Comp165 Major Programming Assignment 3

Brick Breaker GUI

Introduction:

This assignment requires you to implement the GUI based classes in the Brick Breaker game using the JavaFX graphics library. You will implement the Player Profile GUI to create new players for tracking high scores and the game board that allows the user to play the game. Next, you will implement the game board and other classes to implement the actual Brick Breaker animated graphics.

ProfilePane Class:

ProfilePane	
-profiles:GameProfiles -profileFilename : String -configFilename : String -controls : Node	Profiles of all registered players File containing profiles data File containing Level data. Passed to GameBoard. //placeholder for the controls used in your GUI
+ProfilePane(profileFileName:String, configFileName:String)	

Level 1: **ProfilePane Design & Implementation (25pts)**

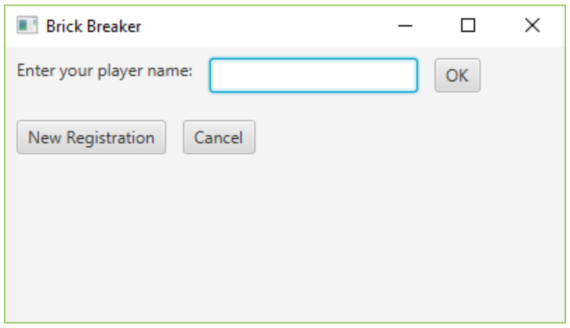


Figure 1- Example Profile GUI

a. Design a GUI for the player profile function of the Brick Breaker game. Your design should include the following:
   - A control to select an existing player profile. The GUI you used for MP1 allowed the user to enter the player name in a textbox. Better solutions would use a ComboBox or a ListBox.
   - A control to enter a new player's name.
   - A control (e.g. Button) to initiate a search for an existing player (or to indicate that the desired player has been selected in the ComboBox or ListBox).
   - A control (e.g. Button) to initiate the creation of a new player profile.
   - A control to display status messages to the user.

   Note: You do not have to submit your design.

b. Implement your GUI using JavaFX. Start by creating a JavaFX application in Netbeans. Name the project StudentBrickBreaker and copy code contained in the StudentBrickBreaker.java file from MP1 into the newly created StudentBrickBreaker file. Add a new class to your project and name it ProfilePane. ProfilePane should extend one of the subclasses of Pane (StackPane, HBox, VBox, GridPane or BorderPane). The particular Pane you use depends on your GUI design. My design used a HBox with a VBox on the inside for the TextBox and Button at the top of the GUI.

Create properties in ProfilePane for each needed control and add them to Pane or other inner Panes that you create. The constructor for ProfilePane should read in the profile data from the profile text file and save it in the GamesProfiles object (profiles). Reading the player profile text file will require you to link in the jar file developed from MP1. My version of these classes will be available in a jar download.

Instantiate an instance of ProfilePane in your start method and add it to a Scene so that it can be displayed in the Stage. **Your GUI should be displayed but the controls do not have to have event handling implemented at this point.** Grading will be based off of whether your GUI meets all the given requirements.

Level 2: (35pts) Add event handling for your ProfilePane GUI. The particular events you have to handle will depend on your GUI design. Use anonymous listeners for your controls for this GUI. In general, you will have to respond to controls that initiate:

- Searching to see if a new player name is already used for another player profile.
- Adding a new player profile to the player profile text file.
- Launching a new Stage. For this version, you should display a new Stage containing the selected gamer profile followed by a complete list of the other gamer profiles in a TextArea (or some other control).

Level 3 (60pts): Read configuration file, display the bricks for the first level and display the Paddle. You should also be able to move the paddle using the mouse. Your Level 2 project should be altered so that the ProfilePane will instantiate a GameBoard object, place it in a Scene and then place that Scene in the new Stage from Level 2. The new Stage will display the GameBoard that will contain the PlayArea with the bricks and movable paddle. Here is the method I called in one of my ProfilePane event handlers to display the Stage containing my GameBoard:

```
public void startGameBoard() {
    this.setVisible(false);  //do not display the ProfilePane any longer.
    GameBoard gameBoard = new GameBoard(levels, profiles, this.profileFilename);
    Scene gameScene = new Scene(gameBoard);
    Stage gameStage = new Stage();
    gameStage.setScene(gameScene);
    gameStage.setTitle("Brick Breaker");
    gameStage.show();
}
```

Brick Class - Extends the javafx.scene.shape.Rectangle class.

Brick	
-pointValue : int +BRICK_WIDTH : int +BRICK_HEIGHT : int	Brick point value when hit by the ball. Width of the brick (public constant) Height of the brick (public constant)
+Brick(xLoc : int, yLoc : int) //getter and setter for pointValue	Constructor that creates the Brick at the given loc.

Note: The Brick has other basic properties that are inherited from the Rectangle class (e.g. Color). I made my bricks 35x20.

Paddle Class – extends Rectangle. The xLoc will be controlled using the mouse. The yLoc will be fixed.

Paddle	
-BASE_Y : int +PADDLE_WIDTH : int +PADDLE_HEIGHT : int -paHeight : int -paWidth : int	The y location in the playArea where the Paddle will move. Width of the Paddle (public constant) Height of the Paddle (public constant) The height of the play area in pixels. The width of the play area in pixels.
+Paddle(paWidth : int, paHeight : int) +move(xLoc : double) : void	Instantiate the Paddle in the middle of the playArea. Move the paddle so that the top edge is centered at location xLoc, BASE_Y.

Note: I made my paddle 70x10.

PlayArea Class – extends the Pane class. This class will eventually display your bricks, paddle and ball. It is where the game action occurs. Other properties will be added in subsequent levels as needed.

PlayArea	
- bricks : Brick[][] -BASE_Y : int -paHeight : int -paWidth : int	2D array of Brick objects. The yloc in the playArea where the first row of bricks will start. The height of the play area in pixels. The width of the play area in pixels.
+PlayArea(paHeight :int, paWidth:int, Level level) -createBricks(level : Level) : void +movePaddle(xLoc : double) : void	 Create the bricks and populate the 2D array. Move the Paddle to the givein xLoc

The constructor for the PlayArea should create the bricks and the paddle and add them to the PlayArea. I made my PlayArea 700x600.

GameBoard Class – This is the outermost Pane that should **extend the BorderPane class**. It will have a zone for the PlayArea (center zone), the ScorePane (added in later levels) and the StatusPane (added later).

GameBoard	
-playArea : PlayArea -profiles : GameProfiles -currentLevel : int -levels : Level[] -profilesFilename : String -paddleHandler : PaddleHandler	The play area. All gamer profiles The current game level – starts at 0 All game level information from config file. The profiles file name. Object of the inner class PaddleHandler that implements EventHandler<MouseEvent>
+GameBoard(levels:Level[], profiles : GamerProfiles, profilesFilename : String)	

MouseHandler hint: You are going to register the MouseHandler object with the GameBoard object in the GameBoard object constructor:

```
this.setOnMouseMoved( <your MouseHandler object name> );
```

Level 4 (75pts): Create an animated ball that will collide with the paddle and walls but not the bricks. When the player clicks the left mouse button, the ball should appear and launch from the top middle position of the current paddle position at a random angle between 40 and 140 degrees (on a standard unit circle where 90 degrees is north). Register an anonymous mouse listener with the GameBoard class that will launch a new ball when the left mouse is clicked. Of course, launching a new ball should only occur if the ball is currently inactive (i.e. a new game or just after a dead ball).

Ball Class: extends the Circle class

Ball	
-speed : int -direction : double	Pixels moved in current direction at each move. Current direction of ball (0 to 360 degrees).
+Ball(paWidth : int, paHeight : int, dir : double) +move() : void +getTopEdge():int +getBottomEdge():int +getLeftEdge():int +getRightEdge():int	Instantiate the Paddle in the middle of the playArea. Move the ball at the current speed and direction. Returns centerY() - radius Returns center() + radius Returns centerX() – radius Returns centerX() + radius

Note: My version of the game used a ball with a radius of 7 pixels.

Additions to PlayArea Class: Most of these are needed to allow the event handlers in the GameBoard to access needed objects in the PlayArea. You are free to add other methods if needed.

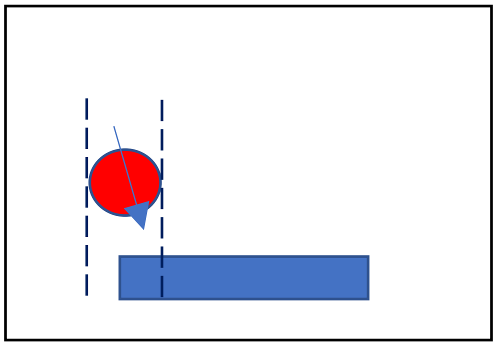
+handleCollisions():void – used to determine if the ball has collided with a wall, the paddle or a brick (implemented in Level 5). If so, take appropriate action.

+moveBall():void – move the ball at the current speed and direction (i.e. call ball.move()).

+newBall():void – Reposition the ball to launch position, generate a random direction and set the visibility to true.

+setBallVisibility(visibility : Boolean) : boid

Ball Animation: Create an extended AnimationTimer class in the GameBoard class. The contents of the handle() method will be similar to what we did in the BouncingBall application. After every move of the ball, you will call the *handleCollision()* method (See next section) in the PlayArea class to determine if the ball has collided with a wall, the paddle or a brick (implemented in Level 5).



Paddle Collision: Add a new method to the PlayArea class named *checkCollisions():void*. This method will ultimately check for all game collisions. For this level, if the ball is headed down toward the paddle and has a yLoc low enough to collide with the paddle, you need to check to see if there is a collision with the paddle. Determine whether the ball's lateral position overlaps with the paddle boundaries. If so, the ball should bounce using the same principles as a collision with the bottom wall. Here is some pseudo-code:

```
if (bottom of the ball is equal to or lower than the top edge of the paddle and not past the paddle) &&
   (right edge of ball is  > left edge of paddle && left edge of ball is < right edge of paddle)   then
        handle the collision
else if (top of the ball < bottom of paddle) then
        ball is dead
```

When the ball becomes dead, set the visibility property to false and stop the AnimationTimer. Clicking the left mouse button should move the ball to the center of the paddle, set the ball visibility back to true and start the AnimationTimer again. To add some variety, try changing the direction of the ball after it hits the paddle 5 consecutive times.

Level 5 (90pts): Implement brick collision detection with the ball and scoring.

ScorePane Class: extends the HBox class. Displays the current score in a large font.

ScorePane	
-score : int -lblScore : Label	Current score Label that displays the score in the ScorePane.

+ScorePane()
+setScore(score : int) : void
+getScore() : int
+incrementScore(pts : int)
	
	Set the score to the given value.
	Return the current score.
	Increment the current score by the given amount.

The ScorePane will be instantiated in the GameBoard constructor. Add a parameter to the PlayArea constructor to allow the passing of a reference to the ScorePane as an argument. Also add a new ScorePane property in the PlayArea class and assign the new parameter value to the property. You will need the reference to the ScorePane so that you can update the score when there is a collision of the ball with a brick in the PlayArea.

Brick Collision: Add a method in the PlayArea class that determines after every ball move if the ball has collided with a visible brick. This new method is called inside of the handleCollisions() method you created in Level 4. If there is a collision, the ball is reflected, the brick visibility property is changed to false and the current score is updated by the point value of the brick. Note: The ball can collide with bricks moving in an upward or downward direction.

Level 6 (100pts): Implement the remainder of the game features.

- Determine if all the bricks have been hit. If so, start the next level.
- Determine of all the balls have been lost. If so, end the game.
- Add the player profile information on the GameBoard: including name and current high score.
- Update the PlayerProfile if a new high score has been achieved.
- You are free to add any game frills such as sound, shrinking/expanding paddle, multiple balls, etc.