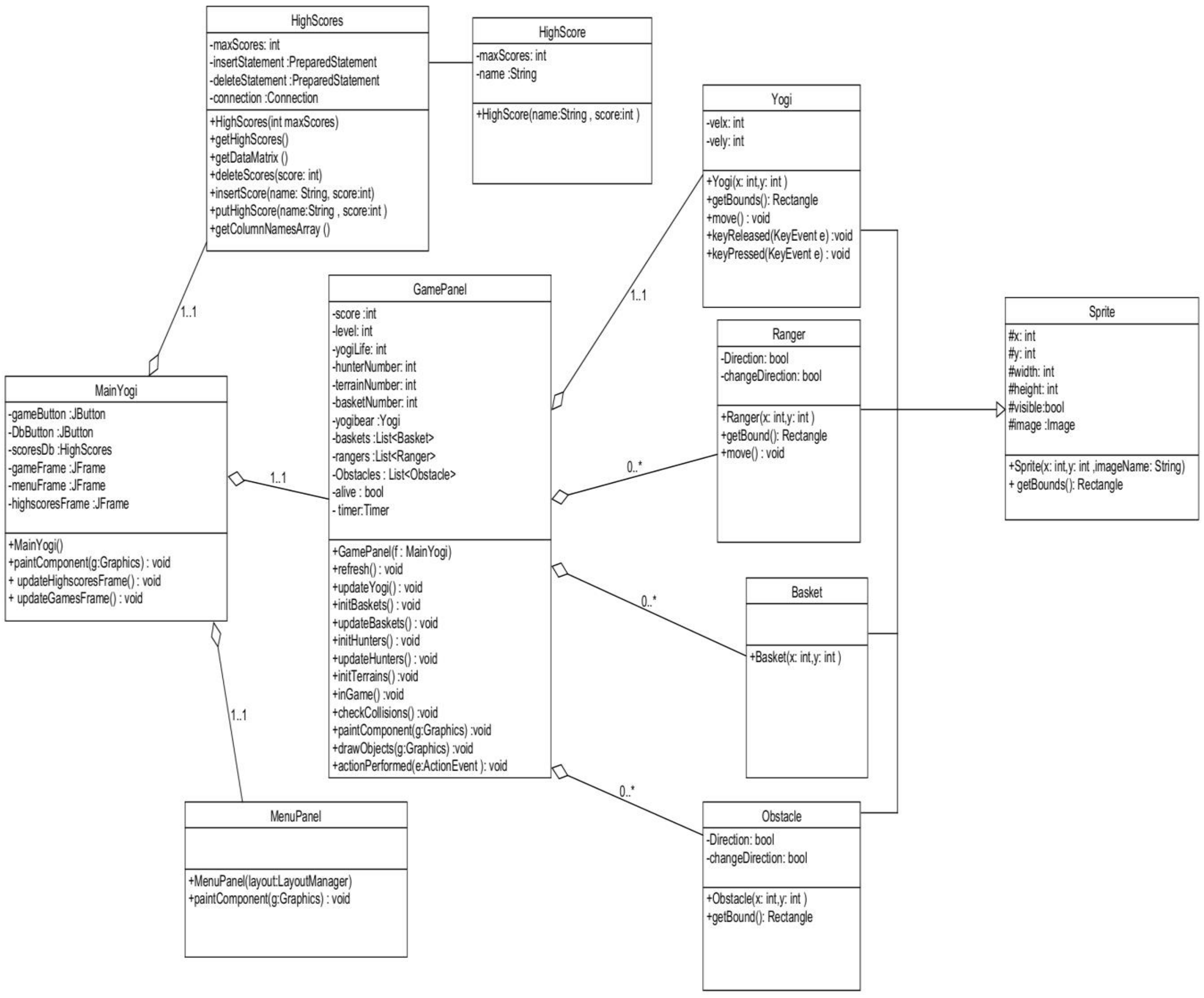
Yogi Bear wants to collect all the picnic baskets in the forest of the Yellowstone National Park. This park contains mountains and trees, that are obstacles for Yogi. Besided the obstacles, there are rangers, who make it harder for Yogi to collect the baskets. Rangers can move only horizontally or vertically 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.

Common requirements:

- Your program should be user friendly and easy to use. You have to make an object oriented implementation, but it is not necessary to use multilayer architectures (MV, MVC etc.).
- You have to use simple graphics for the game display. The "sprite" of the player’s character should be able to moved with the well known WASD keyboard buttons. You can also implement mouse event handlers to other functions of the game.
- You can generate the game levels with an algorithm, or simply load them from files. If you load the levels from file, then each one should be put into different files, and you have to create at least 10 predefined levels. If you generate the levels, then take care that the levels should be playable (player should be able to solve it).
- Each game needs to have a timer, which counts the elapsed time since the start of the game level.
- The task description should contain the minimal requirements. It is free to add new functionalities to the games.

UML Diagram


HighScores
-maxScores: int
-insertStatement :PreparedStatement
-deleteStatement :PreparedStatement
-connection :Connection
+HighScores(int maxScores)
+getHighScores()
+getDataMatrix ()
+deleteScores(score: int)
+insertScore(name: String, score:int)
+putHighScore(name:String , score:int)
+getColumnNamesArray ()
HighScore
-maxScores: int
-name :String
+HighScore(name:String , score:int)
Yogi
-velx: int
-vely: int
+Yogi(x: int,y: int)
+getBounds(): Rectangle
+move() : void
+keyReleased(KeyEvent e) :void
+keyPressed(KeyEvent e) : void
1..1
GamePanel
-score :int
-level: int
-yogiLife: int
-hunterNumber: int
-terrainNumber: int
-basketNumber: int
-yogibear :Yogi
-baskets :List<Basket>
-rangers :List<Ranger>
-Obstacles : List<Obstacle>
-alive : bool
- timer:Timer
+GamePanel(f : MainYogi)
+refresh() : void
+updateYogi() : void
+initBaskets() : void
+updateBaskets() : void
+initHunters() : void
+updateHunters() : void
+initTerrains() :void
+inGame() :void
+checkCollisions() :void
+paintComponent(g:Graphics) :void
+drawObjects(g:Graphics) :void
+actionPerformed(e:ActionEvent): void
1..1
MainYogi
-gameButton :JButton
-DbButton :JButton
-scoresDb :HighScores
-gameFrame :JFrame
-menuFrame :JFrame
-highscoresFrame :JFrame
+MainYogi()
+paintComponent(g:Graphics) : void
+ updateHighscoresFrame() : void
+ updateGamesFrame() : void
1..1
Ranger
-Direction: bool
-changeDirection: bool
+Ranger(x: int,y: int)
+getBound(): Rectangle
+move() : void
0..*
Sprite
#x: int
#y: int
#width: int
#height: int
#visible:bool
#image :Image
+Sprite(x: int,y: int ,imageName: String)
+ getBounds(): Rectangle
0..*
Basket
+Basket(x: int,y: int)
1..1
0..*
MenuPanel
+MenuPanel(layout:LayoutManager)
+paintComponent(g:Graphics) : void
Obstacle
-Direction: bool
-changeDirection: bool
+Obstacle(x: int,y: int)
+getBound(): Rectangle

Implementation

1) **Sprite:**
   a. Constructor ():
      a. Parameters: It takes x and y coordinates and the image.
      b. Inner work: creates the object according to the received x and y coordinates and the image.
   b. getBounds():
      a. Parameters: no parameters.
      b. Inner work: creates a rectangle with the size of the image.
2) **Yogi:**
   a. Constructor ():
      a. Parameters: It takes x and y coordinates.
      b. Inner work: call its super class constructor (Sprite) with the received x and y coordinates and the image.
   b. move ():
      a. Parameters: No parameters.
      b. Inner work: responsible for the movement of Yogi inside the park.
   c. KeyEvents:
      a. keyPressed (): Changes the velx and velx values according to the received KeyEvent..
      b. keyReleased (): Set velx and velx to 0 when the key is not pressed anymore.
      c.
3) **Ranger:**
   a) Constructor ():
      a. Parameters: It takes x and y coordinates.
      b. Inner work: call its super class constructor (Sprite) with the received x and y coordinates and the image.
   b) move ():
      a. Parameters: No parameters.
      b. Inner work: responsible for the movement of Ranger inside the park.
   c) getBound ():
      a. Parameters: No parameters.
      b. Inner work: creates a rectangle with the size of the Ranger.

4) Basket:

a) Constructor ():

a. Parameters: It takes x and y coordinates.

b.Inner work: call its super class constructor (Sprite) with the received x and y coordinates and the image.

5) Obstacle:

a) Constructor ():

a. Parameters:It takes x and y coordinates.

b. Inner work: call its super class constructor (Sprite) with the received x and y coordinates and the image (get one of the Obstacles images randomly).

b) getBound () :

a. Parameters: It takes int.

b. Inner work: creates a rectangle with the size of the Obstacle if the int equals 0.

6) HighScores:

a) Constructor ():

a. Parameters:It takes the max number of cloums in the table.

b. Inner work: contact with the database.

b) getHighScores () :

a. Parameters: No parameters.

b. Inner work: creates a sorted list of the highest scores in the database.

c. Exception: throws SQLException.

c) getDataMatrix():

a. Parameters: No parameters.

b. Inner work: creates a sorted matrix of the highest scores in the database.

c. Exception: throws SQLException.

7) GamePanel():

a) This class is a subclass of JPanel class and implements ActionListener, and it is responsible for the game logic, and its Constructor is responsible for adding keyListener for Yogi and initializing the rangers, baskets, obstacles, and Yogi , and starting the timer.

b) ActionListener:

a. actionPerformed(): responsible for checking if Yogi is still alive, updating the rangers, baskets, and Yogi, and checking if there are any collisions between objects and redrawing the objects after refreshing them.

c) refresh(): responsible for generating a new level Increases and the difficulty by generating more obstacles, rangers, and baskets to be Collected, and spawning Yogi to the entrance of the park.

d)paintComponent: responsible for drawing the background image for the game and drawing the rangers, baskets, and Yogi if Yogi is still alive, and drawing the Labels for the level, Lives, collected baskets, and time.

8) MenuPanel():

a) This class is a subclass of JPanel class and it is responsible for drawing the background image for the menu.

9) MainYogi():

a) This class is responsible for the Main Menu frame ,scores frame , and game frame.

b) ActionListener:

a. actionPerformed():This class has two actionPerformed methods one for the game button which is responsible for updating the game frame and setting the game frame to visible, and the other one is for the scores button which is responsible for updating the scores frame and setting the score frame to visible.

c) updateGamesFrame(): responsible for initializing the game frame and adding the GamePanel and the menu to it.

d) updateHighscoresFrame():responsible for initializing the scores frame and generating a table from the highest scores in the database.

Analysis

We perform the game into three main steps:

1. When the program starts, a menu window will appear where the player can select from the menu items which have two items Game and Scores.

2. If the player selects “New Game” from the menu items, the game will be generated, and a new window will be displayed for the game.

3. Once the player loses all his/her lives, a player name window will be generated and displayed to get the player’s name to add it the database of the game, and then a new window will be generated and displayed to ask the player if he/she wants to play again or no.

4. If the player selects “Scores” from the menu items, the highest 10 scores in the game database the Scores table will be generated and a new window will be displayed for the Scores table.

Testing

1. **Test level generation.**
2. **Test Yogi movement.**
3. **Test Rangers movement.**
4. **Test Obstacles randomly creation.**
5. **Test Scores table generation.**
6. **Test database getting the highest 10 scores.**
7. **Test the menu item restart.**
8. **Test the menu item exit.**
9. **Test the player’s name window generation.**