Final Project
-Object-Oriented Programming-

Project Name: BrickBreaker

Name: Justin Theofilus Yonathan

ID: 2502036382

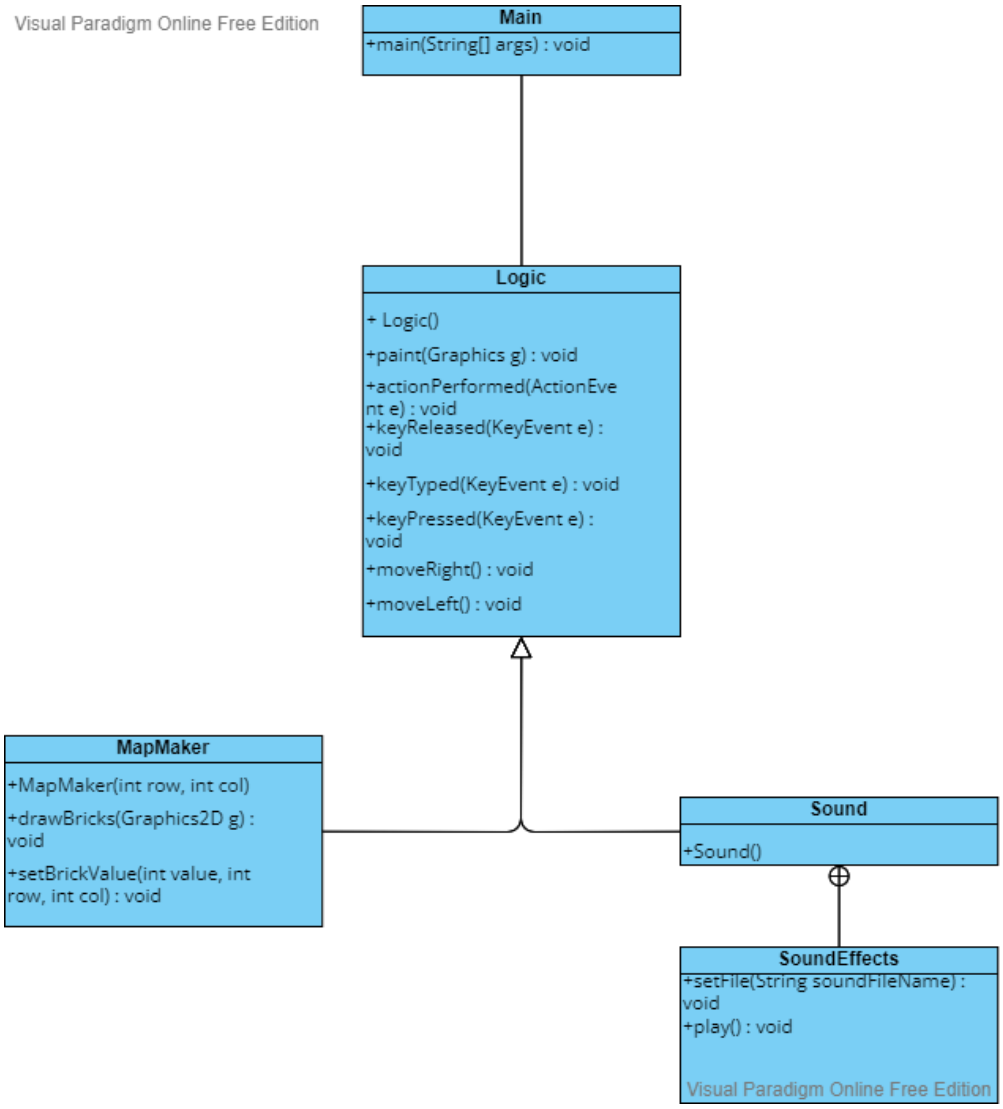
Class: L2BC

Project Specification

For my Object-Oriented Programming final project, I decided to make a classic 8-bit style BrickBreaker game. Before settling on this idea I had other ideas on what to make for the OOP final project such as Pacman and a top-down view shooter game, but after seeing the syntax required to create game windows and other instances I decided to take on something much simpler, which is why I made a BrickBreaker game instead. Moreover, I have an interest in 8-bit games so that adds to the reason why I decided to make an 8-bit style game. In this project, I used a lot of imports to construct the game but the main ones are Javax.swing.JFrame, Javax.swing.JPanel, Java.awt.graphics & graphics2D, Java.awt.event (ActionListener/Event & KeyListener/Event). To explain briefly what these imports are used for, JPanel and JFrame are used to draw the game window, whereas graphics and graphics2D are used to draw the sprites of the game, and lastly ActionListener, ActionEvent, KeyListener, and KeyEvent are used to read key presses from the keyboard. As a whole, these imports give the game its structure and mechanics. In this game, I also added sound effects to the game by importing javax.sound, that way I was able to add sound effects for collisions and key presses which makes the game a bit brighter in my opinion. In this game, there are several states that players can enter. First, there is the initial state where everything is just at a standstill. Secondly, there is the playing state which is when the brick breaker game plays normally. Thirdly, we have the pause, win, and lose states which basically freezes the ball in place and displays a text corresponding to each condition. As for the mechanics of the game, since it's just an 8-bit brick breaker game it is really simple, players just have to break every single brick in order to win by bouncing the ball off of the slider using the left and right arrow keys as controls.

Solution Design

Class Diagram:



Code Explanation:

For the code explanation, I will be explaining what each java file does and how each of them contributes to running the game. Firstly, we have the Main class. Within the Main class, we mainly use it as the driver class that actually allows the code to be executed, and inside the class and the main method, we have multiple method calls in order to set up the game window properties using methods such as setBounds, setTitle, setVisible, etc. Also within the Main class, we created an object for the Logic class and JFrame import under the names game and obj respectively.

```
package newfp;
//Imported Class
import javax.swing.JFrame;

public class Main {

    public static void main(String[] args) {
        //Creating object instance
        JFrame obj = new JFrame();
        Logic game = new Logic();
        //Setting the window size
        obj.setBounds(10,10,700,600);
        //Setting title
        obj.setTitle("BrickBreaker");
        //Others
        obj.setResizable(false);
        obj.setVisible(true);
        obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        obj.add(game);
```

Secondly, we have the Logic class which can basically be called the brain of the program, and as the name of the class suggests it contains the logic on how pretty much everything works. When the Main class accesses the Logic class the first thing that the Logic class does is import the modules, after that the variables are initialized, and we reach the constructor method. Within the constructor, there are a couple of methods being called but the most important ones would probably be creating an object instance of the MapMaker class, which gives the data for the bricks, and calling the addKeyListener to allow key presses to be read.

```
//Class constructor
public Logic(){
    map = new MapMaker(3,7);
    //map2 = new MapMaker2(2,6);
    addKeyListener(this);
    setFocusable(true);
    setFocusTraversalKeysEnabled(false);
    timer = new Timer(delay,this);
    timer.start();
}
```

After going through the constructor, we have the method paint() which accepts a Graphics parameter from the java.awt import. Inside the paint method, we are drawing the sprites of the game such as the slider, the background, borders, and the ball.

Moreover, by combining it with conditionals we are also able to draw the texts for the pause screen, winning screen, and lose screen.

```
//Drawing borders
g.setColor(Color.white);
g.fillRect(0,0,3,592);
g.fillRect(0,0,692,3);
g.fillRect(682,0,3,592);

//Drawing scores
g.setColor(Color.white);
g.setFont(new Font("serif",Font.BOLD, 25));
g.drawString("score: "+ score, 550, 30);

//Drawing slider
g.setColor(Color.white);
g.fillRect(slider_x,550,100,8);
```

```
//Drawing text for the win screen
if(numBricks <= 0){
    Run = false;
    Pause = false;
    ballx_dir = 0;
    bally_dir = 0;
    g.setColor(Color.green);
    g.setFont(new Font("serif", Font.BOLD, 30));
    g.drawString("You Won!! Final Score: "+score, 150, 300);

    g.setFont(new Font("serif", Font.BOLD, 20));
    g.drawString("Press Enter to Restart!",230,350);
}
```

Next, we have the actionPerformed method which is a required method to be written because the Logic class implements ActionListener and KeyListener along with keyPressed, keyReleased, and keyTyped in which the last two remain unused in this game. Within the actionPerformed method, we are able to write the code for the physics of the game such as ball movement, collision, direction change, etc. To explain briefly. First, we have the collision between the ball and slider. This is done by changing the direction of the ball every time the ball collides with the slider and the way the program could detect that is by creating rectangles that serve as the sensor between the ball and the slider, so when the rectangles intersect, the direction of the ball changes.

```
//Required Methods by ActionListener and KeyListener
@Override
public void actionPerformed(ActionEvent e) {
    timer.start();
    if(Run){
        //Implementing collision between ball and slider
        if(new Rectangle(ball_x, ball_y, 20, 20).intersects(new Rectangle(slider_x,550,100,8))){
            bally_dir = - bally_dir;
            sfx.se.setFile(bliplSFX);
            sfx.se.play();
```

Second, we have the collision between the ball and bricks, so how this works is first the code takes the data of the brick layout which is defined in the MapMaker class in the form of a two-dimensional array, next the code iterates through each index of the 2d array and checks for collisions using the same method as the slider collision. When a collision between the ball and brick is detected it changes the specific brick with a corresponding index's value from 1 to 0, making it disappear on collision.

```
A: for(int i = 0; i < map.map.length; i++){
                    //map2.map[0].length
    for(int j = 0; j < map.map[0].length; j++){
        if(map.map[i][j] > 0){
                        //map2
            int brick_x = j*map.brickWidth + 80;
            int brick_y = i*map.brickHeight + 50;
            int brickWidth = map.brickWidth;
            int brickHeight = map.brickHeight;
            //Initializing rectangle which acts as the sensor for the collision
            Rectangle rect = new Rectangle(brick_x,brick_y,brickWidth,brickHeight);
            Rectangle ball_rect = new Rectangle(ball_x,ball_y,20,20);
            Rectangle brick_rect = rect;

            //Implementing collision between bricks and ball
            if(ball_rect.intersects(brick_rect)){
                //Making broken bricks disapear on collision
                //map2
                map.setBrickValue(0, i, j);
                numBricks --;
                score += 5;
                //Implementing direction change of ball
                if(ball_x + 19 <= brick_rect.x || ball_x + 1 >= brick_rect.x + brick_rect.width){
                    ballx_dir = -ballx_dir;
                }else{
                    bally_dir = -bally_dir;
                }
                sfx.se.setFile(breakSFX);
                sfx.se.play();
                break A;
```

Thirdly, we have the collision between the ball and borders which is set up using conditionals, so when the ball passes a certain x or y value it automatically changes directions, creating the illusion of bouncing on collision with the borders. Last but not least, we have the ball movement which is basically making the position of the ball change by adding the directional value continuously.

```
        //Moving the ball
        ball_x += ballx_dir;
        ball_y += bally_dir;


        //Setting collision points and changing ball direction
        //Collision for left border
        if(ball_x < 0){
            ballx_dir = -ballx_dir;
            sfx.se.setFile(blip2SFX);
            sfx.se.play();
        //Collition for top border
        }if(ball_y < 0){
            bally_dir = -bally_dir;
            sfx.se.setFile(blip2SFX);
            sfx.se.play();
        //Collision for right border
        }if(ball_x > 665){
            ballx_dir = -ballx_dir;
            sfx.se.setFile(blip2SFX);
            sfx.se.play();
        }
    }
```

The next and last few methods within the Logic class are the keyPressed, moveRight, and moveLeft methods. The keyPressed method allows us to read key press events and determine what to do should that event be triggered and the moveLeft/Right methods are just methods that define how much the slider should move on each press of the left and right arrow keys.

```
//Method to move sliders
public void moveRight(){
    Run = true;
    slider_x += 20;
}

public void moveLeft(){
    Run = true;
    slider_x -= 20;
}
```

```
@Override
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_RIGHT){
        //Setting limits of slider movement
        if(slider_x >+ 580){
            slider_x = 582;
        }else{
            moveRight();
        }
    }if(e.getKeyCode() == KeyEvent.VK_LEFT){
        //Setting limits of slider movement
        if(slider_x < 5){
            slider_x = 4;
        }else{
            moveLeft();
        }
    //Restarting the game when user wins or loses
    }if(e.getKeyCode() == KeyEvent.VK_ENTER){
        if(!Run){
            Run = true;
            ball_x = 350;
            ball_y = 350;
            ballx_dir = n;
            bally_dir = -5;
            slider_x = 310;
            score = 0;
            numBricks = 21;
            map = new MapMaker(3,7);
            sfx.se.setFile(Continue);
            sfx.se.play();
        }
```

The third class that this game has is the MapMaker class which is a straightforward class that has the role of defining how the brick layout is constructed and then drawing it. It is done by first creating a two-dimensional array to store the data which is then split according to how many rows and columns we defined while creating the MapMaker object instance back in the Logic class, the way it is split is by using the setStroke method from the graphics2D import. Within the constructor of the class, it is defined that each brick has a value of 1, and if the value becomes 0 that brick will disappear. At the bottom of the MapMaker class, we also have a method that allows the value of the bricks to be changed just by calling this method which simplifies the process.

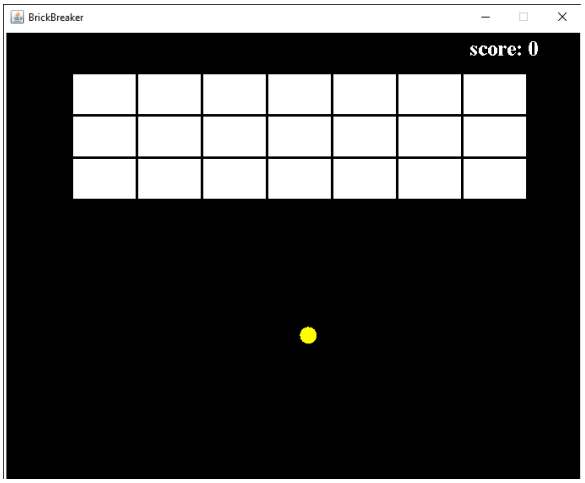
```
//Method to allow bricks to disappear
public void setBrickValue(int value, int row, int col){
    map[row][col] = value;
}
```

```
//Class constructor
public MapMaker(int row, int col){
    map = new int[row][col];
    //Returning whether each box in table exists or not
    //1 = Exists
    //0 = Doesn't Exist
    for(int i = 0; i < map.length; i++){
        for(int j = 0; j < map[0].length; j++){
            map[i][j] = 1;
        }
    }


    brickWidth = 540/col;
    brickHeight = 150/row;
}
//Method to draw the bricks
public void drawBricks(Graphics2D g){
    for(int i = 0; i < map.length; i++){
        for(int j = 0; j < map[0].length; j++){
            if(map[i][j] > 0){
                //Drawing big rectangle
                g.setColor(Color.white);
                g.fillRect(j*brickWidth +80, i*brickHeight + 50, brickWidth, brickHeight);

                //Separating big rectangle into smaller bricks
                g.setStroke(new BasicStroke(3));
                g.setColor(Color.black);
                g.drawRect(j*brickWidth +80, i*brickHeight + 50, brickWidth, brickHeight);
            }
```

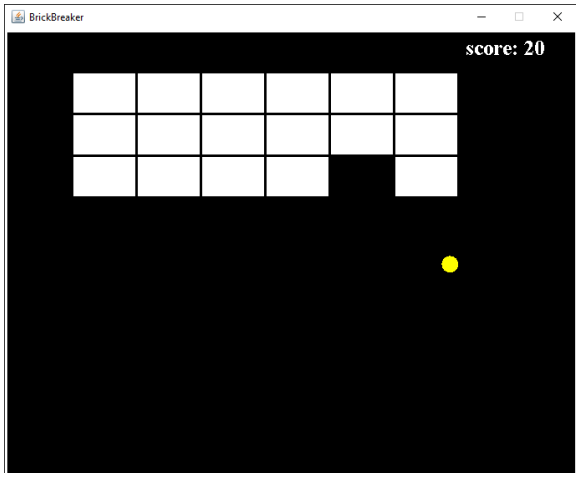
The fourth and last class that my project has is the Sound class which is for adding sound effects under certain conditions. Inside the Sound class, we have the setFile method which is used to assign an audio file to be played, and also the method play which as the name suggests is used to play the sound.

```
//Imported Classes
import java.io.File;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;

public class Sound {
    //Class Constructor
    public Sound(){
    }
    //initializing clip variable
    Clip clip;
    //Method to set a certain audio file to play
    public void setFile(String soundFileName){
        try{
                File file = new File(soundFileName);
                AudioInputStream sound = AudioSystem.getAudioInputStream(file);
                clip = AudioSystem.getClip();
                clip.open(sound);
        }
        catch(Exception e){

        }
    }
    //Method to play the audio file
    public void play(){
        clip.setFramePosition(0);
        clip.start();
    }
```

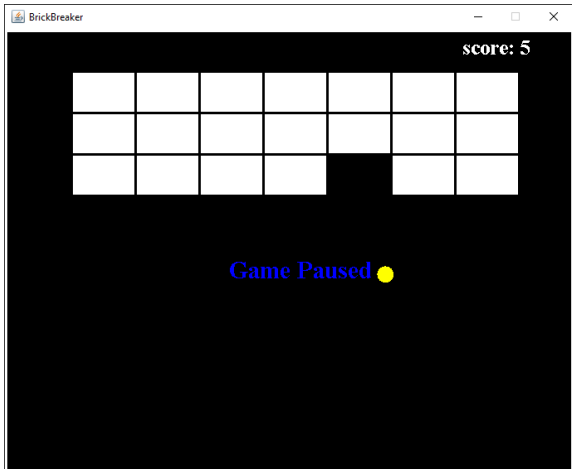
Working Program Screen Shots



Initial state

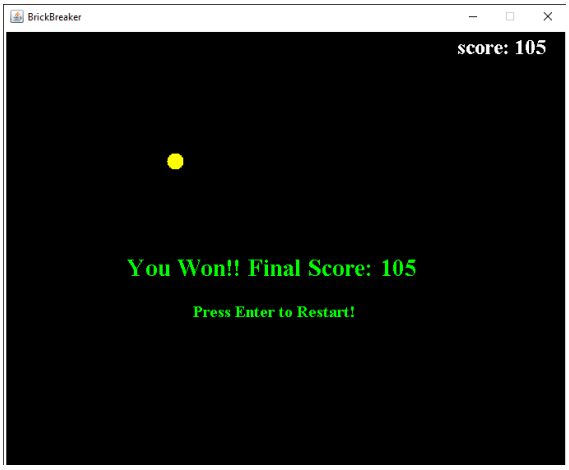


Playing state



Pause state

Lose state



Win state

Reflection

To me, getting to create a game in java provides a whole new challenge when compared to python last semester. I feel like things have gotten slightly more difficult as there are so many imports that I have to implement in java in order to run this game. On the other hand, while creating my project in the previous semester I only used one major import which was pygame. Hence the reason I chose a very simple game to create. Other than that, this experience has certainly sparked a new interest in me for developing java based games because despite the issues I stated before, I have found that java has its own advantages as well, especially in regards to functionality. I believe that by getting this valuable experience, I have grown to understand if not a little bit about java and the potential ways it could be used. To sum up, I am glad to have taken part in this experience as I believe it has given me some knowledge and experience which will help me in realizing my dream.

References:

- https://mixkit.co/free-sound-effects/win/
- https://freesound.org/people/jalastram/sounds/317754/
- https://freesound.org/people/NoiseCollector/sounds/4379/
- https://freesound.org/people/NoiseCollector/sounds/4378/
- https://freesound.org/people/cabled_mess/sounds/350876/
- https://www.youtube.com/c/RyiSnow
- https://www.youtube.com/c/AwaisMirza1/videos
- https://online.visual-paradigm.com/