VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangam", Belagavi: 590 018

A Mobile Application Development Mini Project report on

"Dots and Boxes"

Submitted in partial fulfillment of the requirement for the award of Degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

By

ANUBHAV TEKRIWAL (1AY20CS018)

Under the guidance of

Ms. Swathi U

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
ACHARYA INSTITUTE OF TECHNOLOGY
(Affiliated to Visvesvaraya Technological University, Belagavi)

2022-2023

ACHARYA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi)
Soladevanahalli, Bangalore – 560090

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that the Mobile Application Development mini project entitled **"Dots and Boxes"** is a bonafide work carried out by **ANUBHAV TEKRIWAL (1AY20CS018)** of 6th semester in partial fulfillment for the award of degree of **Bachelor of Engineering in Computer Science & Engineering of the Visvesvaraya Technological University**, **Belagavi**, during the year **2022-2023.** It is certified that all corrections/ suggestions indicated for internal assessments have been incorporated in the Report deposited in the departmental library. The Mini Project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the **Bachelor of Engineering Degree**.

Signature of Guide **Signature of H.O.D**

Ms. Swathi U
(Assistant Professor)
(Dept. of CSE)

Dr. Ajith Padyana
(Professor and HOD)
(Dept. of CSE)

Name of the Examiners **Signature with date**

1. _____

2. _____

ABSTRACT

The "Dots and Boxes" mobile game project is an engaging and interactive multiplayer game designed for Android devices. The objective of the game is for two players to strategically connect dots and form squares. The game offers a user-friendly interface and intuitive touch controls. The project is developed using Android Studio and follows the Android Manifest structure for defining permissions and activities. The manifest file declares the application's activities, including MenuActivity, HelpActivity, GameActivity and ResultActivity. The app provides a user-friendly interface with portrait orientation support. With the implemented code and specifications, the project provides players with an entertaining gameplay experience, encouraging strategic thinking and decision-making.

ACKNOWLEDGEMENT

I express my gratitude to my institution and management for providing me with good infrastructure, laboratory facilities and inspiring staff, and whose gratitude was of immense help in completion of this mini project successfully.

I express my sincere gratitude to our principal, **Dr. M M Rajath Hegde** and vice principal, **Prof. Marigowda C K** for providing me the required environment and for their valuable suggestions.

My sincere thanks to **Dr. Ajith Padyana,** Professor and Head of the Department, Computer Science and Engineering, Acharya Institute of Technology for his valuable support and for rendering me the resources for this mini project.

I heartily thank **Ms. Swathi U**, Assistant Professor, Department of Computer Science and Engineering, Acharya Institute of Technology who guided me with their valuable suggestions at every stage for completing this Mobile Application Development mini project.

My gratitude is rendered to many people who helped me in all possible ways.

ANUBHAV TEKRIWAL
(1AY20CS018)

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The application made is an engaging and interactive multiplayer game designed for Android devices. The primary objective of the project is to create an engaging multiplayer game that promotes strategic thinking and decision-making. The game aims to entertain users by providing a platform for friendly competition between two players. The game offers a user-friendly interface and intuitive touch controls. The project is developed using Android Studio and mainly based on Java programming language. Java is a platform independent general purpose programming language which uses object-oriented approach.

The project includes a menu with options to start the game or access the help section for instructions. By clicking the start button, two players can engage in the game, with the first player represented by the color red and the second player by blue. After completing the game, players can view their respective scores.

1.1.1 Objective

Develop a gaming application that enables users to participate in a friendly competition. The application should incorporate features such as an instruction section, a start playing option, color indications for players, and a computation mechanism to determine the final results.

1.1.2 Key Features

The proposed system for playing Dots and Boxes on a mobile application offers several key features that enhance accessibility, convenience, and enjoyment for players:

- Digital Platform: The mobile application provides a digital platform for playing Dots and Boxes, eliminating the need for physical game boards. This allows players to engage in the game anytime and anywhere using their mobile devices.
- Intuitive Interface: The mobile application offers an intuitive and user-friendly interface with a digital grid layout. Players can simply tap on the points to draw lines, making the gameplay interactive and engaging.
- Automated Tracking: The application automates the detection of completed squares and provides real-time score updates. This eliminates the manual tracking of moves, scores, and completed squares, saving time and ensuring accurate and instant updates.

- Enhanced Gameplay Experience: The mobile application includes various screens such as a menu screen, a help screen, a gameplay screen, and a results screen. These features contribute to an improved gameplay experience and allow players to participate in friendly competitions.
- Offline Mode: The application offers an offline mode, enabling players to enjoy the game even when they don't have an internet connection. This is especially useful for situations where internet access is limited or unavailable, ensuring uninterrupted gameplay.
- Accessibility and Convenience: By transitioning to a mobile application, the proposed system improves accessibility and convenience for players. Dots and Boxes become readily available and easy to play for all users, eliminating the need for physical components and enabling gameplay on the go.

CHAPTER 2

SYSTEM REQUIREMENTS SPECIFICATION

A software requirement definition is an abstract description of the services which the system should provide, and the constraints under which the system must operate. It should only specify the external behaviour of the system.

2.1 FUNCTIONAL REQUIREMENTS

1. The application should provide a menu option with the ability to select game start or help section.
2. The system should initialize the game when the start button is clicked, setting up the game board and player configurations.
3. The application should allow the two players to take turns and interact with the game board by placing lines to form squares.
4. The system should keep track of each player's score throughout the game and update it accordingly.
5. The application should determine when the game is finished and display the final scores to the players.
6. The system should provide a section where users can read instructions on how to play the game.
7. The application should assign specific colors (e.g., red and blue) to each player for visual distinction.
8. The system should enable users to view the final results, including the scores achieved by each player.

2.2 NON-FUNCTIONAL REQUIREMENTS

1. The application should respond quickly to user actions, providing a smooth and responsive gameplay experience.
2. The system should have an intuitive and user-friendly interface, making it easy for players to understand and navigate the game.
3. The application should be compatible with different devices and operating systems, ensuring broad accessibility for users.

4. The system should be stable and reliable, minimizing crashes, errors, and unexpected behavior during gameplay.
5. The system should be designed in a modular and well-structured manner, allowing for easy maintenance, updates, and future enhancements.
6. The application should be portable, enabling players to enjoy the game on various devices without any significant issues.
7. The system should utilize system resources efficiently, optimizing performance and minimizing resource usage.

2.2.1 HARDWARE REQUIREMENTS

The section of hardware arrangement is an important undertaking related to software part of the development process. It requires software to process and run correctly. Without the correct hardware most of the codes and programs would not have run properly. The process should be powerful enough to handle the entire operations of the applications and the appropriate hardware systems are needed to have sufficient capacity to accumulate the file and applications.

Minimum RAM: 128 Mb

Device: Android

2.2.2 SOFTWARE REQUIREMENTS

The major constituent in the development of an android application is the sector of a compatible identification of resources. Selected software should be acceptable by the hardware as well as the platform we are working upon. It as well should be feasible for the system. Following are the description of the software specification.

1) Android Studio
2) Java
3) XML

CHAPTER 3

METHODOLOGY

3.1 DATA FLOW DIAGRAM

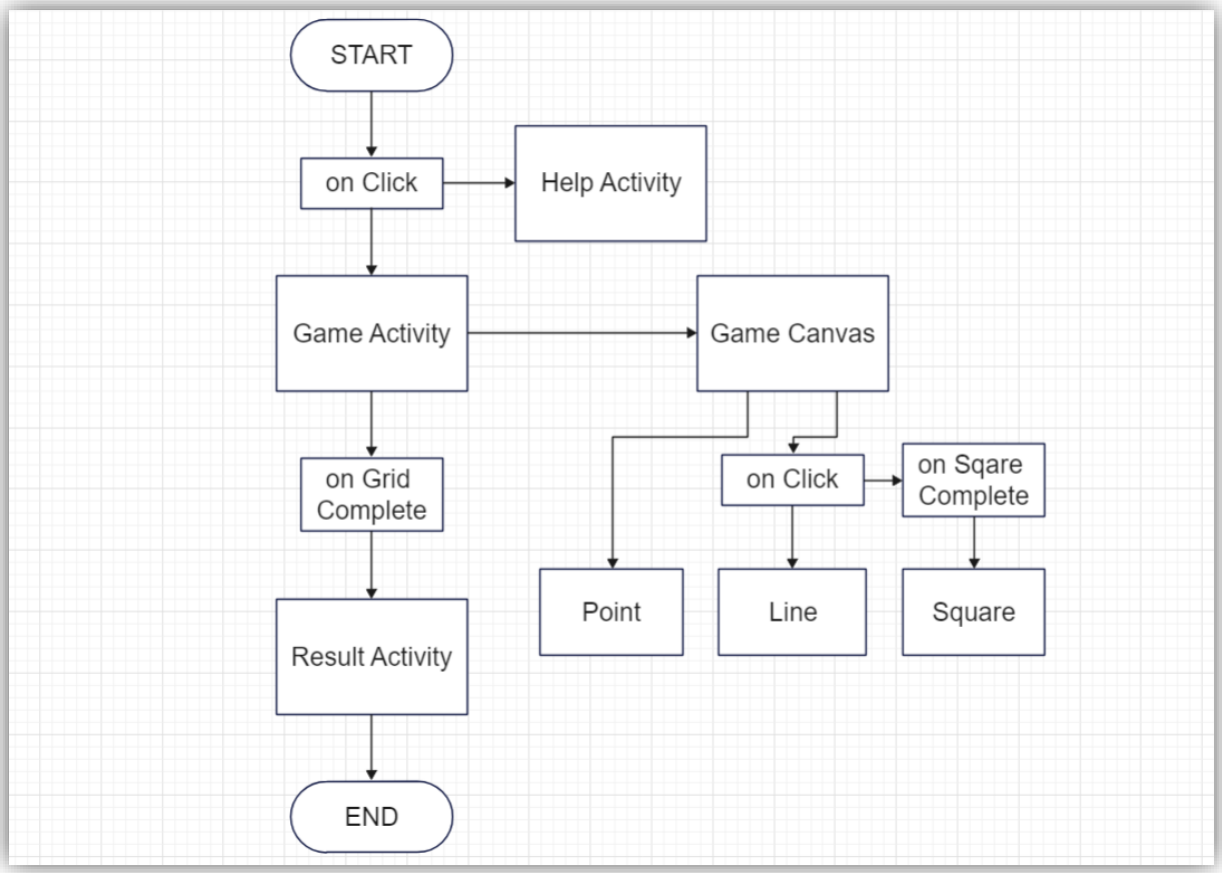

Fig 3.1 Data Flow Diagram Description

The overall data flow and interactions within the system are depicted using **a Level-0 Data Flow Diagram**, as showcased in figure 3.1. A Level-0 DFD provides a high-level view of the system's processes and their connections to external entities. It depicts the system as a single process box and showcases the data exchanges between the system and external entities. By using a Level-0 DFD, I was able to capture the essence of the system's functionality and its interactions.

The above data flow diagram illustrates the flow of actions and data in a game application. The user interacts with the menu options and can access the help section or start playing the game. The game screen uses the **GameCanvas** to draw a grid, where the user can click between points to draw lines. The application checks for completed boxes and draws squares accordingly. Once the game

is completed, the final results are displayed. The above diagram showcases the logical flow of actions and data exchange in the game application.

The above data flow diagram starts with the **MenuActivity**, which serves as the main menu for the game. The user is presented with options to either access the HelpActivity or start playing the game by clicking buttons.

If the user clicks on the "Help" button, the app navigates to the **HelpActivity**. This activity provides instructions and guidelines on how to play the game.
On the other hand, if the user clicks on the "Start" button, the app navigates to the GameActivity. This activity is responsible for managing the gameplay.

The **GameActivity** utilizes the GameCanvas file to draw the grid and manage user interactions. The GameCanvas is a custom view that handles touch events and drawing operations. It displays a grid composed of points, which represent the intersections of the lines.

When the user taps on a point within the grid, the GameCanvas detects the touch event and checks if a line can be drawn at that location. If the touch is valid (inside the grid and no line already exists), a line is drawn on the canvas.

After drawing a line, the GameCanvas checks if a box (also known as a square) is completed by the addition of the new line. If a box is completed, the corresponding square is drawn on the canvas, indicating the player who completed it.

Throughout the gameplay, the GameCanvas keeps track of the scores for each player. Whenever a square is added, the scores are updated accordingly.

Once the grid is fully completed (all squares are filled), the GameCanvas triggers the completion event. The GameActivity then transitions to the ResultsActivity.

In the **ResultsActivity**, the scores of both players are displayed. The activity retrieves the scores from the GameActivity using the getIntent().getIntExtra() method. The scores are shown to the user using appropriate views or widgets.

Additionally, the ResultsActivity provides an option for the user to return to the main menu. If the user clicks the "Main Menu" button, the app navigates back to the MenuActivity.

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1 JAVA

There are various approaches to developing Android applications, and one of the commonly used methods is using the Java programming language in conjunction with Android Studio. Building Android apps with Java involves a combination of familiar Java syntax and Android-specific frameworks and libraries. If you have experience with Java programming, you can easily transition into Android development by leveraging your existing knowledge. However, if you're new to programming, it's important to first grasp the fundamentals of the Java language and gain proficiency in performing basic programming tasks before delving into Android app development. This foundational understanding of Java will provide a solid framework for learning how to utilize Android-specific features and functionalities to build robust and engaging applications.

4.2 APPLICATION IMPLEMENTATION

MenuActivity.java

- This file serves as the entry point for the main menu screen of the Android application. It is responsible for displaying the menu layout (**activity_menu.xml**) to the user. The activity provides two functionalities: showing the help section and starting the game.
- When the user clicks on the "Help" button, the **showHelp** method is triggered. It creates an intent to navigate to the **HelpActivity**, which displays the help content to the user.
- Similarly, when the user clicks on the "Start Game" button, the **startGame** method is called. It creates an intent to navigate to the **GameActivity**, where the actual gameplay takes place.

```
package com.example.anubhav.Activities;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import com.example.anubhav.R;
```

```
public class MenuActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu);
}

    public void showHelp(View view) {
        Intent intent = new Intent(MenuActivity.this, HelpActivity.class);
        startActivity(intent);
    }

    public void startGame(View view) {
        Intent intent = new Intent(MenuActivity.this, GameActivity.class);
        startActivity(intent);
    }
}
```

HelpActivity.java

- The **HelpActivity** file is responsible for displaying the help section of the Android application.
- In the **onCreate** method, the layout file **activity_help.xml** is inflated and set as the content view for the activity. This layout contains the UI components and design elements necessary to present the help content to the user.

```
package com.example.anubhav.Activities;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import com.example.anubhav.R;

public class HelpActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_help);
    }
}
```

GameActivity.java

- The **GameActivity** file is responsible for managing the game screen of the Android application.
- In the **onCreate** method, the layout file **activity_game.xml** is inflated and set as the content view for the activity.
- The code initializes an array **scores** to keep track of the scores for each player. It retrieves the colors array from the resources and sets it on the **GameCanvas** view through binding.gameCanvas.setPlayers(colors). It is a custom view that handles the drawing of the game grid and player interactions.
- A **CanvasListener** is set on the GameCanvas to listen for events such as adding a square or completing the grid. When a square is added, the corresponding player's score is incremented in the scores array. When the grid is completed, the showResults method is called.
- The **showResults** method retrieves the scores from the `scores` array and creates an intent to start the **ResultsActivity**. The scores are passed as extras in the intent, and the activity is started to display the results to the user.

```
package com.example.anubhav.Activities;

import android.content.Intent;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

import com.example.anubhav.R;
import com.example.anubhav.databinding.ActivityGameBinding;
```

```
public class GameActivity extends AppCompatActivity {
    private int[] scores;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityGameBinding binding = ActivityGameBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        String[] colors = getResources().getStringArray(R.array.playerColors);
        binding.gameCanvas.setPlayers(colors);
        scores = new int[2];
        binding.gameCanvas.setListener(new GameCanvas.CanvasListener() {
            @Override
            public void onSquareAdded(int player) {
                scores[player]++;
            }
            @Override
            public void onGridCompleted() {
                showResults();
            }
        });
    }

    public void showResults() {

        int score1 = scores[0];
        int score2 = scores[1];

        Intent intent = new Intent(GameActivity.this, ResultsActivity.class);
        intent.putExtra("score1", score1);
        intent.putExtra("score2", score2);
        startActivity(intent);
    }
```

}

GameCanvas.java

- **setListener(CanvasListener listener):** Sets the canvas listener for receiving events.
- **setPlayers(String[] colors):** Sets the colors for players.
- **onSizeChanged(int w, int h, int oldw, int oldh):** Called when the size of the canvas changes. It calculates the dot width, line width, and grid cell size based on the new dimensions.
- **onDraw(Canvas canvas):** Called when the canvas needs to be redrawn. It draws the squares, lines, and dots on the canvas.
- **onTouchEvent(MotionEvent event):** Called when a touch event occurs on the canvas. It handles touch events and draws lines based on user input.
- **checkForSquare(Line line):** Checks if a square is completed based on the drawn line and adds it to the squares list if necessary.

```
package com.example.anubhav.Activities;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;
import androidx.annotation.Nullable;
import com.example.anubhav.Utils.Line;
import com.example.anubhav.Utils.Point;
import com.example.anubhav.Utils.Square;
import java.util.ArrayList;

public class GameCanvas extends View {
    private final int three =3;
    private int currentPlayer = 1;
    private int fullSizeOfOneCell;
```

```
private float dotWidth, lineWidth;
private boolean squareAdded = false;
private final Paint dotColor, lineColor;
private final Point[][] gridPattern = new Point[4][4];
private final ArrayList<Line> lines;
private final ArrayList<Square> squares;
private CanvasListener listener;
private final String[] dualColors = new String[2];

public GameCanvas(Context context, @Nullable AttributeSet attrs) {
    super(context, attrs);

    lines = new ArrayList<>();
    squares = new ArrayList<>();

    dotColor = new Paint();
    dotColor.setColor(Color.parseColor("#494849"));  //grey dots on grid

    lineColor = new Paint();
    lineColor.setColor(Color.parseColor("#FF6160"));    //red - initial player
}

public void setListener(CanvasListener listener) {
    this.listener = listener;
}

public interface CanvasListener {

    void onSquareAdded(int player);

    void onGridCompleted();
}

public void setPlayers(String[] colors) {
```

```
    System.arraycopy(colors, 0, this.dualColors, 0, 2);
}

@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    super.onSizeChanged(w, h, oldw, oldh);

    // Calculate the radius of each circle
    dotWidth = (float) (w * 0.02);

    // Calculate the width of the lines
    lineWidth = (float) (w * 0.015);
    lineColor.setStrokeWidth(lineWidth);

    // Calculate the size of each grid cell
    int gridSize = (int) (w - 2 * dotWidth);
    fullSizeOfOneCell = gridSize / three;

    // Generate the grid pattern
    for (int i = 0; i <= three; i++) {
        for (int j = 0; j <= three; j++) {
            gridPattern[i][j] = new Point(fullSizeOfOneCell * i, fullSizeOfOneCell * j);
        }
    }
}

@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // Draw the squares which have already been added to the list
    for (Square square : squares) {
        canvas.drawRect(square.rectF, square.paint);
    }
```

```
        // Draw the lines which have already been added to the list
        for (Line line : lines) {
            canvas.drawLine(line.getStartX() + dotWidth, line.getStartY() + dotWidth,
line.getStopX() + dotWidth, line.getStopY() + dotWidth, line.getPaint());
        }

        Point point;
        // Draw the grid cells
        for (int i = 0; i <= three; i++) {
            for (int j = 0; j <= three; j++) {
                point = gridPattern[i][j];
                canvas.drawRect(point.getX(), point.getY(), point.getX() + (2 * dotWidth),
point.getY() + (2 * dotWidth), dotColor);
            }
        }

        // Check if the grid is completed
        if (squares.size() == 9)
            listener.onGridCompleted();
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        double oX = event.getX();       // detected touch X coordinate
        double oY = event.getY();       // detected touch Y coordinate
        double x = oX - dotWidth;       // adjusted touch X coordinate
        double y = oY - dotWidth;       // adjusted touch Y coordinate

        // Convert touch coordinates to grid indices
        double rx = x / fullSizeOfOneCell;  // this tells which cell from the right it is they should be
less than 3 in decimal values
        double ry = y / fullSizeOfOneCell;  // this tells which cell from the bottom it is
        double rxS = Math.floor(rx);        // these are the base of that cell,
```

```
double ryS = Math.floor(ry);        // so possible values can only be 0, 1 or 2
double x1 = rx - rxS;               // determines the coordinate
double y1 = ry - ryS;               // taking origin as the base of that cell

if (event.getAction() == MotionEvent.ACTION_UP) {
   Line line = new Line(lineColor);

   // Check if the touch is within the valid grid range
   // x + y = 1 is the line though the diagonal of the cell
   if (rxS < three && ryS < three) {
      if (x1 >= y1) {
         if (x1 + y1 >= 1) {  //RIGHT
            line.setStartX((float) ((rxS + 1) * fullSizeOfOneCell));
            line.setStartY((float) (ryS * fullSizeOfOneCell));
            line.setStopX((float) ((rxS + 1) * fullSizeOfOneCell));
            line.setStopY((float) ((ryS + 1) * fullSizeOfOneCell));
            line.setOrientation(468);
         }
         else if (x1 + y1 < 1) {  //BOTTOM
            line.setStartX((float) (rxS * fullSizeOfOneCell));
            line.setStartY((float) (ryS * fullSizeOfOneCell));
            line.setStopX((float) ((rxS + 1) * fullSizeOfOneCell));
            line.setStopY((float) (ryS * fullSizeOfOneCell));
            line.setOrientation(135);
         }
      }
      else {
         if (x1 + y1 >= 1) {  //TOP
            line.setStartX((float) (rxS * fullSizeOfOneCell));
            line.setStartY((float) ((ryS + 1) * fullSizeOfOneCell));
            line.setStopX((float) ((rxS + 1) * fullSizeOfOneCell));
            line.setStopY((float) ((ryS + 1) * fullSizeOfOneCell));
            line.setOrientation(135);
         }
```

```
            else if (x1 + y1 < 1) {  //LEFT
                line.setStartX((float) (rxS * fullSizeOfOneCell));
                line.setStartY((float) (ryS * fullSizeOfOneCell));
                line.setStopX((float) (rxS * fullSizeOfOneCell));
                line.setStopY((float) ((ryS + 1) * fullSizeOfOneCell));
                line.setOrientation(468);
            }
        }

        boolean found = false;
        for (Line line1 : lines) {
            if (line.equals(line1)) {
                found = true;
                break;
            }
        }

        // Add the line to the list of lines if it doesn't already exist
        if (!found) {
            if (!squareAdded)
                nextPlayer();
            line.setColor(dualColors[currentPlayer]);
            lines.add(line);
            checkForSquare(line);
            postInvalidate();
            return true;
        }
      }
    }
    return true;
}


  private void checkForSquare(Line line) {
```

```
Square s1, s2;
boolean exists1 = false, exists2 = false;
float startX = line.getStartX(), startY = line.getStartY(), stopX = line.getStopX(), stopY =
line.getStopY();

// X increases in right direction
// Y increases in downward direction
// Check for squares when the line orientation is 135 degrees ie horizontal line
if (line.getOrientation() == 135) {
   Line tL = new Line(startX, startY - fullSizeOfOneCell, startX, startY);     //topLeft line
   Line tR = new Line(stopX, stopY - fullSizeOfOneCell, stopX, stopY);        // topRight
Line
   Line tT = new Line(startX, startY - fullSizeOfOneCell, stopX, stopY -
fullSizeOfOneCell);   //topTop line
   Line bL = new Line(startX, startY + fullSizeOfOneCell, startX, startY);     // bottomLeft
line
   Line bR = new Line(stopX, stopY + fullSizeOfOneCell, stopX, stopY);        //
bottomRight line
   Line bB = new Line(startX, startY + fullSizeOfOneCell, stopX, stopY +
fullSizeOfOneCell);   // bottomBottom line

   // Check if the three lines forming a square exist in the list of lines
   for (Line line1 : lines) {
      if (tL.equals(line1)) {              // basically, means that if the topLeft line of the
horizontal line drawn rn already exists
         for (Line line2 : lines) {
            if (tR.equals(line2)) {        // if topRight also exits
               for (Line line3 : lines) {
                  if (tT.equals(line3)) {    // and if topTop also exists
                     exists1 = true;        // means the above square is to be drawn
                     break;
                  }
               }
            }
```

```
            }
        }
    }

    // Create a new square if the first square exists
    if (exists1) {
        s1 = new Square(startX + dotWidth + lineWidth, startY - fullSizeOfOneCell +
dotWidth + lineWidth, stopX + dotWidth - lineWidth, startY + dotWidth - lineWidth,
dualColors[currentPlayer]);
        s1.lineIndex = lines.indexOf(line);
        squares.add(s1);
    }

    // Check if the three lines forming the second square exist in the list of lines
    for (Line line1 : lines) {
        if (bL.equals(line1)) {
            for (Line line2 : lines) {
                if (bR.equals(line2)) {
                    for (Line line3 : lines) {
                        if (bB.equals(line3)) {
                            exists2 = true;
                            break;
                        }
                    }
                }
            }
        }
    }

    // Create a new square if the second square exists
    if (exists2) {
        s2 = new Square(startX + dotWidth + lineWidth, startY + dotWidth + lineWidth, stopX
+ dotWidth - lineWidth, startY + fullSizeOfOneCell + dotWidth - lineWidth,
dualColors[currentPlayer]);
```

```
            s2.lineIndex = lines.indexOf(line);
            squares.add(s2);
        }
    }
    // Check for squares when the line orientation is not 135 degrees
    else {  // ie if the line drawn was vertical, we need to check the left and right boxes
        Line lT = new Line(startX - fullSizeOfOneCell, startY, startX, startY);
        Line lL = new Line(startX - fullSizeOfOneCell, startY, stopX - fullSizeOfOneCell,
stopY);
        Line lB = new Line(stopX - fullSizeOfOneCell, stopY, stopX, stopY);
        Line rT = new Line(startX, startY, startX + fullSizeOfOneCell, startY);
        Line rR = new Line(startX + fullSizeOfOneCell, startY, stopX + fullSizeOfOneCell,
stopY);
        Line rB = new Line(stopX, stopY, stopX + fullSizeOfOneCell, stopY);

        // Check if the three lines forming a square exist in the list of lines
        for (Line line1 : lines) {
            if (lT.equals(line1)) {
                for (Line line2 : lines) {
                    if (lL.equals(line2)) {
                        for (Line line3 : lines) {
                            if (lB.equals(line3)) {
                                exists1 = true;
                                break;
                            }
                        }
                    }
                }
            }
        }

        // Create a new square if the first square exists
        if (exists1) {
            s1 = new Square(startX - fullSizeOfOneCell + dotWidth + lineWidth, startY +
```

```
dotWidth + lineWidth, startX + dotWidth - lineWidth, stopY + dotWidth - lineWidth,
dualColors[currentPlayer]);
            s1.lineIndex = lines.indexOf(line);
            squares.add(s1);
        }

        // Check if the three lines forming the second square exist in the list of lines
        for (Line line1 : lines) {
            if (rT.equals(line1)) {
                for (Line line2 : lines) {
                    if (rR.equals(line2)) {
                        for (Line line3 : lines) {
                            if (rB.equals(line3)) {
                                exists2 = true;
                                break;
                            }
                        }
                    }
                }
            }
        }

        // Create a new square if the second square exists
        if (exists2) {
            s2 = new Square(startX + dotWidth + lineWidth, startY + dotWidth + lineWidth, startX
+ fullSizeOfOneCell + dotWidth - lineWidth, stopY + dotWidth - lineWidth,
dualColors[currentPlayer]);
            s2.lineIndex = lines.indexOf(line);
            squares.add(s2);
        }
    }

    // Check if at least one square exists
    if (exists1 | exists2) {
```

```
            squareAdded = true;
            listener.onSquareAdded(currentPlayer);
            if (exists1 && exists2)
                listener.onSquareAdded(currentPlayer);
        } else
            squareAdded = false;
    }


    private void nextPlayer() {
        currentPlayer = (currentPlayer +1) %2;
    }
}
```

ResultsActivity.java

- This class represents the activity that displays the results of the game.
- **onCreate(Bundle savedInstanceState):** This method is called when the activity is created. It sets the layout of the activity using the ActivityResultsBinding class, which is automatically generated based on the XML layout file. It retrieves the scores passed from the previous activity using getIntExtra() method and sets them in the appropriate TextViews.
- **showMainMenu(View view):** This method is invoked when the user clicks on the "Main Menu" button. It creates an intent to navigate to the MenuActivity and starts that activity.
- **onBackPressed():** This method is overridden to handle the back button press. Its purpose is to prevent the user from going back to the GameActivity screen after the game has finished. It creates an intent to navigate to the MenuActivity and starts that activity.

```
package com.example.anubhav.Activities;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
```

```
import android.view.View;

import com.example.anubhav.databinding.ActivityResultsBinding;

public class ResultsActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityResultsBinding binding = ActivityResultsBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        int score1 = getIntent().getIntExtra("score1", 0);
        int score2 = getIntent().getIntExtra("score2", 0);
        binding.ts1.setText(String.valueOf(score1));
        binding.ts2.setText(String.valueOf(score2));
    }
    public void showMainMenu(View view) {
        startActivity(new Intent(ResultsActivity.this, MenuActivity.class));
    }
    @Override
    public void onBackPressed() {
        startActivity(new Intent(ResultsActivity.this, MenuActivity.class));
    }
}
```

Point.java

- This class represents a point in a 2D space. It has two private instance variables x and y, which store the coordinates of the point.
- The class provides a constructor that takes the x and y coordinates as parameters and initializes the instance variables accordingly.
- The class also includes getter methods **getX()** and **getY()** to retrieve the x and y coordinates of the point, respectively.

- This Point class is used in the code to represent the points in the grid. It is utilized for drawing the grid cells, determining the touch coordinates, and performing calculations related to the grid.

```
package com.example.anubhav.Utils;

public class Point {

    final private float x;
    final private float y;

    public Point(float x, float y) {
        this.x = x;
        this.y = y;
    }

    public float getX() {
        return x;
    }

    public float getY() {
        return y;
    }
}
```

Line.java

- The Line class represents a line segment in a 2D space.
- It provides methods such as **setStartX()**, **setStartY()**, **setStopX()**, and **setStopY()** to set the coordinates of the starting and ending points of the line.
- The **getStartX()**, **getStartY()**, **getStopX()**, and **getStopY()** methods allow retrieving the coordinates of the line's points.
- The **setOrientation()** method is used to set the orientation of the line.

- The **setColor()** method sets the color of the line using the Paint class from the Android graphics library.
- Constructors like **Line(Paint paint)** and **Line(float startX, float startY, float stopX, float stopY)** are available to create line objects with specified properties.
- The **equals()** method is implemented to compare two lines for equality based on their coordinates.
- The **getPaint()** method returns the Paint object associated with the line, which defines its color and other drawing attributes.
- The Line class is utilized in the application for drawing and manipulating lines within the game canvas.

```
package com.example.anubhav.Utils;
import android.graphics.Color;
import android.graphics.Paint;

public class Line {

   private float startX, startY, stopX, stopY;
   private int orientation;
   Paint paint;

   public int getOrientation() {
      return orientation;
   }

   public void setStartX(float startX) {
      this.startX = startX;
   }

   public void setStartY(float startY) {
      this.startY = startY;
   }
   public void setStopX(float stopX) {
```

```
    this.stopX = stopX;
}
public void setStopY(float stopY) {
    this.stopY = stopY;
}

public float getStartX() {
    return startX;
}
public float getStartY() {
    return startY;
}
public float getStopX() {
    return stopX;
}
public float getStopY() {
    return stopY;
}

public void setOrientation(int orientation) {
    this.orientation = orientation;
}
public void setColor(String color) {
    paint.setColor(Color.parseColor(color));
}

public Line(Paint paint) {
    this.paint = new Paint(paint);
}

public Line(float startX, float startY, float stopX, float stopY) {
    this.startX = startX;
    this.startY = startY;
    this.stopX = stopX;
```

```
        this.stopY = stopY;
    }

    public Paint getPaint() {
        return paint;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Line)) return false;
        Line line = (Line) o;
        return (Float.compare(line.startX, startX) == 0 &&
                Float.compare(line.startY, startY) == 0 &&
                Float.compare(line.stopX, stopX) == 0 &&
                Float.compare(line.stopY, stopY) == 0)
                ||
                (Float.compare(line.startX, stopX) == 0 &&
                        Float.compare(line.startY, stopY) == 0 &&
                        Float.compare(line.stopX, startX) == 0 &&
                        Float.compare(line.stopY, startY) == 0);
    }
}
```

Sqare.java

- The Square class represents a square in a 2D space.
- It contains a **RectF** object named **rectF**, which represents the rectangular bounds of the square.
- The paint object of the **Paint** class is used to define the color and other drawing attributes of the square.
- The constructor **Square(float left, float top, float right, float bottom, String color)** creates a new square object with specified coordinates and color.

- Within the constructor, a new **RectF** object is instantiated and assigned to **rectF**, and a new Paint object is created and assigned to paint.
- The **setColor()** method sets the color of the square using the **Color.parseColor()** method.
- The **set()** method of rectF is called to set the bounds of the square based on the provided coordinates.
- The Square class is utilized in the application to create squares and store their position and appearance information for drawing purposes.

```
package com.example.anubhav.Utils;

import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.RectF;

public class Square {

    public RectF rectF;
    public Paint paint;
    public int lineIndex;

    public Square(float left,float top,float right,float bottom,String color){
        rectF=new RectF();
        paint=new Paint();
        paint.setColor(Color.parseColor(color));
        rectF.set(left, top, right, bottom);
    }
}
```

CHAPTER 5

TESTING

5.1 TEST CASE 1

Table 5.1 Menu Activity Verification

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Pass/Fail
TC_1	Verify Menu Activity	-	Menu screen is displayed with buttons	Menu screen displayed with correct buttons	Pass

- This test case verifies that the menu screen of the application is correctly displayed.
- No specific input is required as it focuses on the initial state of the application.
- The expected output is the menu screen with the necessary buttons or options.
- The actual output is compared against the expected output to determine the test result.
- The test result confirms that the menu screen is displayed correctly with the expected buttons, indicating a successful outcome. Thus, the test case is marked as "Pass."

5.2 TEST CASE 2

Table 5.2 Help Activity Verification

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Pass/Fail
TC_2	Verify Help Activity	-	Help screen is displayed with instructions	Help screen displayed with correct instructions	Pass

- This test case ensures that the help screen of the application is correctly displayed.
- No specific input is required as it focuses on the initial state of the help activity.
- The expected output is the help screen with relevant instructions or guidance for the users.
- The test result indicates that the help screen is displayed correctly with the expected instructions, validating a successful outcome. Hence, the test case is marked as "Pass."

5.3 TEST CASE 3

Table 5.3 Game Activity Verification

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Pass/Fail
TC_3	Verify Game Activity	-	Game screen is displayed with a grid for playing the game	Game screen displayed with correct grid layout	Pass

- This test case ensures that the game activity is correctly displayed with the appropriate grid layout for playing the game.
- No specific input is required as it focuses on the initial state of the game activity.
- The expected output is the game screen with a grid layout consisting of points and lines to facilitate gameplay.
- The test result indicates that the game screen is displayed as expected with the correct grid layout, validating a successful outcome. Therefore, the test case is marked as "Pass."

5.4 TEST CASE 4

Table 5.4 Line Drawing Verification

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Pass/Fail
TC_4	Test Line Drawing	Tap on empty grid space	Line is drawn between the tapped points	Line is drawn correctly	Pass

- This test case focuses on testing the line drawing functionality in the game.
- The input is to tap on an empty grid space to initiate the line drawing action.
- The expected output is that a line should be drawn between the tapped points on the grid.
- The test result indicates that the line is drawn correctly as expected, validating a successful outcome. Therefore, the test case is marked as "Pass."

5.5 TEST CASE 5

Table 5.5 Square Detection Verification

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Pass/Fail
TC_5	Test Square Detection	Complete a square by connecting all sides	Square is filled with the current player's color	Square is filled correctly	Pass

- This test case focuses on testing the square detection functionality in the game.
- The input is to complete a square by connecting all sides with lines.
- The expected output is that the square should be filled with the current player's color.
- The test result indicates that the square is filled as expected, validating a successful outcome. Therefore, the test case is marked as "Pass."

5.6 TEST CASE 6

Table 5.6 Game Completion Verification

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Pass/Fail
TC_6	Test Game Completion	Complete all squares in the grid	Results screen is displayed with the final scores	Results screen displayed with correct scores	Pass

- This test case verifies the game completion functionality.
- The input is to complete all squares in the grid by connecting the lines.
- The expected output is the display of the results screen with the final scores.
- The test result indicates that the game completion is handled correctly, and the actual output matches the expected output. Hence, the test case is marked as "Pass."

5.7 TEST CASE 7

Table 5.7 Results Activity Verification

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Pass/Fail
TC_7	Verify Results Activity	-	Results screen is displayed with scores from the game	Results screen displayed with correct scores	Pass

- This test case validates the functionality of the Results Activity.
- There is no specific input required for this test case.
- The expected output is the display of the Results screen, which shows the scores obtained from the game.
- Based on the comparison of actual and expected outputs, the test case is marked as "Pass," indicating that the Results Activity functions correctly.

5.8 TEST CASE 8

Table 5.8 Back Button Verification

Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Pass/Fail
TC_8	Test Back Button Functionality	Press Back button on Results screen	Menu screen is displayed	Returned to Menu screen	Pass

- This test case verifies the functionality of the Back button on the Results screen.
- The input for this test case is pressing the Back button.
- The expected output is the display of the Menu screen.
- The test case confirms that pressing the Back button successfully navigates back to the Menu screen.
- Based on the comparison of actual and expected outputs, the test case is marked as "Pass," indicating that the Back button functionality works correctly.

CHAPTER 6

RESULTS

6.1 APPLICATION START SCREEN

Fig 6.1 Main Menu

- Upon launching the app, users are greeted with a visually appealing and user-friendly interface.
- The main menu screen (Fig 6.1) consists of two prominent buttons: **Play** and **Help(?)**.
- The Play button allows users to start the game and enter the gameplay screen.
- The Help button provides users with assistance and instructions on how to play the game.
- The buttons are visually distinct and easily identifiable, enhancing user navigation and interaction.
- It provides a clear and straightforward user interface, setting the stage for a seamless user experience.

6.2 GAMEPLAY PROGRESSION SCREENS

6.2.1 Initial Grid View

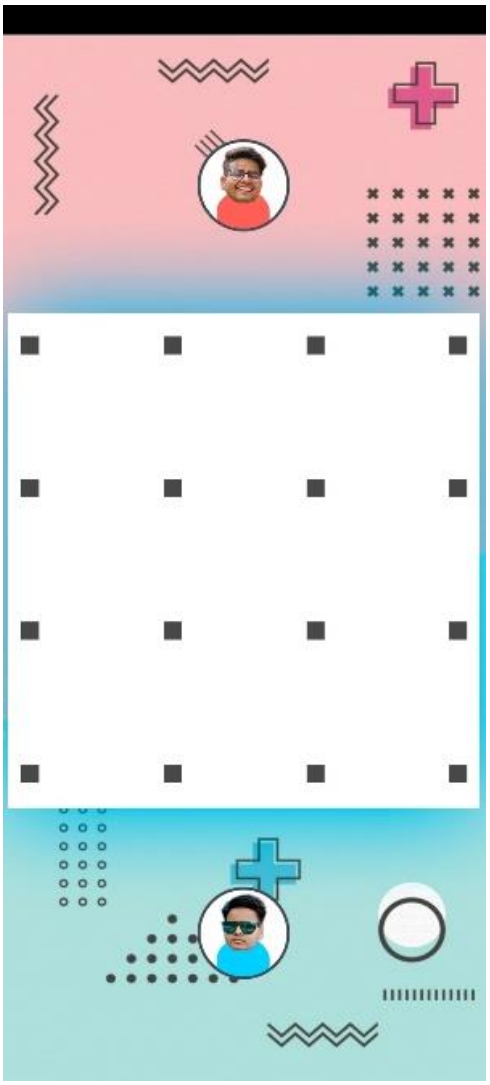

Fig 6.2.1 Initial Grid View

- The initial grid view (Fig 6.2) is displayed after the user presses the **PLAY** button in the main menu.
- It presents a **3x3 grid** of points, representing the game board for the upcoming gameplay.
- The grid is empty, with no lines or squares drawn yet, indicating the starting state of the game.
- The grid serves as the foundation for the gameplay, allowing players to connect points and form lines to create squares.
- It provides a clear visual representation of the game board, setting the stage for strategic moves and gameplay progression.

6.2.2 Lines Drawn

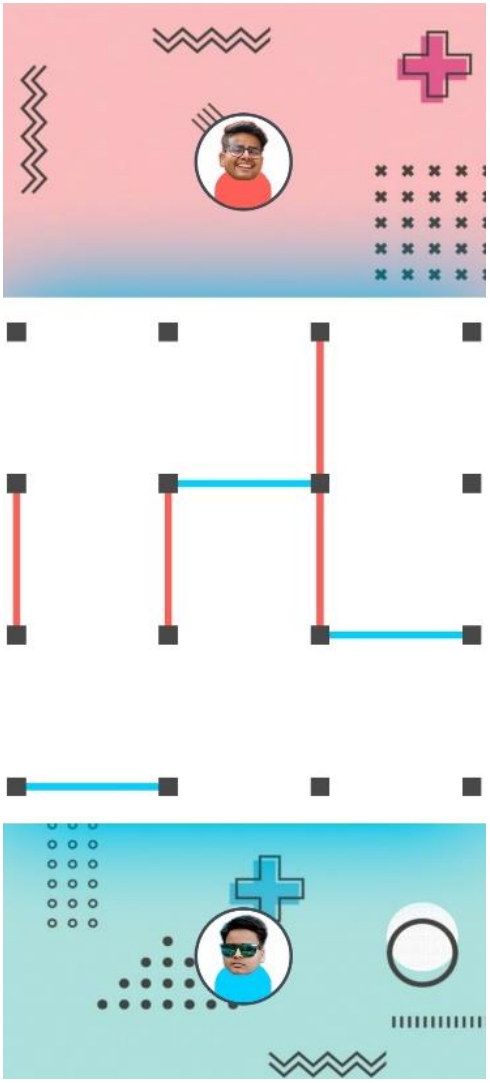

Fig 6.2.2 Lines Drawn in Game

- In Fig 6.3, the snapshot showcases an ongoing game board where players have started drawing lines.
- The lines on the grid connect adjacent points, forming a network of red and blue lines.
- The red and blue lines represent the moves made by the players, indicating their progress in the game.
- At this stage, no squares have been completed yet, as the lines are still in the process of connecting the grid points.
- The snapshot captures the dynamic nature of the gameplay, highlighting the strategic decision-making and the evolving state of the game board.

6.2.3 Square Drawn

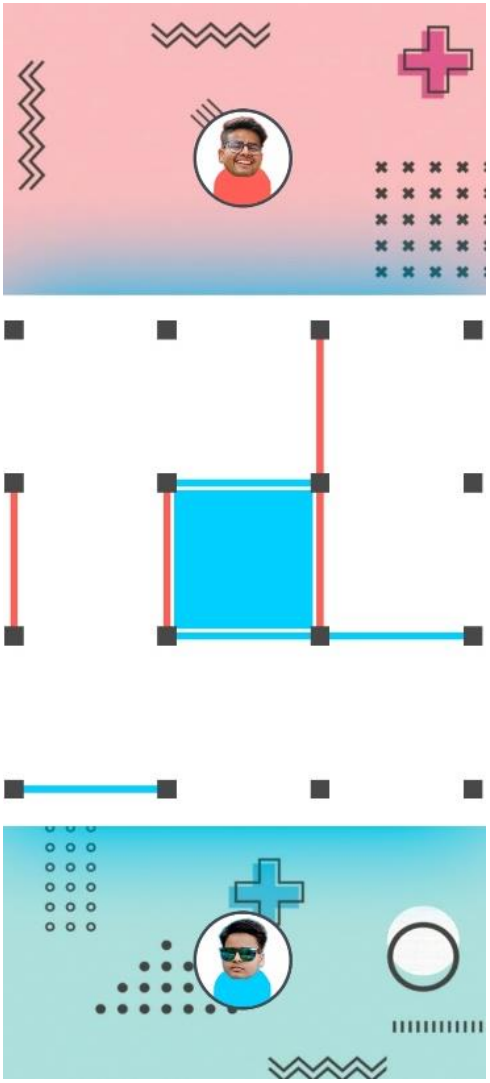

Fig 6.2.3 First Square Completed

- In Fig 6.4, the snapshot depicts the completion of the first square on the game board.
- The square, outlined by the red and blue lines, is now filled with a solid color.
- The solid color indicates that the blue player has successfully closed the square.
- The completion of a square is a significant milestone in the game, as it earns points for the player who closed it.
- The snapshot showcases the progress of the game and the competition between the players to claim squares on the board.

6.2.4 Grid Completed

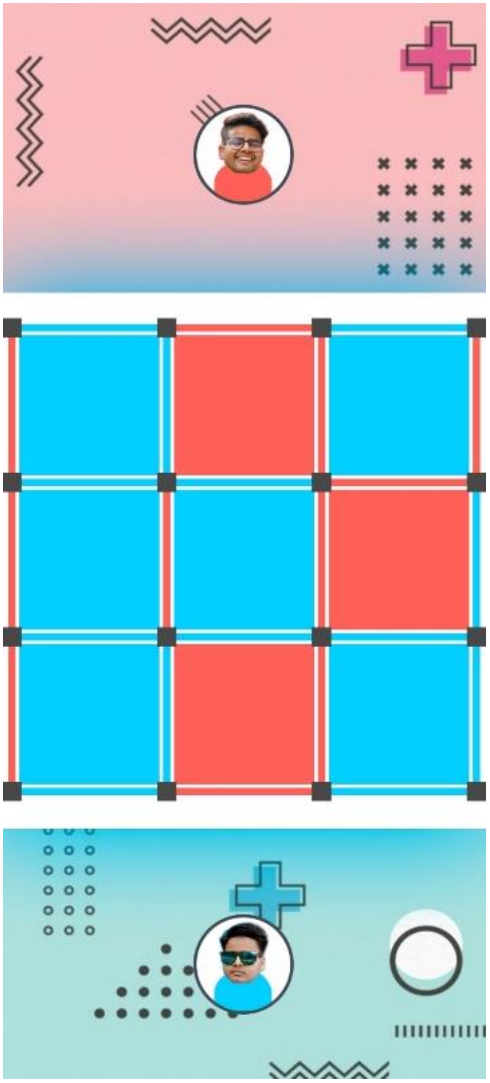

Fig 6.2.4 Grid Completed

- Fig 6.5 presents the game board with all squares on the grid successfully completed.
- The snapshot signifies that the game has reached its conclusion, as every available square has been closed by the players.
- The completion of the grid indicates that all possible moves have been made, and it is now time to determine the final scores and declare a winner.
- The snapshot captures the moment just before transitioning to the results screen, where the final scores and game outcome will be displayed.
- It represents the culmination of the players' efforts throughout the game and sets the stage for the final resolution of the gameplay.

6.3 RESULT SCREEN

Fig 6.3 Results

- Fig 6.6 displays the results screen of the game, presenting the scores of each player along with their respective colors.
- The snapshot provides a visual representation of the final outcome of the game, highlighting the achievements of each player.
- The scores are prominently displayed, allowing players and observers to see the numerical results of their gameplay.
- The colors associated with each player's score further enhance the visual representation, making it easy to distinguish between the players and their corresponding scores.
- The results screen serves as the conclusion of the game, providing a summary of the players' performance and determining the winner based on the scores achieved.

6.4 HELP SCREEN

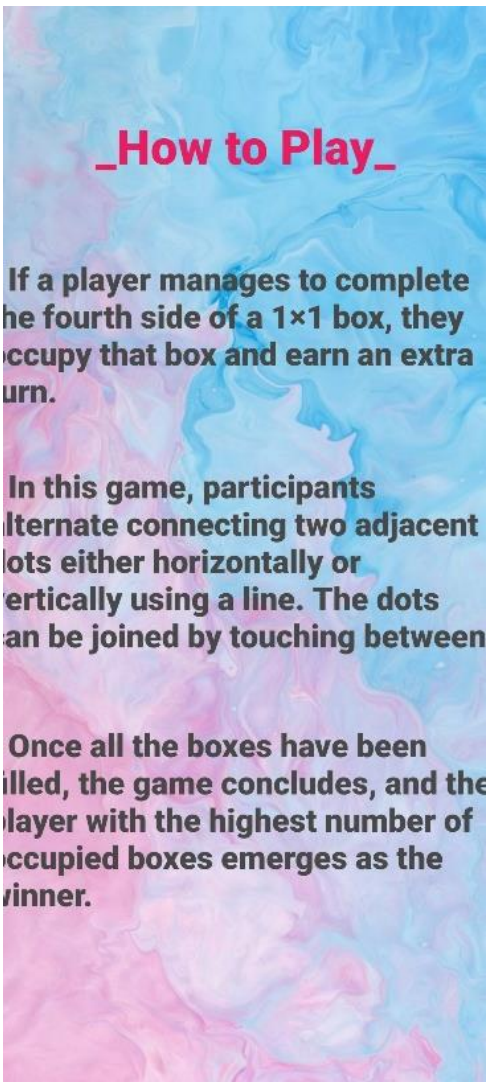

Fig 6.4 Instructions

- Fig 6.7 showcases the instructions screen, which serves as a helpful resource for users to understand and learn how to play the game.
- The snapshot captures the textual content displayed on the instructions screen, providing users with clear guidelines and explanations of the game's rules and objectives.
- It serves as a valuable reference point for both new players who are unfamiliar with the game and experienced players who may need a quick refresher on the rules.

CONCLUSION

This project has successfully implemented a mobile game application using Java and Android Studio. The application allows users to play a grid-based game where they connect dots to form lines and complete squares. The project demonstrates the use of various components and features such as activities, layouts, touch events, canvas drawing, and data flow between different screens. Through the implementation of different classes and utility files, the application provides a seamless gaming experience with intuitive controls and interactive visuals. Future improvements for the mobile game app can include adding new gameplay modes, improving the user interface and graphics, implementing online leaderboards, and providing regular updates for bug fixes and enhancements. Overall, this project showcases the potential of Java and Android development in creating engaging and entertaining mobile applications.

REFERENCES

[1] Erik Hellman, "Android Programming – Pushing the Limits", 1st Edition, Wiley IndiaPvt Ltd, 2014. ISBN-13: 978-8126547197

[2] Dawn Griffiths and David Griffiths, "Head First Android Development", 1st Edition O"Reilly SPD Publishers, 2015. ISBN-13: 978-9352131341

[3] Bill Phillips, Chris Stewart and Kristin Marsicano, "Android Programming: The Big Nerd Ranch Guide", 3rd Edition, Big Nerd Ranch Guides, 2017. ISBN-13: 978- 0134706054

[4] Data from Global Academy of Technology

[5] https://www.w3schools.com/java/

[6] https://en.wikipedia.org/wiki/Android_Studio

[7] https://en.wikipedia.org/wiki/Java_(programming_language)

[8] https://code.visualstudio.com/docs/languages/java

[9] https://developer.android.com/docs