IMPLEMENTATION OF BALL-BREAKER GAME USING SWING AND AWT

A PROJECT REPORT

Submitted by

ABHAY SHAJI [Reg No: RA2112704010006]

Under the Guidance of

Dr. Elangovan.G

(Assistant Professor, Department of Data Science and Business Systems)

In partial fulfillment of the Requirements for the Degree

of

Masters of Technology (Integrated)

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

NOVEMBER 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this project report titled "**IMPLEMENTATION OF BRICK-BREAK GAME USING JAVA SWING"** is the bonafide work of **"ABHAY SHAJI [Reg No: RA2112704010006]"** carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. Elangovan.G	Dr. M Lakshmi
GUIDE	**HEAD OF THE DEPARTMENT**
Assistant Professor	Dept. of Data Science and Business Systems
Dept. of Data Science and Business Systems	

Signature of Examiner

Signature of External Examiner

ABSTRACT

To study the concepts of Object-Oriented Programming in Java using the implementation of Swing in a game. It introduces the remake version of the Brick-Breaker game. It was developed by JSwing framework of Java. This project aims to bring the fun and simplicity of breaking bricks with some new features. It will include better placement of the target in the boxes with Randomizer Algorithm. The player controls a small, thin box, which moves left and right on the lower bordered plane, bouncing the ball or picking (or some other item), trying to avoid the ball going out of the playing area. The box shape object bounces the ball and when the ball hits the brick, the brick breaks .The user controls the direction of the bounce box (left, or right), and the bounce box body follows.

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr T.V.Gopal**, for his invaluable support.

We wish to thank **Dr Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department**, Dr M. Lakshmi** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We register our immeasurable thanks to our Faculty Advisor**, Dr K Shantha Kumari, Ph.D.** , Data Science and Business Systems, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, **Elangovan.G, Ph.D.**, Data Science and Business Systems, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under his/her/their mentorship. He/She/They provided me with the freedom and support to explore the research topics of my interest. Her/His/Their passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Data Science and Business Systems staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

ABHAY SHAJI

LIST OF FIGURES: -

LIST OF ABRIAVIATIONS

AI	Artificial Intelligence
GUI	Graphical User Interface
AWT	Abstract Window Toolkit
MVC	Model–view–controller
API	Application Programming Interface

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER 1

1.1INTRODUCTION

Brick Breaker a Breakout clone[2] which the player must smash a wall of bricks by deflecting a bouncing ball with a paddle. The paddle may move horizontally and is controlled with the BlackBerry's trackwheel, the computer's mouse or the touch of a finger (in the case of touchscreen). The player gets 3 lives to start with; a life is lost if the ball hits the bottom of the screen. When all the bricks have been destroyed, the player advances to a new, harder level. There are 34 levels. Many levels have unbreakable silver bricks. If all lives are lost, the game is over. There are many versions of brick breaker, some in which you can shoot flaming fireballs or play with more than one ball if the player gets a power up..

1.2 GAMEPLAY

Brick Breaker a Breakout clone[2] which the player must smash a wall of bricks by deflecting a bouncing ball with a paddle. The paddle may move horizontally and is controlled with the BlackBerry's trackwheel, the computer's mouse or the touch of a finger (in the case of touchscreen). The player gets 3 lives to start with; a life is lost if the ball hits the bottom of the screen. When all the bricks have been destroyed, the player advances to a new, harder level. There are 34 levels. Many levels have unbreakable silver bricks. If all lives are lost, the game is over. There are many versions of brick breaker, some in which you can shoot flaming fireballs or play with more than one ball if the player gets a power up.

1.3 HISTORY

The Breakout design dates back to the arcade game Blockade, developed and published by Gremlin in 1976. It was cloned as Bigfoot Bonkers the same year. In 1977, Atari, Inc. released two Blockade-inspired titles: the arcade game Dominos and Atari VCS game Surround. Surround was one of the nine Atari VCS launch titles in the US and was sold by Sears under the name Chase. That same year, a similar game was launched for the Bally Astrocade as Checkmate.

The first known home computer version, titled Worm, was programmed in 1978 by Peter Trefonas for the TRS-80, and published by CLOAD magazine in the same year. This was followed shortly afterwards with versions from the same author for the Commodore PET and Apple II. A clone of the Hustle arcade game, itself a clone of Blockade, was written by Peter Trefonas in 1979 and published by CLOAD. An authorized version of Hustle was published by Milton Bradley for the TI-99/4A in 1980. The single-player Snake Byte was published in 1982 for Atari 8-bit computers, Apple II, and VIC-20; a snake eats apples to complete a level, growing longer in the process. In Snake for the BBC Micro (1982), by Dave Bresnen,

CHAPTER 2

LITERATURE REVIEW

2.1 A simple game written in java applying OOP principles

The first step is to write a code which puts the various pieces on the playing board. Itprobably makes sense to implement procedure to run (which drives the game) as two methodcalls: one that sets up the game and the other that plays it. The most important part of the setup consists of creating rows of bricks on the top of the game. The number of dimensions andspacing of the bricks, and the distance from the top of the window to the first line of bricks, are specified using some named constants given in class Breakout. The next step is creating a paddle. You will need to reference the paddle often.Be careful that the paddle stays completely on the board even if the mouse moves off the board. The next step is an interesting part. In order to make brick breaker or breakout into a real game, you have to be able to tell when the ball collides with another object in the window. However, the ballis not a single point. It occupies physical area, so it may collide with something on the screen even though its centre does not. The easiest thing to do is typically of the simplifying assumptions made in real computer games are to check a few carefully chosen points on the outside of the ball and see whether any of those points has collided with anything. As soon as you find something at one of those points (other than the ball of course) you can declare that the ball has collided with that object.

Java should be fast enough for most games, so where is the catch? There are some reasons:

• Many game developers lack expertise in java.

• Lack of good game development frameworks is common.

• Programmers don't want to accept Java as a games programming language. Most only accept C++ as that?

• No support for game consoles (though the PC market still exists)

In terms of machine speed, Java beats C++ in several scientific computing benchmarks these days. You’ll write pathological code in either language that performs badly if you would like to, however over-all, they're at par and are for an extended time. In terms of memory usage, Java will have some over-head. Hello World could be a 4K program in java. However, that overhead is pretty nonsensical in today's multi GB systems. Finally, Java will have additional of a start-up time. i might not advocate mistreatment Java for brief run-time utilities like UNIX system program line commands. In those cases, start-up can dominate your performance. In an exceedingly game but, it's fairly insignificant. Properly written Java game code doesn't suffer gigahertz pauses. a bit like C/C++ code, it will need some active memory management however to not the amount C/C++ will. As long as you retain your memory usage to either long lived objects (persist for a complete level or game) and really short lived objects (vectors and such, passed around and quickly destroyed when calculation) gigahertz mustn't be an apparent issue.

Java is nice for business logic, servers, and platform freelance code that has got to run faithfully. There area unit many factors why Java is not typically employed in games:

• Bounds checking safety mechanisms (marginal performance distinction these days).

• Having to convert between C++ knowledge structures and Java knowledge structures (can't simply copy memory between buffers)

• Several of the books and tutorials follow the gang thus it's arduous to search out non-C++ game dev info

• The core graphics libraries (DirectX and OpenGL) and plenty of off-the-rack engines area unit C/C++ primarily based

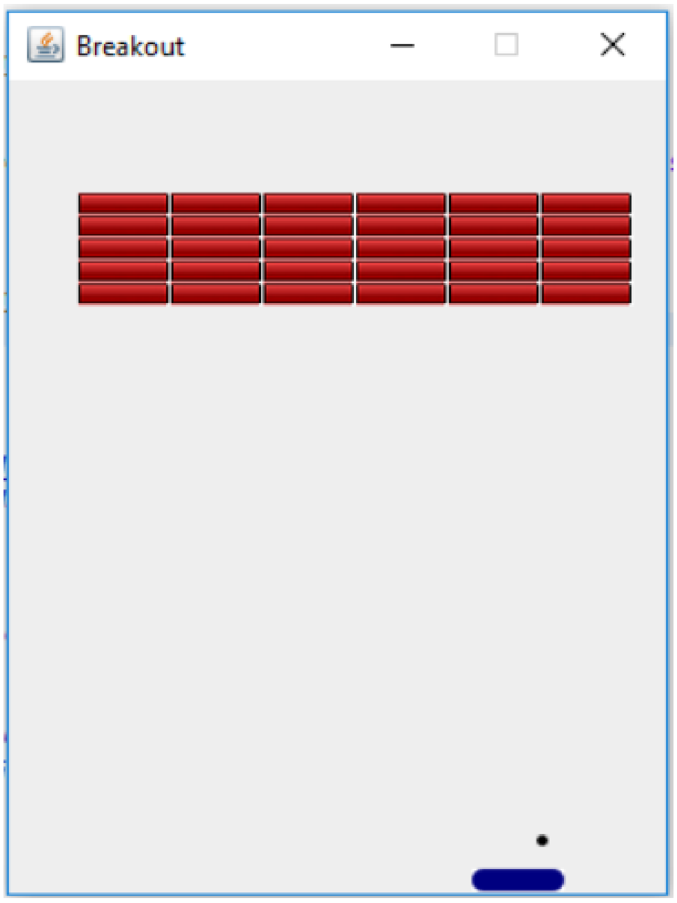
• several games try and run as quick as potential so that they will add additional visually appealing options. It's rough to figure with C++ libraries from bytecode languages like Java (writing a JNI layer) and .net (Lots of marshalling/unmarshalling, api/structure attributes). So it adds quite a bit of work for little benefit.

Game developers wish to be getting ready to the metal and infrequently can write their tight inner loops in assembly. Java does not provide identical level of potential performance, each in terms of consistent speed or memory use. Game programmers conjointly unremarkably use Java, as a result of Java supports multithreading and sockets. Multithreading uses less memory and makes the foremost of obtainable mainframe, while not block the user out once serious processes ar running within the background. Sockets facilitate in building multiplayer games. Plus, Java runs on a virtual machine, thus your game is easier to distribute. In our game, we've got one paddle, one ball and thirty bricks. I actually have created a picture for a ball, paddle and a brick over Inkscape. We have a tendency to use a timer to form a game cycle. we have a tendency to don't work with angles, we have a tendency to merely amendment directions. Top, bottom, left and right. I used to be galvanized by the pybreakout game. it absolutely was developed in PyGame library by Nathan Dawson. First, we'd like to determine on the categories and also the knowledge and ways in them. we have a tendency to will demonstrate Associate in Nursing approach that was 1st projected by Abbott in 1983. The approach states to identify the categories from the matter description of the matter. Staring at the outline, we will produce the jailbreak category wherever the most technique is written. We'll conjointly include classes for the most participants of the game: Ball, Brick, Paddle, and Player. We have a tendency to will create 2 enumeration types: Ball Color and Speed. There, we'll store the possibilities for the ball color and ball speed. We'll have to be compelled to produce 2 extra categories that related to Java: BreakoutFrame and BreakoutPanel. These are the window and also the drawing panel for our application, severally. Lastly, we'll produce a Breakout Shape category that will be the taxon of all the shapes which will be displayed. Since drawing the paddle, ball, and brick are similar, it's cheap to form one taxon. In jailbreak, the initial configuration of the planet seems as shown on the correct. the coloured rectangles within the high a part of the screen are bricks, and also the slightly larger parallelogram at all-time low is that the paddle. The paddle is in a very mounted position within the vertical dimension, however moves back and forth across the screen in conjunction with the mouse till it reaches the sting of its house. An entire game consists of 3 turns. On every flip, a ball is launched from the middle of the window toward very cheap of the screen at a random angle. That ball bounces off the paddle and also the walls of the planet, in accordance with the physical principle usually expressed as "the angle of incidence equals the angle of reflection" (which seems to be terribly simple to implement as mentioned later during this handout). Thus, when 2 bounces—one off the paddle and one off the proper wall—the ball might need the flight shown within the second diagram. (Note that the dotted line is there only to show the ball's path and won't appear on the screen.) Now comes the interesting part. In order to create jailbreak into a true game, you've got to be ready to tell whether or not the ball is colliding with another object within the window. As scientists typically do, it helps to start by creating a simplifying assumption and so reposeful that assumption later. Suppose the ball were one purpose instead of a circle.

CHAPTER 3

EXISTING SYSTEM

Brick Breaker a Breakout clone[2] which the player must smash a wall of bricks by deflecting a bouncing ball with a paddle. The paddle may move horizontally and is controlled with the BlackBerry's trackwheel, the computer's mouse or the touch of a finger (in the case of touchscreen). The player gets 3 lives tostart with; a life is lost if the ball hits the bottom of the screen. When all the bricks have been destroyed, the player advances to a new, harder level. There are 34 levels. Many levels have unbreakable silver bricks. If all lives are lost, the game is over. There are many versions of brick breaker, some in which you can shoot flaming fireballs or play with more than one ball if the player gets a power up..



1.1 -GAME

Here is a quick overview of how it is designed. In the snippets below,

The size of the panel is set as obj.setBounds(10,10,700,600);
brick width(You can think of this as something similar to a square on a game board.) totalbricks constant defines the maximum number of possible bricks on the board. The size 600x400 defines the size of the board in pixels,
private int playerX = 310;
  private int ballposX = 120; records the initial X position of the ball

  private int ballposY = 350; records the initial Y position of the ball
  private int ballXdir = -1; records the Xdirection of the ball is moving.
  private int ballYdir = -2; records the Y direction of the ball is moving.

When the Ball hits the breick the brick breaks and its value is deleted .

The variable in Totalbricks keeps track of how many bricks are remaining in the pannel we have at any time. In the above example after breaking bricks is decreased to totalbricks- 1. Remember, in computer science, we start counting from 21 not 1.

You control the BOUNCE BOX by using the cursor keys. Clicking the left arrow key changes the direction of the bounce brick to point East, Left to West, Up to The head keeps moving in the same direction until an arrow key is clicked to change its direction.

CHAPTER 4

UML DIAGRAMS

FIG-4.1 CLASS DIAGRAM

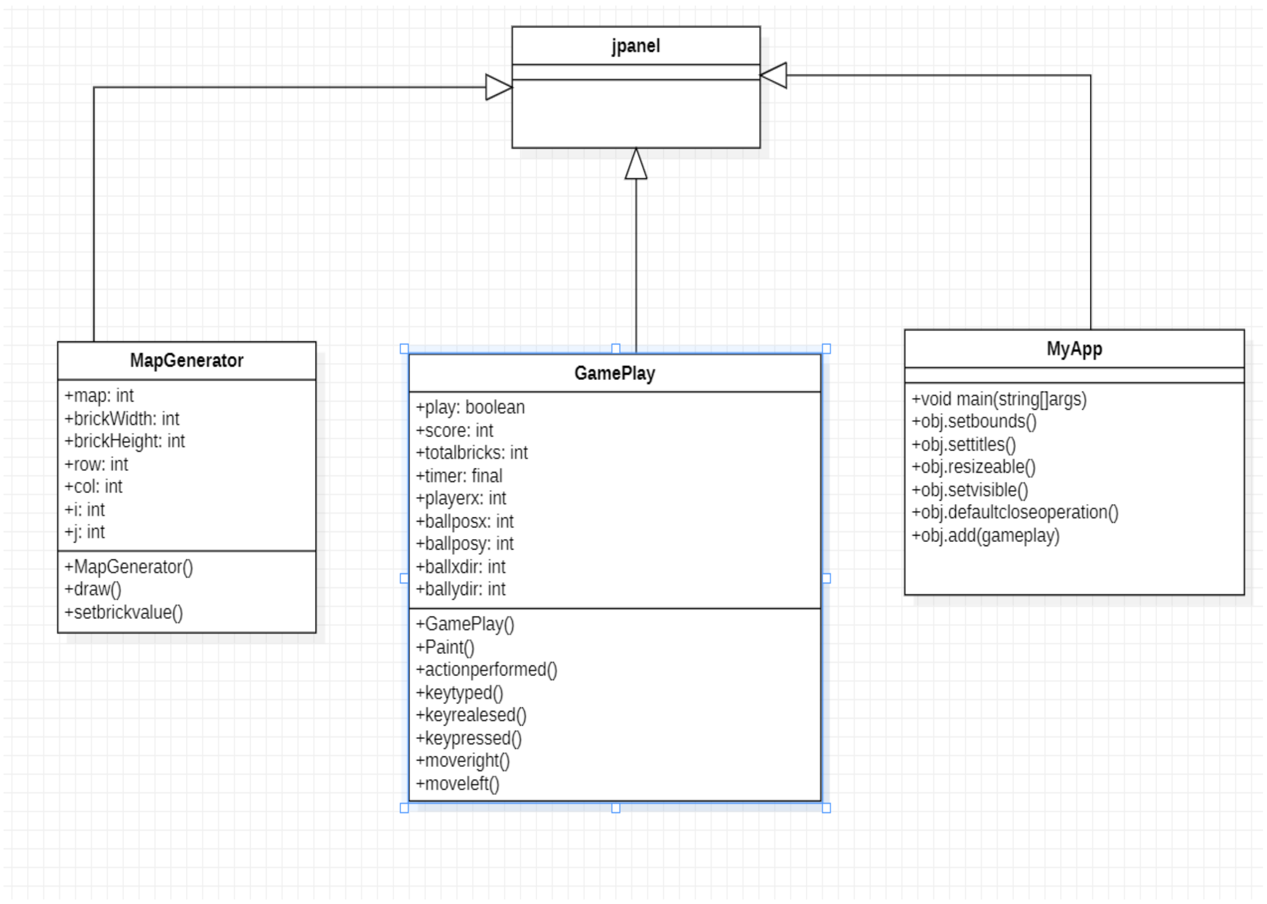


FIG - 4.2 FLOW CHART

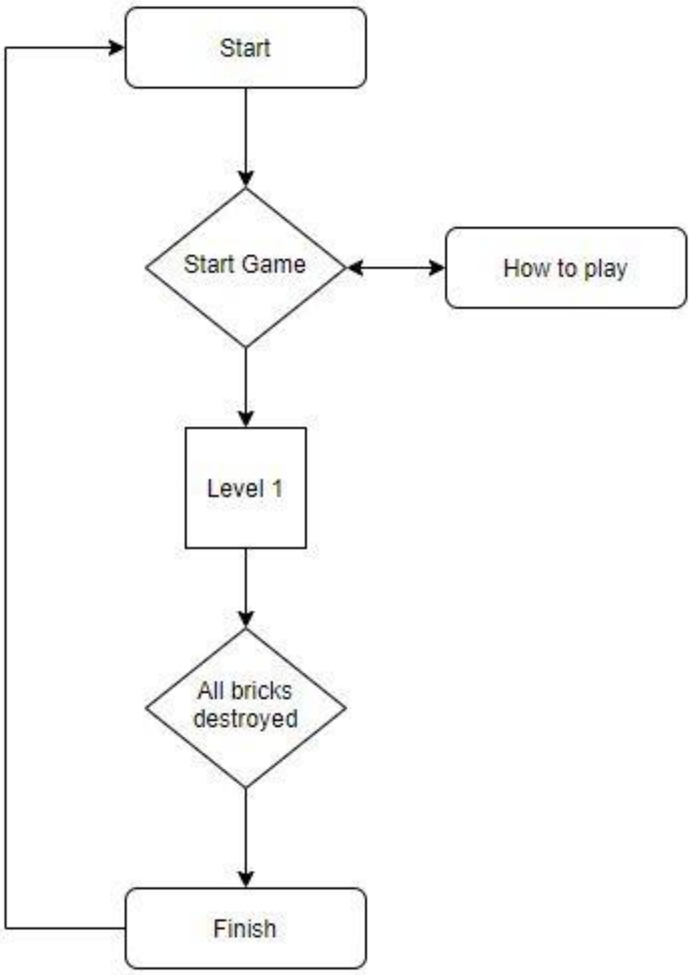


FIG - 4.3 USE CASE DIAGRAM

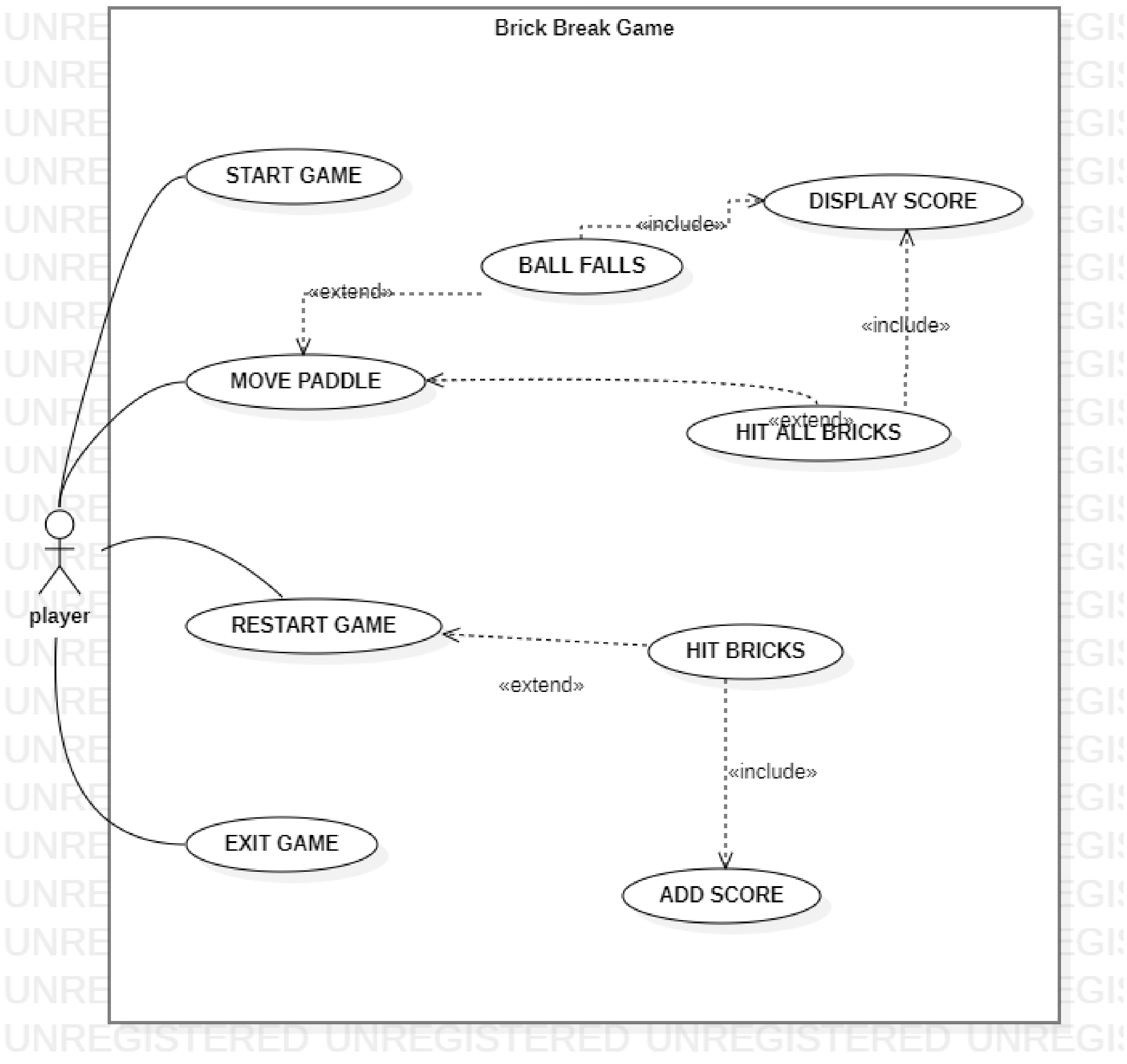
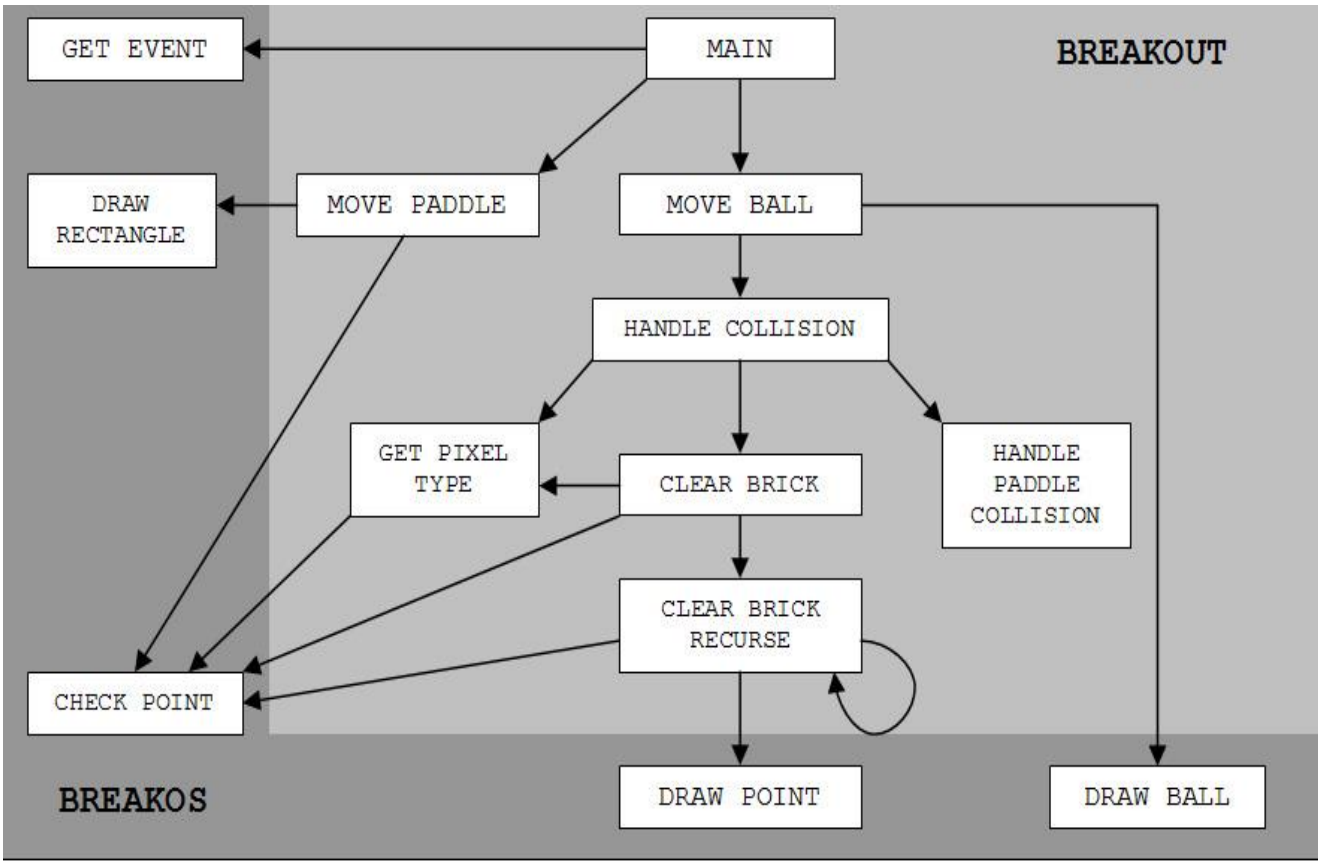


FIG - 4.4 GAME ARCHITECHURE DESIGN:



CHAPTER 5

MODULES EXPLANATION

5.1 OVERVIEW

Game Components are behind the scene classes and methods which make up the game and all the functionalities. The components are mesh together and there are lot of inter relationships between them. The main components are described below:
Main class: Main class is where all the components are connected and it’s the heart and mind of the game. The GUI of the game is represented in this class. The movement is also listened from here & different methods are used for different functionality.

5.2 DataofSquare

The class contains ArrayList that will contain the colours that are to be displayed in the GamePanel using ArrayList<Color>. It contains dark Gray, Blue and white as colours for the blocks. Using lightMeUp to change the colour of each square

5.3 KeyboardListener

Key events indicate when the user is typing at the keyboard. Specifically, key events are fired by the component with the keyboard focus when the user presses or releases keyboard keys. For detailed information about focus, see How to Use the Focus Subsystem.
To define special reactions to particular keys, use key bindings instead of a key listener. In general, you react to only key-typed events unless you need to know when the user presses keys that do not correspond to characters. For example, to know when the user types a Unicode character — whether by pressing one key such as 'a' or by pressing several keys in sequence — you handle key-typed events. On the other hand, to know when the user presses the F1 key, or whether the user pressed the '3' key on the number pad, you handle key-pressed events.
To fire keyboard events, a component must have the keyboard focus.
To make a component get the keyboard focus, follow these steps:

Make sure the component's isFocusable method returns true. This state allows the component to receive the focus. For example, you can enable keyboard focus for a JLabel component by calling the setFocusable(true) method on the label.

Make sure the component requests the focus when appropriate. For custom components, implement a mouse listener that calls the requestFocusInWindow method when the component is clicked.The focus subsystem consumes focus traversal keys, such as Tab and Shift Tab. If you need to prevent the focus traversal keys from being consumed, you can call component.setFocusTraversalKeysEnabled(false) on the component that is firing the key events. Your program must then handle focus traversal on its own. Alternatively, you can use the KeyEventDispatcher class to pre-listen to all key events. The focus page has detailed information on the focus subsystem.You can obtain detailed information about a particular key-pressed event. For example, you can query a key-pressed event to determine if it was fired from an action key. Examples of action keys include Copy, Paste, Page Up, Undo, and the arrow and function keys. You can also query a key-pressed or key-released event to determine the location of the key that fired the event. Most key events are fired from the standard keyboard, but the events for some keys, such as Shift, have information on whether the user pressed the Shift key on the left or the right side of the keyboard. Likewise, the number '2' can be typed from either the standard keyboard or from the number pad.

5.4 Java Swing

Swing is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT]. Swing offers much-improved functionality over AWT, new components, expanded components features, excellent event handling with drag and drop support.
Swing has about four times the number of User Interface [UI] components as AWT and is part of the standard Java distribution. By today’s application GUI requirements, AWT is a limited implementation, not quite capable of providing the components required for developing complex GUI’s required in modern commercial applications. The AWT component set has quite a few bugs and really does take up a lot of system resources when compared to equivalent Swing resources. Netscape introduced its Internet Foundation Classes [IFC] library for use with Java. Its Classes became very popular with programmers creating GUI’s for commercial applications.
Swing is a Set Of API (API- Set Of Classes and Interfaces)
Swing is Provided to Design a Graphical User Interfaces

Swing is an Extension library to the AWT (Abstract Window Toolkit) Includes New and improved Components that have been enhancing the looks and Functionality of GUI's Swing can be used to build(Develop) The Standalone swing GUI Apps Also as Servlets And Applets It Employs model/view design architecture Swing is more portable and more flexible than AWT, The Swing is built on top of the AWT Swing is Entirely written in Java Java Swing Components are Platform-independent And The Swing Components are lightweight Swing Supports Pluggable look and feels And Swing provides more powerful components such as tables, lists, Scrollpanes, Colourchooser, tabbedpane, etc Further Swing Follows MVC Many programmers think that JFC and Swing is one and the same thing, but that is not so. Advance features such as JTable, JTabbedPane, JScollPane etc Java is a platform-independent language and runs on any client machine, the GUI look and feel, owned and delivered by a platform specific O/S, simply does not affect an application's GUI constructed using Swing components.

Lightweight Components: Starting with the JDK 1.1, its AWT supported lightweight component development. For a component to qualify as lightweight, it must not depend on any non-Java [O/s based) system classes. Swing components have their own view supported by Java's look and feel classes

Pluggable Look and Feel: This feature enables the user to switch the look and feel of Swing components without restarting an application. The Swing library supports components look and feel that remains the same across all platforms wherever the program runs. The Swing library provides an API that gives real flexibility in determining the look and feel of the GUI of an application

Swing Classes Hierarchy

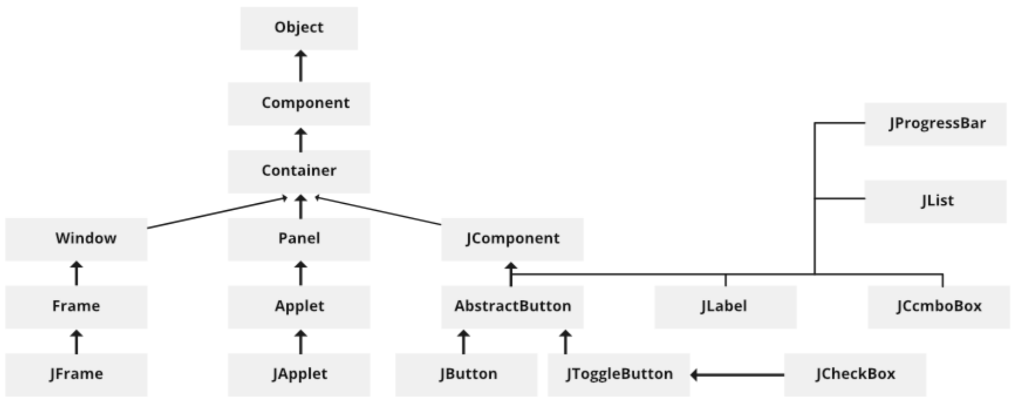


FIG 5.4.1

The MVC Connection

In general, a visual component is a composite of **three distinct aspects:**

The way that the component looks when rendered on the screen The way such that the component reacts to the user

The state information associated With the component

Over the years, one component architecture has proven itself to be exceptionally effective:- **Model-View-Controller** or **MVC** for short.

In MVC terminology, the **model** corresponds to the state information associated with the Component

The **view** determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.

The **controller** determines how the component reacts to the user

The simplest Swing components have capabilities far beyond AWT components as follows:

Swing buttons and the labels can be displaying images instead of or in addition to text

The borders around most Swing components can be changed easily. For example: it is easy to put a 1 pixel border around the outside of a Swing label

Swing components do not have to be rectangular. Buttons, for example, can be round

Now The Latest Assertive technologies such as screen readers can easily getting the information from Swing components. For example: A screen reader tool can easily capture the text that is displayed on a Swing button or label

5.5 JAVA AWT: -

Java AWT is an API that contains large number of classes and methods to create and manage graphical user interface (GUI) applications. The AWT was designed to provide a common set of tools for GUI design that could work on a variety of platforms. The tools provided by the AWT are implemented using each platform's native GUI toolkit, hence preserving the look and feel of each platform. This is an advantage of using AWT. But the disadvantage of such an approach is that GUI designed on one platform may look different when displayed on another platform that means AWT component are platform dependent.

AWT is the foundation upon which Swing is made i.e Swing is a improved GUI API that extends the AWT. But now a days AWT is merely used because most GUI Java programs are implemented using Swing because of its rich implementation of GUI controls and light-weighted nature.The hierarchy of Java AWT classes are given below, all the classes are available in **java.awt** package.

Component class

Component class is at the top of AWT hierarchy. It is an abstract class that encapsulates all the attributes of visual component. A component object is responsible for remembering the current foreground and background colors and the currently selected text font.

Container

Container is a component in AWT that contains another component like button, text field, tables etc. **Container** is a subclass of component class. **Container** class keeps track of components that are aded to another component.

Panel

Panel class is a concrete subclass of **Container**. Panel does not contain title bar, menu bar or border. It is container that is used for holding components.

Window class

Window class creates a top level window. Window does not have borders and menubar.

Frame

Frame is a subclass of **Window** and have resizing canvas. It is a container that contain several different components like button, title bar, textfield, label etc. In Java, most of the AWT applications are created using **Frame** window. Frame class has two different constructors,

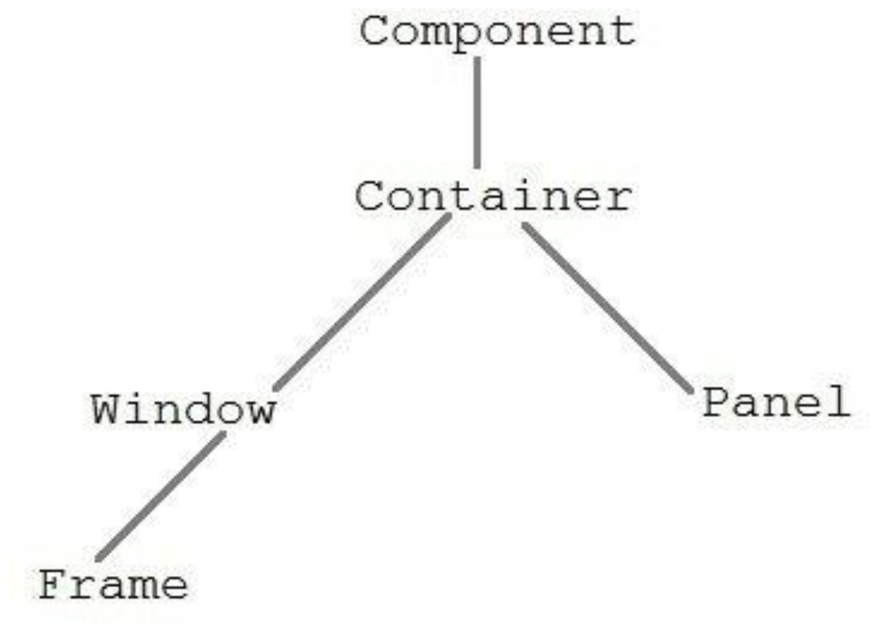


FIG 5.5.1 -AWT CLASSES

CHAPTER 6

CODE

In Gameplay.java

```
package com.mycompany.brick;

import javax.swing.JPanel;
import javax.swing.Timer;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

/**
 *
 * @author chinm
 */
public class  GamePlay extends JPanel implements KeyListener, ActionListener {

    private boolean play = false;
    private int score = 0;
    private int totalbricks = 21;
    private final Timer Timer;
    private int playerX = 310;
    private int ballposX = 120;
    private int ballposY = 350;
    private int ballXdir = -1;
    private int ballYdir = -2;
    private com.mycompany.brick.MapGenerator map;

    public GamePlay() {
        map = new com.mycompany.brick.MapGenerator(3, 7);
        addKeyListener(this);
        setFocusable(true);
        setFocusTraversalKeysEnabled(false);
        int delay = 8;
        Timer = new Timer(delay, this);
        Timer.start();
    }

    public void paint(Graphics g) {
        g.setColor(Color.white);
        g.fillRect(1, 1, 692, 592);

        map.draw((Graphics2D) g);

        g.setColor(Color.yellow);
        g.fillRect(0, 0, 3, 592);
        g.fillRect(0, 0, 692, 3);
        g.fillRect(691, 0, 3, 592);

        g.setColor(Color.white);
        g.setFont(new Font("serif", Font.BOLD, 25));
        g.drawString("" + score, 590, 30);

        g.setColor(Color.BLACK);
        g.fillRect(playerX, 550, 100, 8);
        //ball
        g.setColor(Color.GREEN);
        g.fillOval(ballposX, ballposY, 20, 20);

        if (ballposY > 570) {
            play = false;
            ballXdir = 0;
            ballYdir = 0;
            g.setColor(Color.red);
            g.setFont(new Font("serif", Font.BOLD, 30));
            g.drawString("    Game Over Score: " + score, 190, 300);

            g.setFont(new Font("serif", Font.BOLD, 30));
            g.drawString("   Press Enter to Restart", 190, 340);
        }
        if(totalbricks == 0){
            play = false;
            ballYdir = -2;
            ballXdir = -1;
            g.setColor(Color.red);
            g.setFont(new Font("serif",Font.BOLD,30));
            g.drawString("    Game Over: "+score,190,300);

            g.setFont(new Font("serif", Font.BOLD, 30));
            g.drawString("   Press Enter to Restart", 190, 340);


        }

        g.dispose();


    }
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
        Timer.start();

        if (play) {
            if (new Rectangle(ballposX, ballposY, 20, 20).intersects(new Rectangle(playerX, 550,
100, 8))) {
                ballYdir = -ballYdir;
            }

            A:
            for (int i = 0; i < map.map.length; i++) {
                for (int j = 0; j < map.map[0].length; j++) {
                    if (map.map[i][j] > 0) {
                        int brickX = j * map.bricksWidth + 80;
                        int brickY = i * map.bricksHeight + 50;
                        int bricksWidth = map.bricksWidth;
                        int bricksHeight = map.bricksHeight;

                        Rectangle rect = new Rectangle(brickX, brickY, bricksWidth, bricksHeight);
                        Rectangle ballrect = new Rectangle(ballposX, ballposY, 20, 20);

                        if (ballrect.intersects(rect)) {
                            map.setBricksValue(0, i, j);
                            totalbricks--;
                            score += 5;
                            if (ballposX + 19 <= rect.x || ballposX + 1 >= rect.x + bricksWidth) {
                                ballXdir = -ballXdir;
                            } else {
                                ballYdir = -ballYdir;
                            }
                            break A;
                        }
                }


            }
        }


        ballposX += ballXdir;
        ballposY += ballYdir;
        if (ballposX < 0) {
            ballXdir = -ballXdir;
        }
        if (ballposY < 0) {
            ballYdir = -ballYdir;
        }
        if (ballposX > 670) {
            ballXdir = -ballXdir;
        }
    }
    repaint();
}

@Override
public void keyTyped(KeyEvent e) {

}



@Override
public void keyReleased(KeyEvent e) {

}

@Override
```

```
public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
        if (playerX >= 600) {
            playerX = 600;
        } else {
            moveRight();
        }
    }
    if (e.getKeyCode() == KeyEvent.VK_LEFT) {
        if (playerX < 10) {
            playerX = 10;
        } else {
            moveLeft();
        }
    }

    if (e.getKeyCode() == KeyEvent.VK_ENTER) {
        if (!play) {
            ballposX = 120;
            ballposY = 350;
            ballXdir = -1;
            ballYdir = -2;
            score = 0;
            playerX = 310;
            totalbricks = 21;
            map = new com.mycompany.brick.MapGenerator(3, 7);

            repaint();
        }
    }


}
```

```
    public void moveRight ()
    {
        play = true;
        playerX += 20;
    }
    public void moveLeft ()
    {
        play = true;
        playerX -= 20;
    }



}
```

In MapGenerator.java: -

```
package com.mycompany.brick;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;

public class MapGenerator {
    public int[][] map;
    public int bricksWidth;
    public int bricksHeight;
    public MapGenerator(int row , int col){
        map = new int[row][col];
        for (int[] map1 : map) {
            for (int j = 0; j < map[0].length; j++) {
                map1[j] = 1;
            }
        }
        bricksWidth = 540/col;
        bricksHeight = 150/row;
    }
    public void draw(Graphics2D g) {
        for (int i = 0; i < map.length; i++) {
            for (int j = 0; j < map[0].length; j++) {
                if (map[i][j] > 0) {
                    g.setColor(Color.BLUE);
                    g.fillRect(j * bricksWidth + 80, i * bricksHeight + 50, bricksWidth, bricksHeight);

                    g.setStroke(new BasicStroke(3));
                    g.setColor(Color.white);
                    g.drawRect(j * bricksWidth + 80, i * bricksHeight + 50, bricksWidth,
bricksHeight);

                }
            }


        }
    }
    public void setBricksValue(int value,int row,int col)
    {
        map[row][col] = value;

    }

}
```

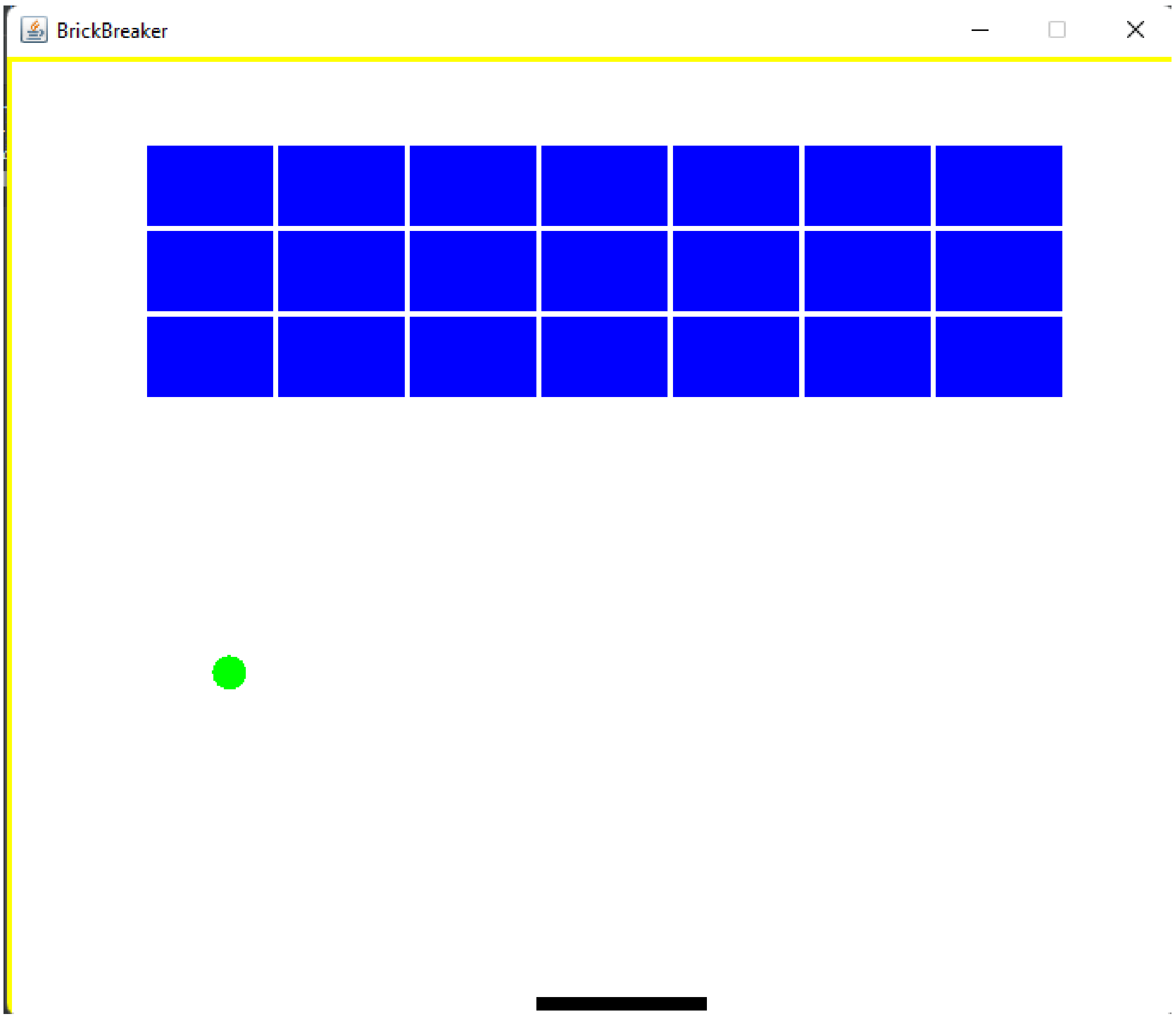
In MyApp.java //the main file

```
package com.mycompany.brick;

import javax.swing.JFrame;
public class MyApp {
    public static void main(String[] args) {
        JFrame obj = new JFrame();
        com.mycompany.brick.GamePlay gameplay = new com.mycompany.brick.GamePlay();
        obj.setBounds(10,10,700,600);
        obj.setTitle("BrickBreaker");
        obj.setResizable(false);
        obj.setVisible(true);
        obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        obj.add(gameplay);
    }

}
```

CHAPTER 7

SCREENSHOTS

FIG – 7.1 Initial panel



WHILE PLAYING

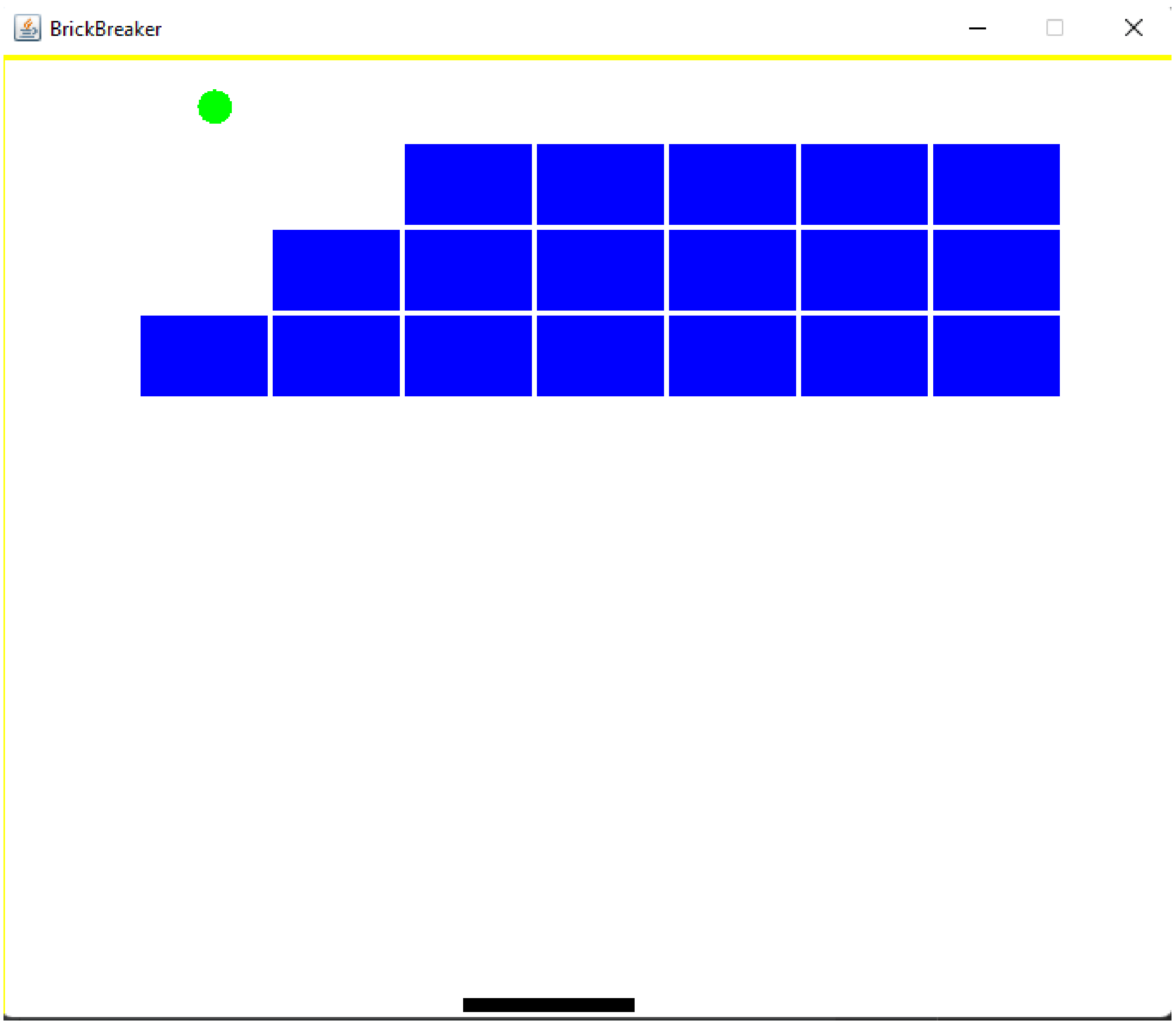
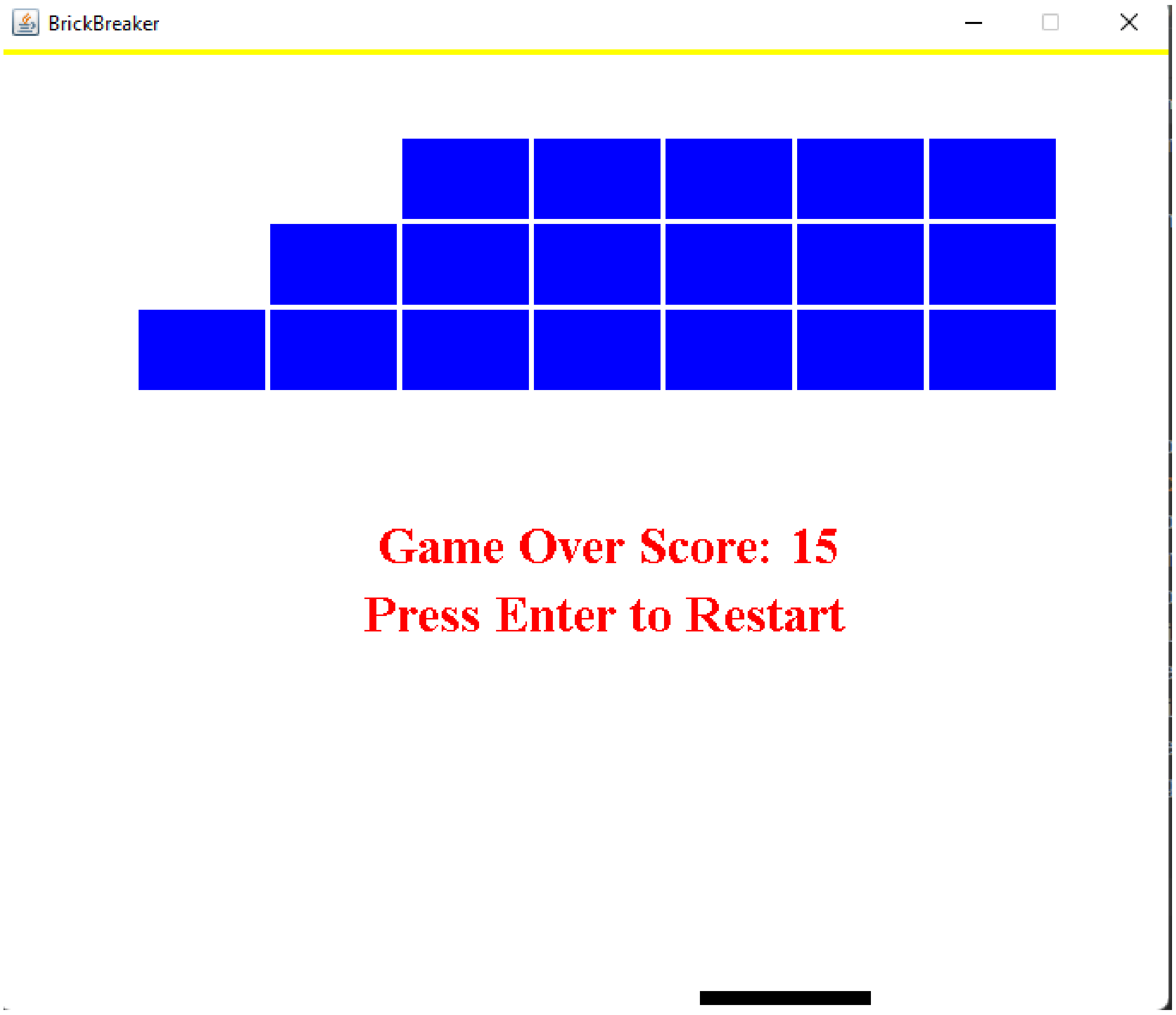


FIG-7.2 OUTPUT2

FAILING THE GAME

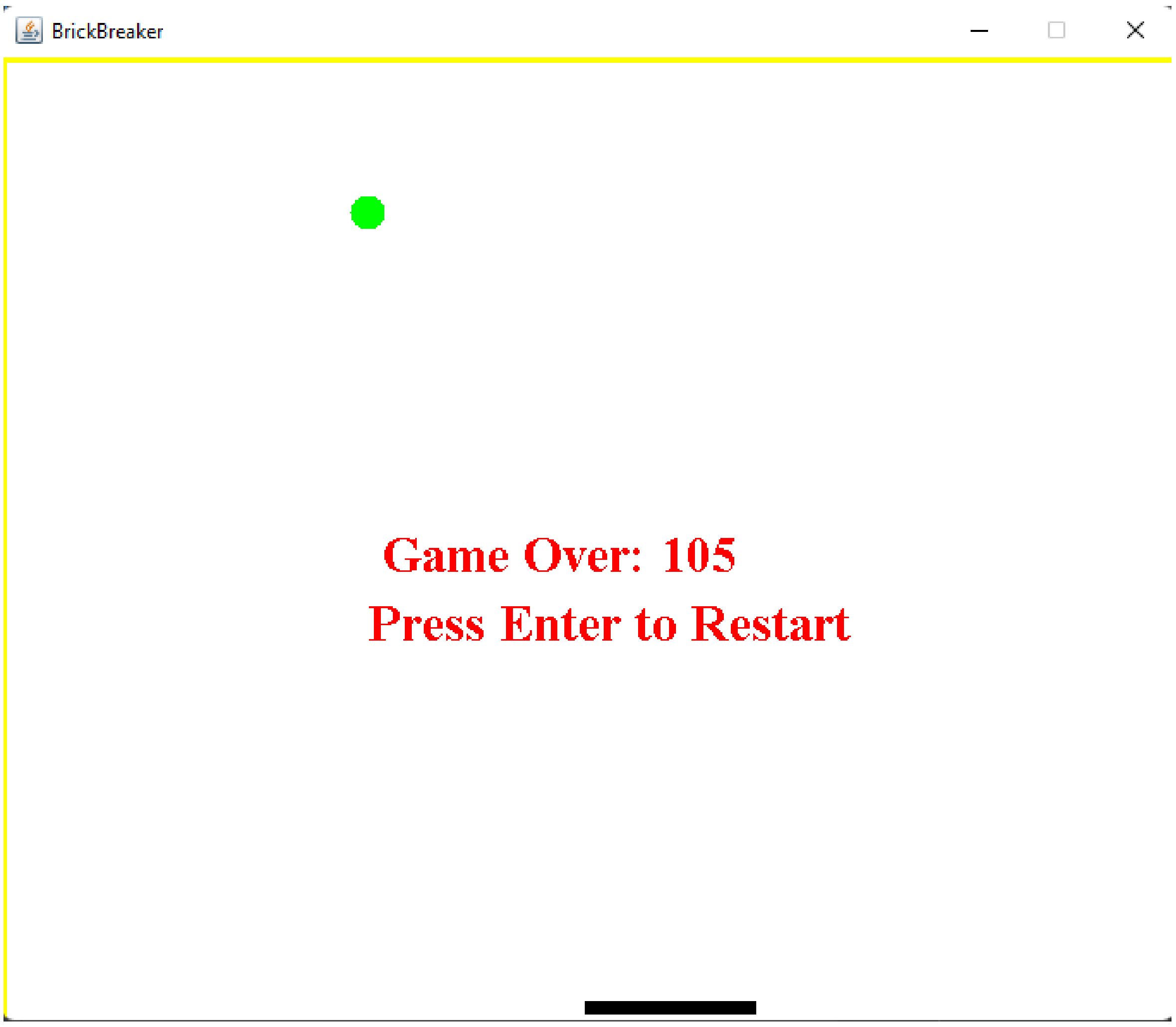


Game Over Score: 15

Press Enter to Restart

FIG7.3- OUTPUT 3

FINISHING THE GAME



Game Over: 105

Press Enter to Restart

FIG7.4 – OUTPUT4

CHAPTER 8

CONCLUSION

We learned several project management techniques used by professionals to develop large scale project. The experience of working in team and integration of modules developed independently, with just requirement specifications, is a very important achievement for the development of the Snake game This project gives us more thrilling, frustrating and also gives us more pleasure. It helps us in many sectors like- planning, designing, developing, managing, programming skill, socket programming and so on. The coding of Brick-breaker was extremely difficult with many errors arising. Many systems had to be written numerous ways before a final working solution was found. For example, two different movement methods were used prior to final version.

REFERENCES

[1] April 2015. 2. E. Lahtinen, K. Ala-Mutka, and H-M Järvinen, "A study of the difficulties of novice programmers," ACM SIGCSE Bulletin. vol. 37. no. 3. ACM, 2005.

[2] Milne and G Rowe. "Difficulties in learning and teaching programming—views of students and tutors." in Education and Information technologies vol 7, no. 1, pp. 55-66, 2002.

[3] Korhonen and L Malmi, "Taxonomy of visual algorithm simulation exercises," in Proceedings of the Third Program Visualization Workshop. Warwick, UK, pp. 118-125, 2004.

[4] M. Hristova, A. Misra, M.Rutter, & R. Mercuri. “Identifying and correcting Java programming errors for introductory computer science students” in ACM SIGCSE Bulletin, vol. 35, No. 1, pp. 153-156. ACM, 2003.

[5]T. Boyle, C. Bradley, P. Chalk, R. Jones, & P. Pickard, "Using blended learning to improve student success rates in learning to program" in Journal of educational Media, vol 28 (2-3), pp. 165-178, 2003. [8] J. Gregory, Game Engine Architecture. CRC Press, 2009.

[6]Aarseth, Espen. 2001. Computer Game Studies, Year One. Game Studies. The International Journal of Computer Game Research. Volume 1, Issue 1, July 2001.

[7] Björk, Staffan. 2008. Games, Gamers, and Gaming Understanding Game Research. Mindtrek 2008, October 7–9, 2008, Tampere.

[8]Björk, Staffan&Holopainen, Jussi. 2004. Patterns in Game Design. Charles River Media Game Development. Charles River Media.

[9]Blessing, Lucienne T.M. &Chakrabarti, Amaresh. 2009. DRM, a Design Research Methodology. Springer. [ICITAIC-2019] ISSN 2348 – 8034 Impact Factor- 5.070 (C)Global Journal Of Engineering Science And Researches 206

[10] Bonsiepe, Gui. 2007. The Uneasy Relationship between Design and Design Research. In Michel, Ralf (Ed.) Design Research Now. Essays and Selected Papers. Birkhäuser. p. 25-41