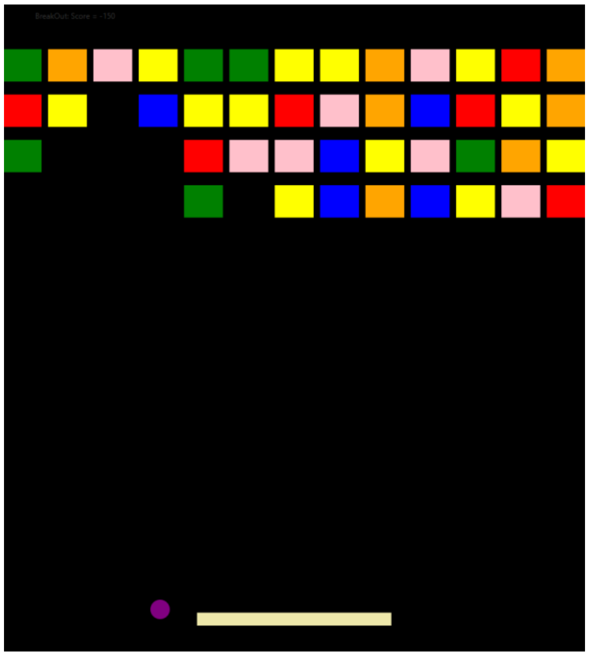
# BREAK-OUT GAME

A Karan Vani Production



MAY 7, 2021

DOCUMENTATION

CI401 – Introduction to Programming

## Table of Contents

## Introduction

Breakout or more commonly known as 'Brick Breaker' is a game a game consisting of three more main objects and one main goal. The objects are, the bricks, a ball, and a bat. The goal of the game is to break all the bricks on the screen using the bat. I will be using a skeleton code given to me by the University of Brighton.  This game is fully coded in Java in the Eclipse IDE.

## Running the Code

The submission includes a zip file that has the game's source code and if needed the assets for the game itself.

### In Eclipse:

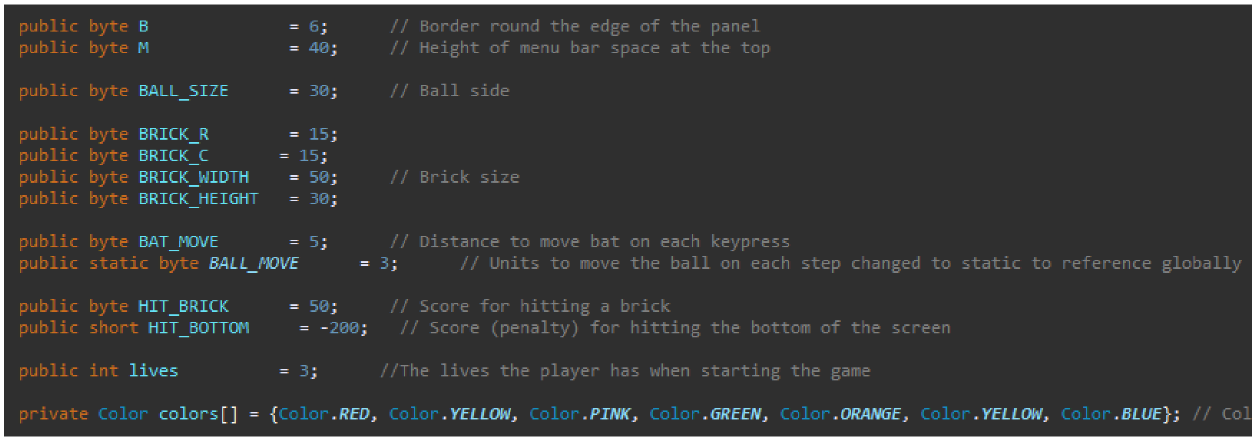
To run the game, extract the zip file into a desired folder and import it into Eclipse. Once imported, you will need to select your own JavaFX library installed on the computer to run the game without any problems. Then click run on the top which is a green circle with a white play button.

### In BlueJ:

To run the game, there is a file called package and is a BlueJ project file. Run this file, it will open BlueJ. Head to tools click on compile then right click on the main class and execute the code. This will run the game and show a console in the background.

## The Code

### Data Types



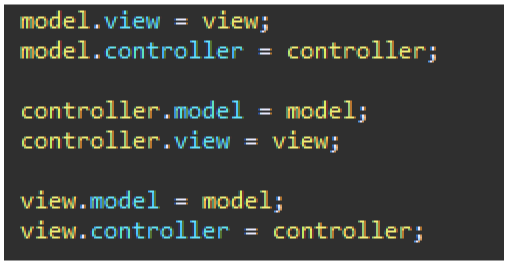
```
public byte B               = 6;     // Border round the edge of the panel
public byte M               = 40;    // Height of menu bar space at the top

public byte BALL_SIZE       = 30;    // Ball side

public byte BRICK_R         = 15;
public byte BRICK_C        = 15;
public byte BRICK_WIDTH     = 50;    // Brick size
public byte BRICK_HEIGHT    = 30;

public byte BAT_MOVE        = 5;     // Distance to move bat on each keypress
public static byte BALL_MOVE      = 3;      // Units to move the ball on each step changed to static to reference globally

public byte HIT_BRICK       = 50;    // Score for hitting a brick
public short HIT_BOTTOM      = -200;   // Score (penalty) for hitting the bottom of the screen

public int lives           = 3;     //The lives the player has when starting the game

private Color colors[] = {Color.RED, Color.YELLOW, Color.PINK, Color.GREEN, Color.ORANGE, Color.YELLOW, Color.BLUE}; // Col
```

Developing games and programs has a certain impact on the memory/ram of the machine it is running. A game like this would not take as much memory but for good practice I used the best possible data types and access modifiers. Due to the classes sharing a majority of the information, a lot of the access modifiers were kept as public.

## The MVC

The game and the code are based on the MVC (Model View Controller) design. Where all these classes talk to each other and make the game work. The Model is responsible for the game initiation, the calculation and is basically the brains of the game. The Controller class takes care of the user inputs and sends the input to the Model which then sends it to the View class, which outputs the current position of the objects on to the screen and is what the player can see.

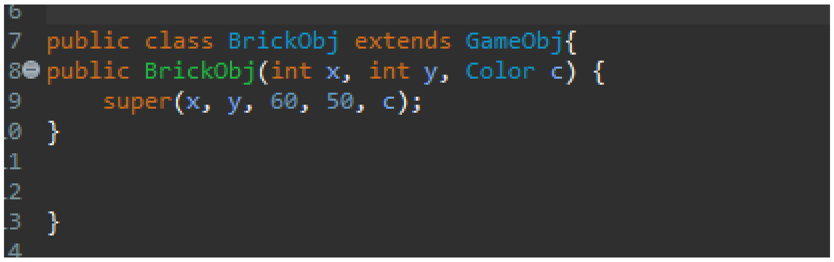
The skeleton code the university provided already connected the model, view, and controller classes together.



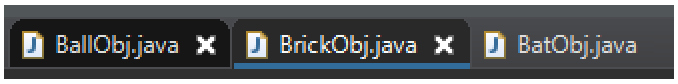
```
model.view = view;
model.controller = controller;

controller.model = model;
controller.view = view;

view.model = model;
view.controller = controller;
```

## Game Objects

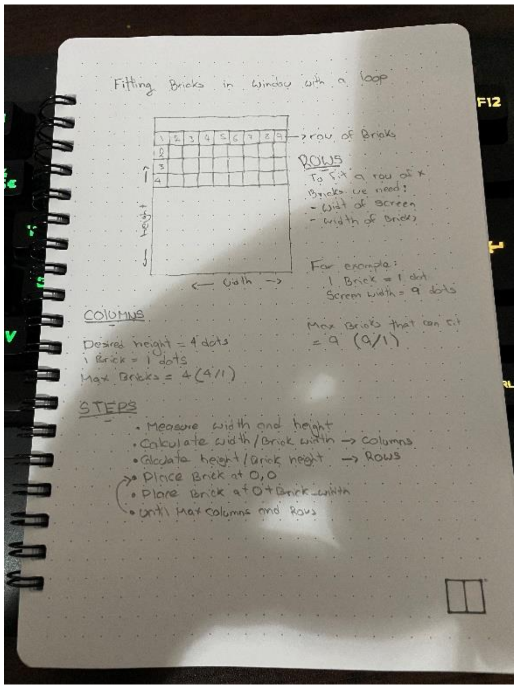
The game itself has the core objects, such as the bricks, the bat, and the ball. Which instead of forming into one class I made three different classes, one for each of the objects. This enabled me to take control over an individual object easily.



```

public class BrickObj extends GameObj{
public BrickObj(int x, int y, Color c) {
    super(x, y, 60, 50, c);
}


}
```



## Adding Bricks



Fitting Bricks in Window with a loop

ROWS
To fit a row of x Bricks we need:
- Widt of screen
- width of Bricks

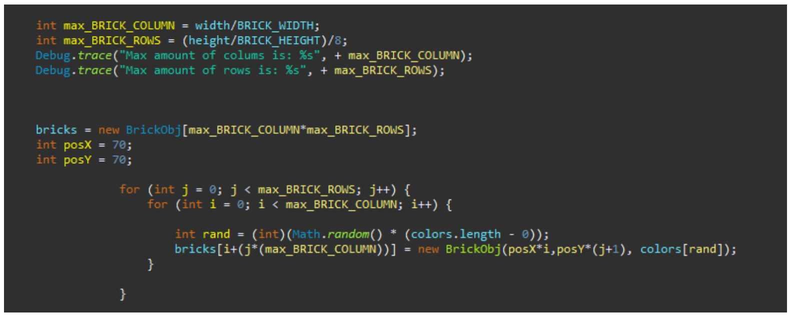
For example:
1 Brick = 1 dot
Screen width = 9 dots

Max Bricks that can fit = 9 (9/1)

COLUMNS
Desired height = 4 dots
1 Brick = 1 dots
Max Bricks = 4 (4/1)

STEPS
- Measure width and height
- Calculate width / Brick width → columns
- Calculate height / Brick height → Rows
- Place Brick at 0,0
- Place Brick at 0 + Brick_width
- until Max columns and Rows

The game was missing a crucial aspect, the bricks. To form the bricks, I decided to design an algorithm that uses arrays and loops to place the bricks. The figure on the right is how I approached the problem. I had taken account of the width and the height of the game screen along with the height and width of the bricks them selves and checked the max

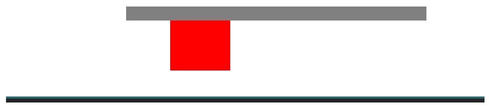


```
int max_BRICK_COLUMN = width/BRICK_WIDTH;
int max_BRICK_ROWS = (height/BRICK_HEIGHT)/8;
Debug.trace("Max amount of colums is: %s", + max_BRICK_COLUMN);
Debug.trace("Max amount of rows is: %s", + max_BRICK_ROWS);


bricks = new BrickObj[max_BRICK_COLUMN*max_BRICK_ROWS];
int posX = 70;
int posY = 70;

            for (int j = 0; j < max_BRICK_ROWS; j++) {
                for (int i = 0; i < max_BRICK_COLUMN; i++) {

                    int rand = (int)(Math.random() * (colors.length - 0));
                    bricks[i+(j*(max_BRICK_COLUMN))] = new BrickObj(posX*i,posY*(j+1), colors[rand]);
                }

            }
```

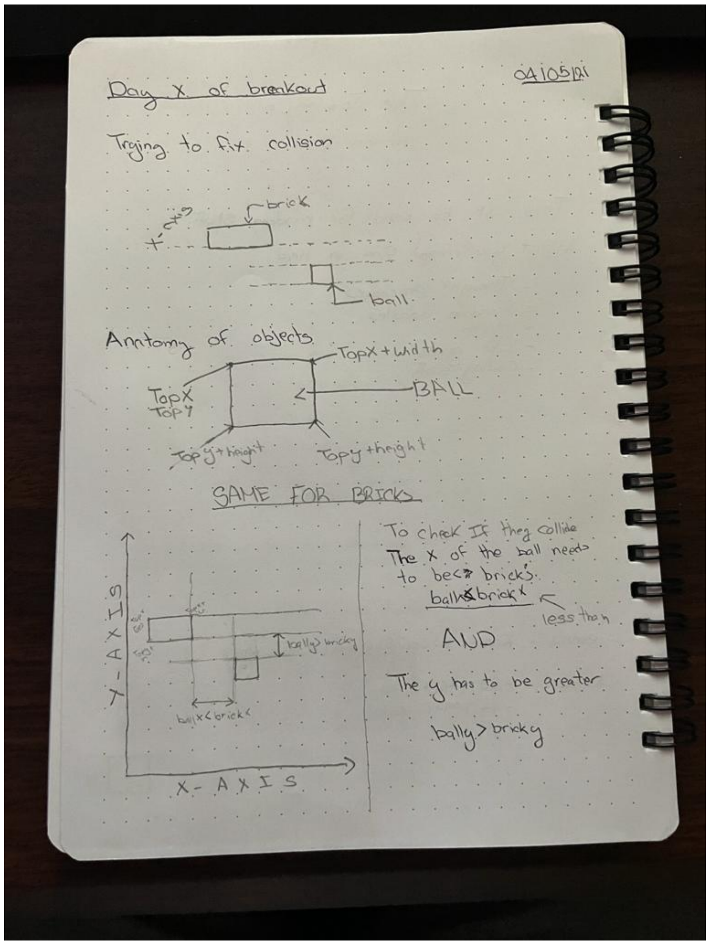
that could fit. I set a double loop, one for the rows and one for the columns and formed the randomly colored bricks.

## The Collision

Once the core game objects and game was set in place, it was time for the collision. The code provided already had collision detection but had some flaws, so I decided to completely reword the collision check. Towards the start of the semester, our lecturer had pointed out a bug during the

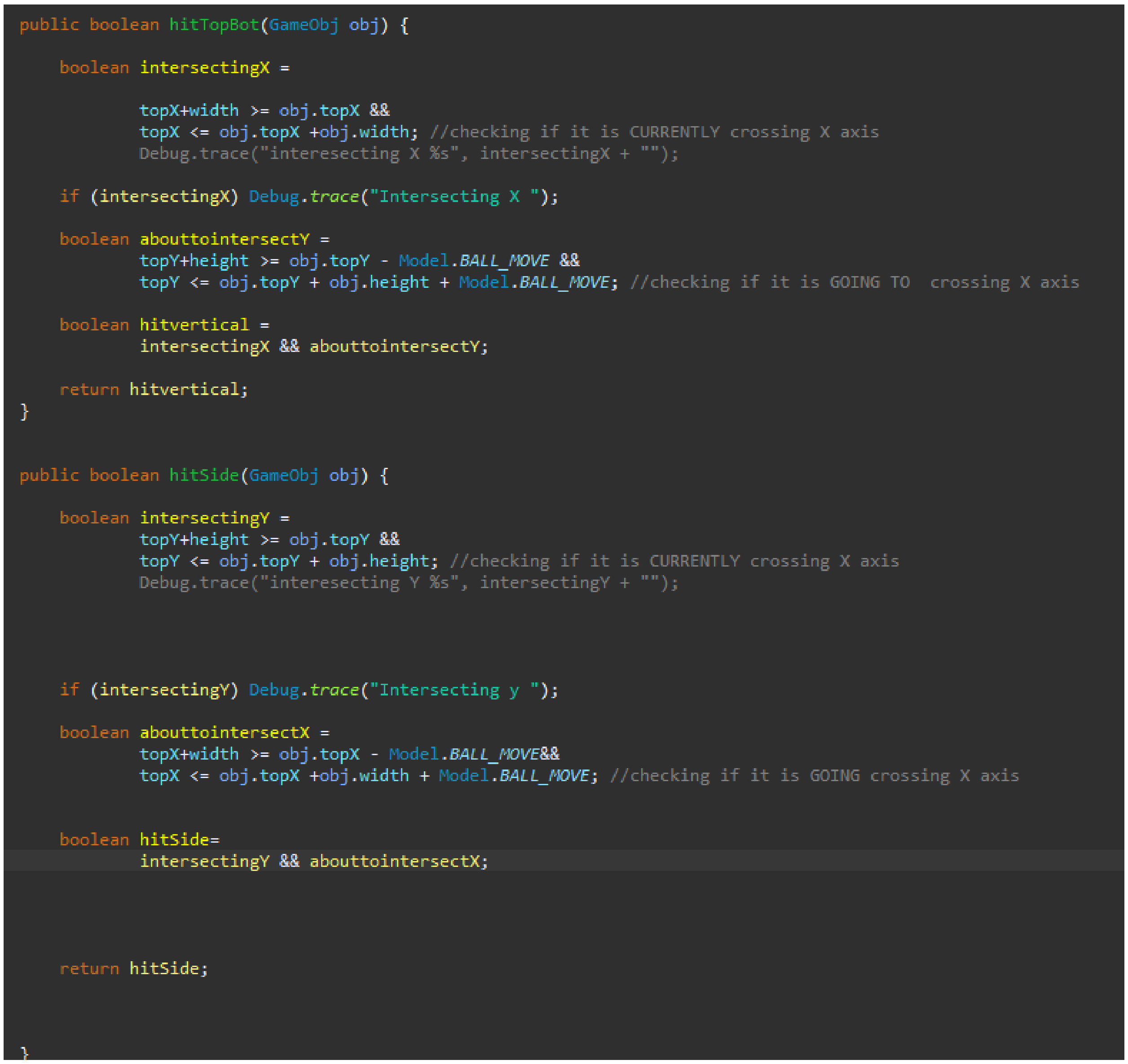


demonstration. This bug was causing the ball to get stuck inside of the bat if it was hit from the corner or side of the bat, as shown in the figure below.



The picture on the right is showing the thought process for the collision.

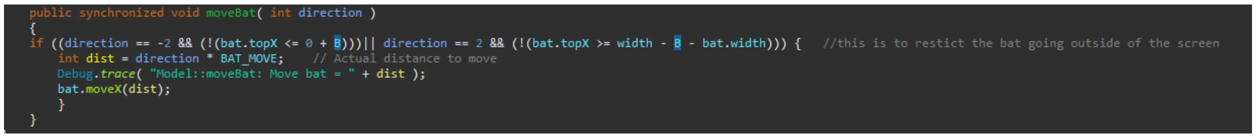
I imagined the window as an X and Y plane and placed a ball and brick on it. And in a summary to check if the ball hit the bricks or bat's side, the game would check if it is intersecting the Y axis, or if it is Intersecting the X axis, or if it is intersecting both the X and Y axis.



```
public boolean hitTopBot(GameObj obj) {

    boolean intersectingX =

            topX+width >= obj.topX &&
            topX <= obj.topX +obj.width; //checking if it is CURRENTLY crossing X axis
            Debug.trace("interesecting X %s", intersectingX + "");

    if (intersectingX) Debug.trace("Intersecting X ");

    boolean abouttointersectY =
            topY+height >= obj.topY - Model.BALL_MOVE &&
            topY <= obj.topY + obj.height + Model.BALL_MOVE; //checking if it is GOING TO  crossing X axis

    boolean hitvertical =
            intersectingX && abouttointersectY;

    return hitvertical;
}


public boolean hitSide(GameObj obj) {

    boolean intersectingY =
            topY+height >= obj.topY &&
            topY <= obj.topY + obj.height; //checking if it is CURRENTLY crossing X axis
            Debug.trace("interesecting Y %s", intersectingY + "");



    if (intersectingY) Debug.trace("Intersecting y ");

    boolean abouttointersectX =
            topX+width >= obj.topX - Model.BALL_MOVE&&
            topX <= obj.topX +obj.width + Model.BALL_MOVE; //checking if it is GOING crossing X axis


    boolean hitSide=
            intersectingY && abouttointersectX;



    return hitSide;


}
```

The code above shows the collision check for the ball. To break it down, it checks if the ball is intersecting the X and/or Y axis and if it is ‘GOING TO’ intersect it the next frame. The reason for me to check if it is going to intersect in the future is to determine where it is going to hit, if it is hitting the top or bottom, the Y will be intersected first and X if it is hitting the side first.

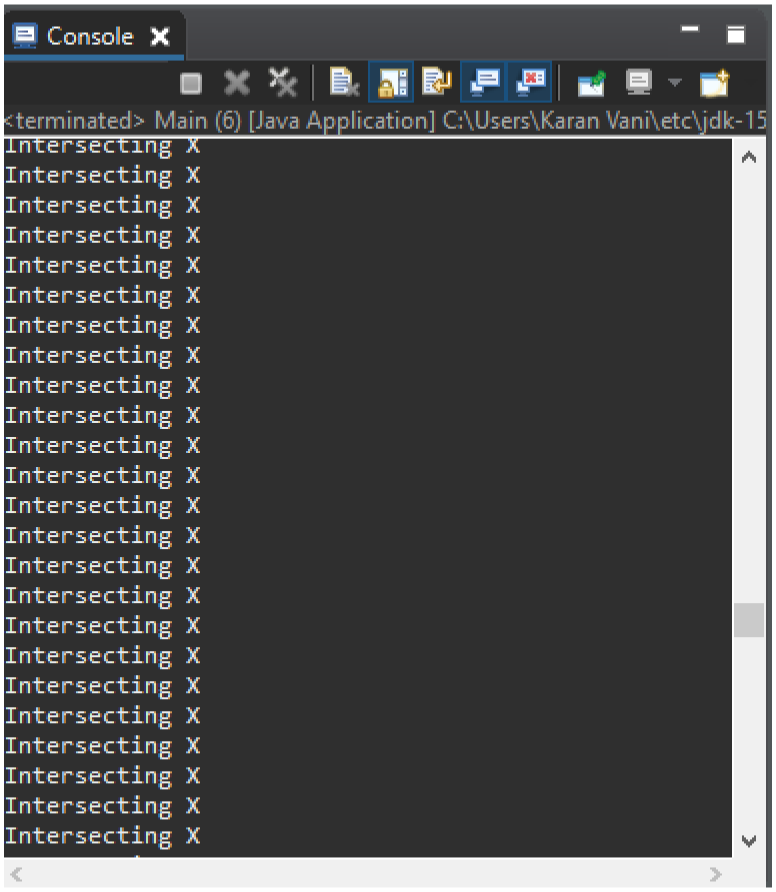
This new collision check got rid of bug causing the ball getting stuck in the bat, but unfortunately presented a new bug. The new bug will cause the ball to get stuck inside of the bat if the player moves the bat the same instant/frame the ball is going to hit the side of the bat.



```
public synchronized void moveBat( int direction )
{
if ((direction == -2 && (!(bat.topX <= 0 + B)))|| direction == 2 && (!(bat.topX >= width - B - bat.width))) {   //this is to restict the bat going outside of the screen
    int dist = direction * BAT_MOVE;    // Actual distance to move
    Debug.trace( "Model::moveBat: Move bat = " + dist );
    bat.moveX(dist);
    }
}
```

Adding another check, I made sure the bat will stay inside of the screen by checking if the bat WILL move outside the width of the screen or not. If it does, the Model Class will not allow the user to move the bat in the direction which will take the bat off screen.

## Tests



```
<terminated> Main (6) [Java Application] C:\Users\Karan Vani\etc\jdk-15
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
Intersecting X
```

To test the code, I continually played the game the right way (which is how the user would play the game to win) and by trying to break the game in the game. The code has 'Debug traces' and a bunch of logs to all the decisions the game will make, as in if the ball is intersecting X and/or Y and so on.

## Reflection

Looking at the final version of the game compared to the original. The game has come a far way. Was I satisfied with the final version? Not as much. Due to the time restrictions, a lot of features have been missed out on, such as super bricks that spawn at a random time frame and once hit by the ball, it gains speed, better optimized, proper AudioClip sounds for the hit sounds. There are a lot of bugs in the game which have not been fixed, such as a cluster of bricks being broken right in the beginning. The thing I feel I exceled at was redesigning the entire collision detection. Coding the game from scratch up would have given more me flexibility and more control and a deeper understanding of the code, this would have also given me a chance to use better access modifiers and data types that would have enhanced the memory usage and performance of the game. Another thing this game would’ve been great would have been automated testing, making a class that hard tests the game In the background or foreground. The game in general has been improved a lot without the features.

A detailed commit history has also been uploaded on Github.