南京信息工程大学

《Java Programming》
Course Design Report

Student Name: Deng Ying

Student ID: 2021830900002/30804538

School: UoR/NUIST

Major: Data Science

Teacher： Wang Jian

June 2024

Application Requirements Analysis

Grade	

1 Application module composition

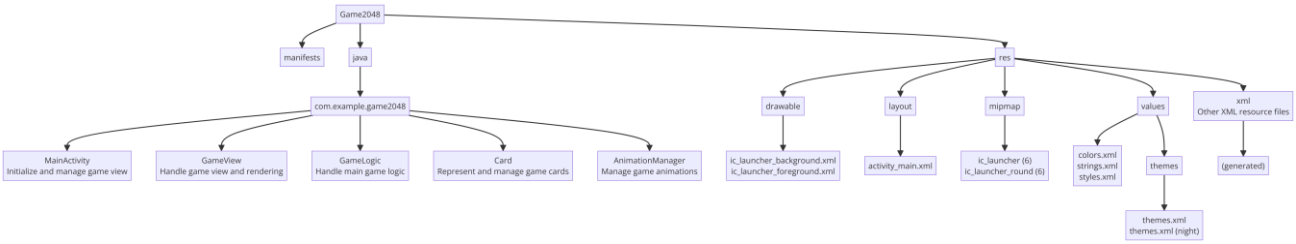

Figure 1: Hierarchical diagram by Mermaidchart

- AnimationManager: Manages the animations within the game, likely handling transitions and movements of game elements.
- Card: Represents the logic and state of a game card, including its value and position.
- GameLogic: Contains the core logic and rules of the 2048 game, managing the game state, moves, and win/loss conditions.
- GameView: Handles the graphical user interface, rendering the game board and interacting with user inputs.
- MainActivity: The main entry point of the application, initializing the game and handling the primary activity lifecycle.
- ic_launcher_background.xml & ic_launcher_foreground.xml: Define the background and foreground images for the application launcher icon.
- activity_main.xml: Defines the layout for the main activity screen.
- ic_launcher & ic_launcher_round: Provide launcher icons in different resolutions for various screen densities.
- colors.xml: Contains color definitions used throughout the application.
- strings.xml: Contains text strings used throughout the application for localization.
- styles.xml: Defines common styles used across the application for consistency.
- themes.xml & themes.xml (night): Define the application's themes, including variations for night mode.

2 Algorithm description for each module

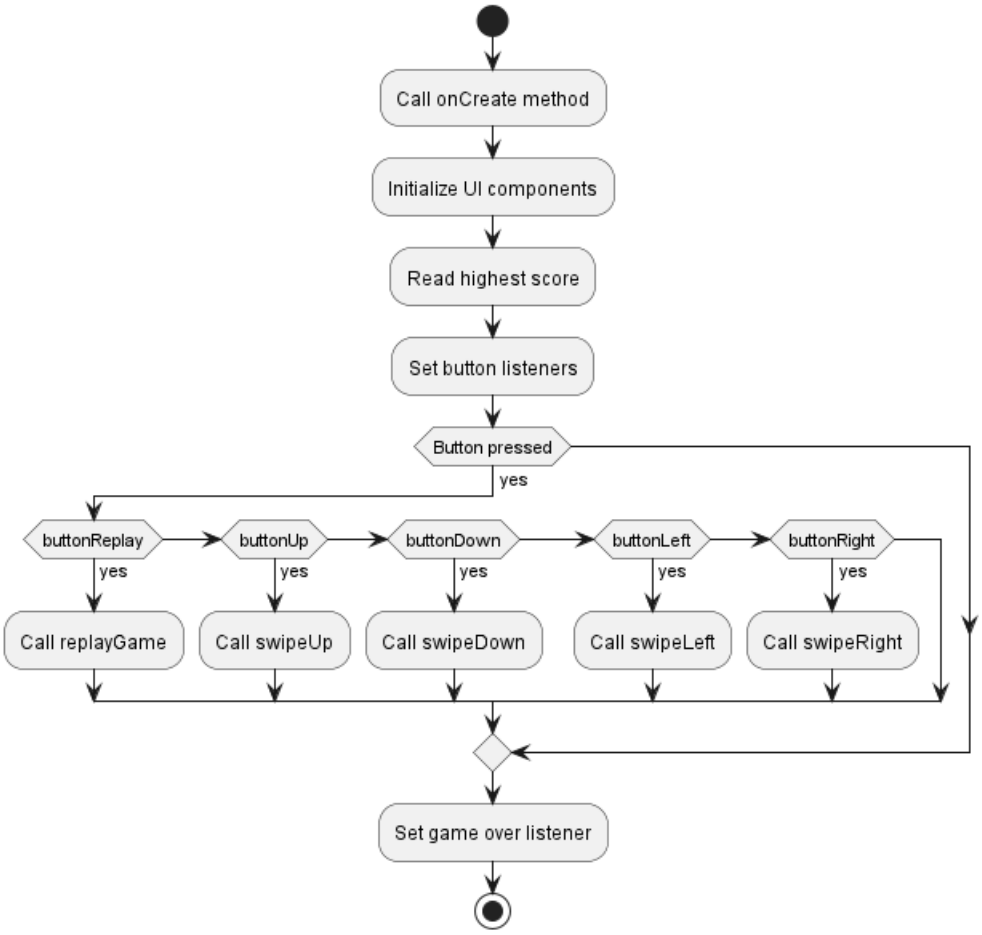

Figure 2: Flowchart of main activity by PlantUML

The flow starts with the `onCreate` method, typically called when the activity is first created in an Android application. After `onCreate` is called, the UI components of the game are initialized. This likely involves setting up the game's layout, buttons, and other visual elements. The algorithm proceeds to read the highest score from a storage mechanism, which could be a local database, shared preferences, or a file. This score is usually displayed on the game's main screen. Next, listeners are set up for the various buttons in the game. These listeners will handle user interactions with the buttons. The flow then enters a loop, continuously checking if any button has been pressed. If a button is pressed, the algorithm determines which button it is. There are five possible buttons to handle: if the replay button is pressed (`buttonReplay`), the `replayGame` function is called; if the up button is pressed (`buttonUp`), the `swipeUp` function is called; if the down button is pressed (`buttonDown`), the `swipeDown` function is called; if the left button is pressed (`buttonLeft`), the `swipeLeft` function is called; if the right button is pressed (`buttonRight`), the `swipeRight` function is called. After handling the button press, the algorithm sets a listener for the game over event. This listener will handle the game over logic, such as updating the score and resetting the game state. The flow then loops back to continuously check for button presses, ensuring the game remains responsive to user inputs.

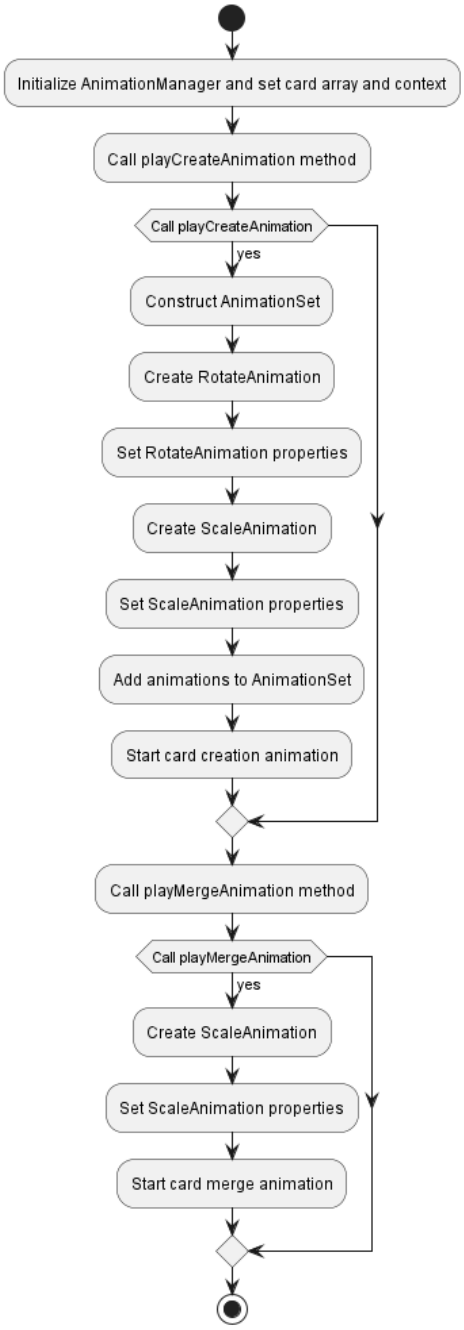

Figure 3:AnimationManager by PlantUML

The flow starts with initializing the `AnimationManager` and setting the card array and context. The process then proceeds to call the `playCreateAnimation` method. Within the `playCreateAnimation` method, an `AnimationSet` is constructed, followed by the creation of a `RotateAnimation`. Properties for the `RotateAnimation` are then set. Afterward, a `ScaleAnimation` is created and its properties are also set. These animations are then added to the `AnimationSet`, and the card creation animation is started. Next, the process moves to call the `playMergeAnimation` method. Within the `playMergeAnimation` method, a `ScaleAnimation` is created, and its properties are set. Finally, the card merge animation is started. This flow ensures that animations for card creation and merging are properly set up and executed in sequence.

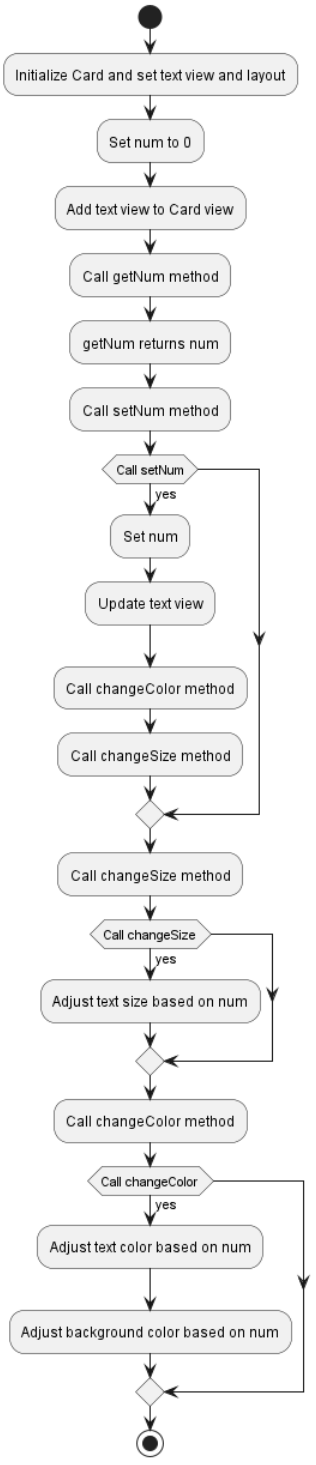

Figure 4: Flowchart of Card by PlantUML

The flow starts with initializing the Card and setting the text view and layout. It then sets num to 0 and adds the text view to the Card view. Next, it calls the getNum method, which returns num. The process then calls the setNum method. If setNum is called, it sets num, updates the text view, and calls the changeColor and changeSize methods.

When the changeSize method is called, it adjusts the text size based on num. Then the changeColor method is called again, adjusting the text color and background color based on num. This ensures that the text view and the Card view update correctly in response to changes in num.

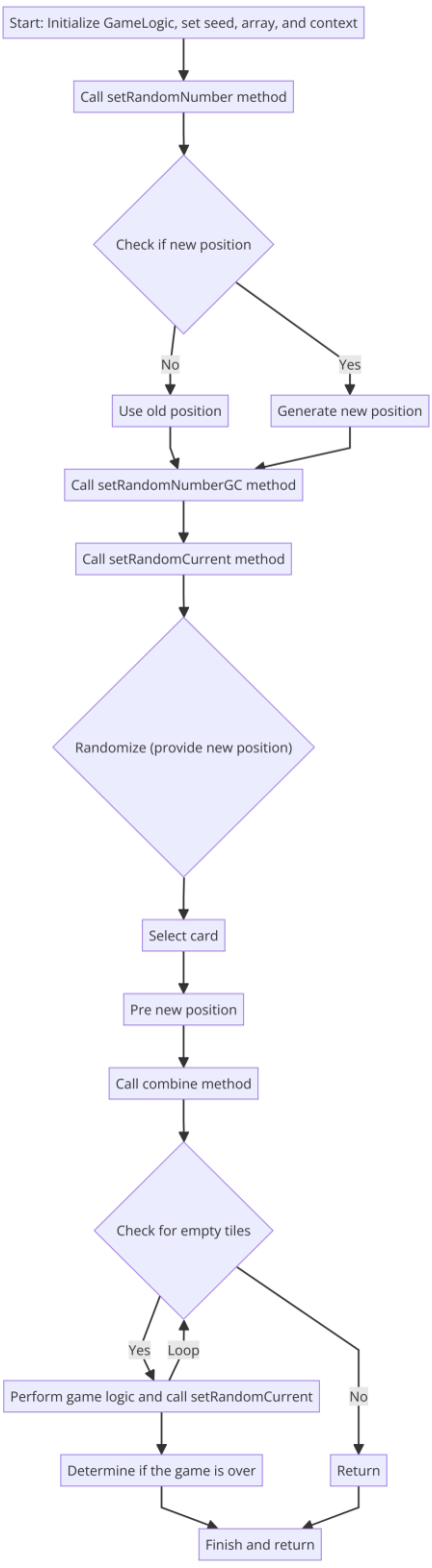

Figure 5: Flowchart of Game Logic by Mermaidchart

The flow starts with initializing the GameLogic, setting the card array and context. It then calls the readyToPlay method, which clears all cards and calls randomCreateCard. The randomCreateCard method randomly generates a card position, sets the card view, and plays the create animation. Next, the checkGame method is called to check for empty tiles or mergeable tiles; if true, it returns a visual update.

Subsequently, the flow calls the swipeUp method to perform the swipe logic, and if the swipe is successful, it calls randomCreateCard. The same process is repeated for the swipeDown, swipeLeft, and swipeRight methods, each performing the respective swipe logic and creating a random card if the swipe is successful. Finally, the checkGameOver method determines if the game is over and returns a visual update.

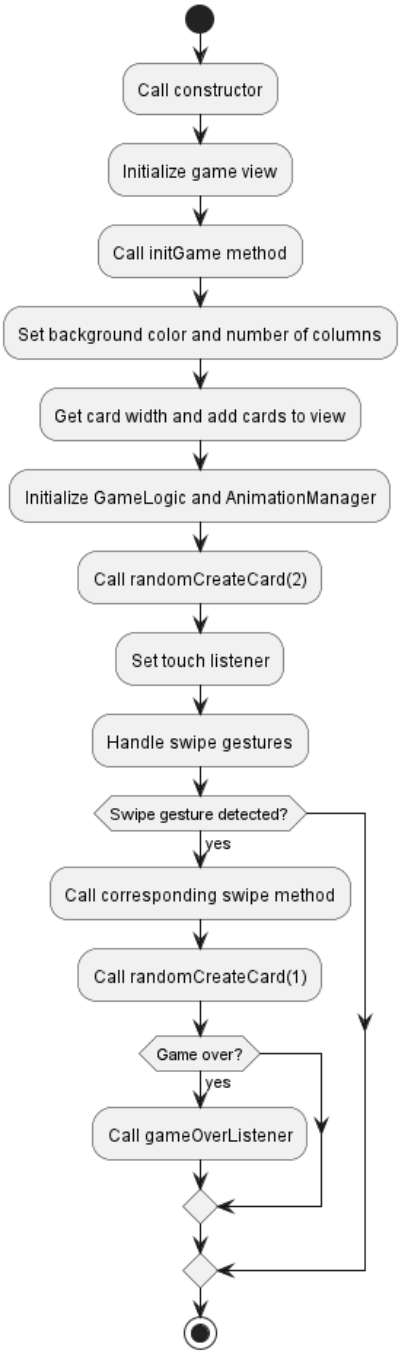

Figure 6: Chartflow of Gameview by PlantUML

The flow starts with calling the constructor to initialize the game. The game view is then initialized, and the initGame method is called. Within this method, the background color and the number of columns are set. The card width is obtained, and cards are added to the view. Next, the GameLogic and AnimationManager are initialized. Two random cards are created by calling randomCreateCard(2). A touch listener is then set to handle swipe gestures. If a swipe gesture is

detected, the corresponding swipe method is called, and a new random card is created with randomCreateCard(1). Finally, the game checks if it is over, and if so, the gameOverListener is called to handle the game-over scenario.

3 Description of the classes that make up the system

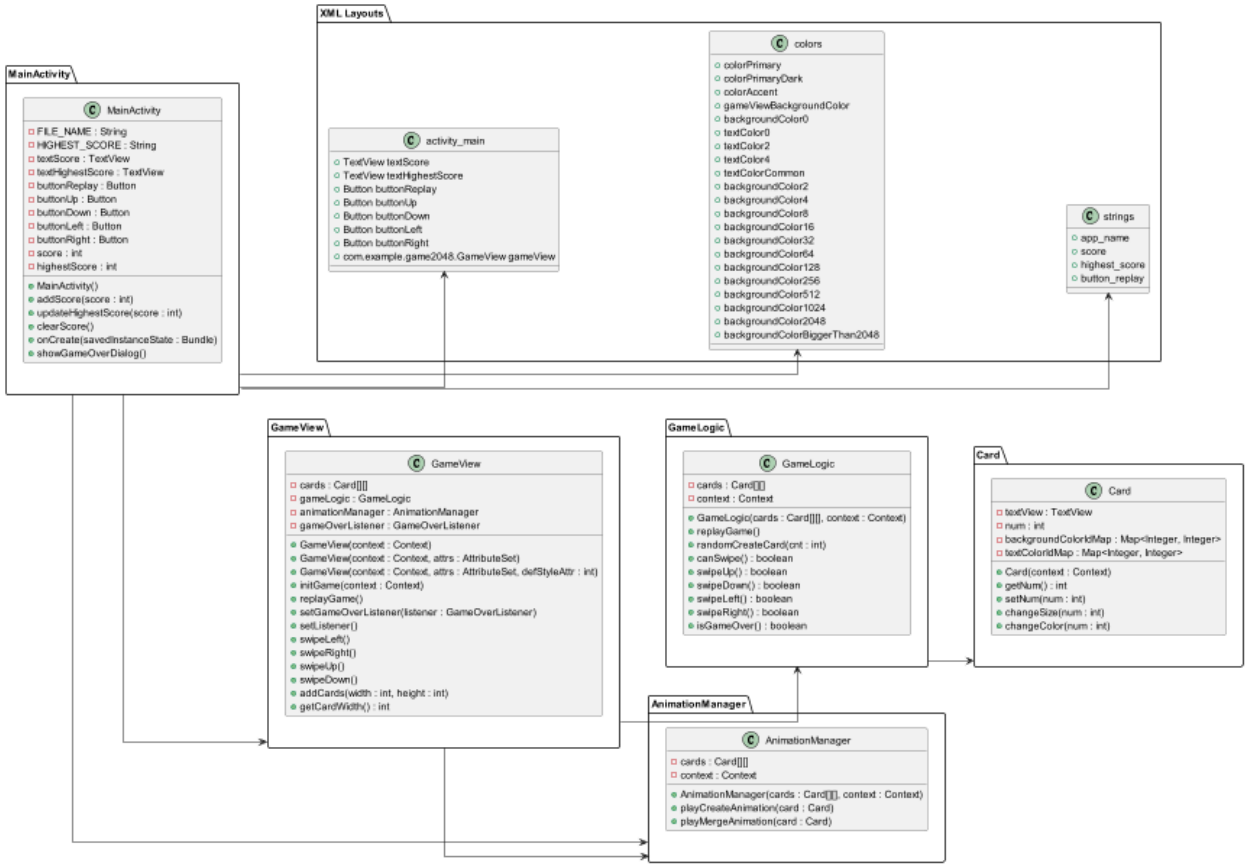

Figure 7：Class diagram of this project by draw.io

- MainActivity

The MainActivity initializes the main activity, sets up UI components, reads the highest score from shared preferences, and sets listeners for game interaction buttons. It increases the current score and updates the score display. If the current score exceeds the highest score, it updates the highest score. It also resets the current score to zero and updates the score display. Additionally, it displays a dialog when the game ends, offering options to restart the game or exit.

- GameView

The GameView is a custom view class used to display and manage the game grid, handle user swipe actions, and interact with GameLogic and AnimationManager. It initializes the game by setting the background color, column count, and adding card views. It initializes the GameLogic and AnimationManager modules and starts the game by creating the first two cards. The GameView also resets the game by clearing the score and restarting the game logic. It sets a listener for game over events and a touch listener to handle swipe gestures for game interaction. Furthermore, it handles respective swipe actions and updates the game state accordingly. It adds

card views to the game grid and calculates and returns the width of the cards based on the device's screen width.

- GameLogic

The GameLogic module manages the core game logic, including the movement and merging of cards, and checking the game state. It resets the game by clearing all cards and creating two new cards. It randomly creates a specified number of new cards in empty spots on the grid. The GameLogic also checks if there are any possible moves left, either empty spots or adjacent cards with the same number. It handles respective swipe actions, merges cards where possible, and creates a new card if the swipe was successful. Finally, it determines if the game is over by checking for any possible moves.

- Card

The Card class represents each card in the game, managing the card's number and appearance, including color and size. It returns the number on the card and sets the number on the card, updating the displayed number. The Card class changes the text and background color based on the number and adjusts the text size based on the number on the card.

- AnimationManager

The AnimationManager manages and performs animations for the cards, including animations for card creation and merging. It plays animations for newly created cards, including rotation and scaling effects, and plays scaling animations for cards that are merged during a move.

- XML Layouts

- activity_main.xml: Defines the layout of the main activity, including text views for displaying the score and highest score, buttons for restarting the game and controlling directions (up, down, left, right). It also includes a custom GameView for displaying the game grid.
- colors.xml: Defines color resources used in the application, including the game background color, card background colors, and text colors.
- strings.xml: Defines string resources used in the application, including the application name, score display text, highest score display text, and restart button text.

- Key Points Summary

Dependencies:

- MainActivity has dependencies on GameView, AnimationManager, activity_main, colors, and strings.
- GameView depends on GameLogic and AnimationManager.
- GameLogic depends on Card.

Operation and testing of each module

4 Test data

System Data Operation Process

- Step 1: Initialization
  Game Setup: When the game starts, the system automatically initializes the game state, including creating a 4x4 grid with two tiles and displaying the highest score.
- Step 2: User Input Handling
  Direction Buttons: The game interface has four direction buttons (up, down, left, right). Users can control the tile movements by clicking these buttons.
- Step 3: Game Logic Execution
  Tile Movement and Merging: Based on the user's direction input, tiles move in the corresponding direction and merge if they encounter tiles with the same value, creating new tiles.
  New Tile Generation: After each move, a new tile with a value of 2 or 4 is randomly generated in an empty spot on the grid.
- Step 4: Game State Update
  Score Update: After each tile merge, the system updates the current score and displays it on the screen.
  Highest Score Update: If the current score exceeds the highest score, the system updates and displays the new highest score.
  Game Over Detection: After each move, the system checks if any more moves are possible. If not, a game-over dialog is displayed.
- Step 5: Game Over Handling
  Game Over Dialog: If the game ends, a dialog box is shown, offering the options to "Restart" or "Exit" the game, allowing the user to choose to start a new game or exit.

Manual Testing Steps

To initialize the game interface, run MainActivity.java. The system will automatically display a grid with two tiles, the current score, and the highest score. For input handling, use the on-screen up, down, left, and right buttons to test tile movement in each direction. During game logic testing, verify that tiles merge correctly when moved and check if a new tile is generated after each move. For state update testing, ensure that the current score is updated correctly after tile merges and verify that the highest score is updated when the current score exceeds it. Confirm that the "Game Over" dialog is displayed correctly when no more merges are possible, and test the functionality of the "Restart" and "Exit" options in the "Game Over" dialog to ensure they work correctly.

5 Screenshot of the running interface

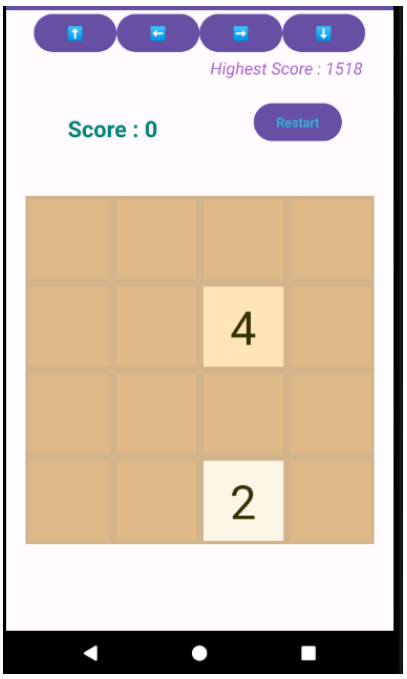

Figure 8：Initial interface

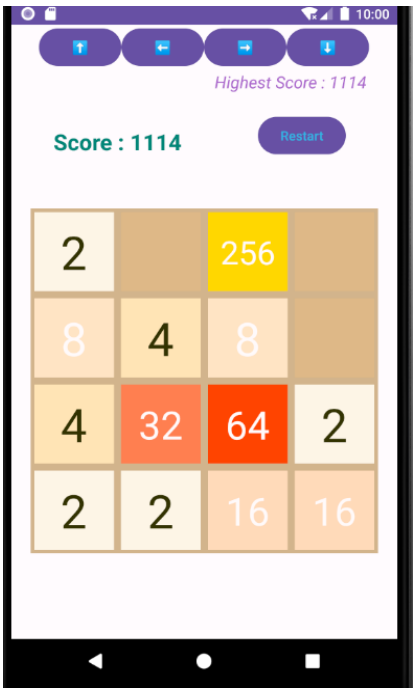

Figure 9: Start the program

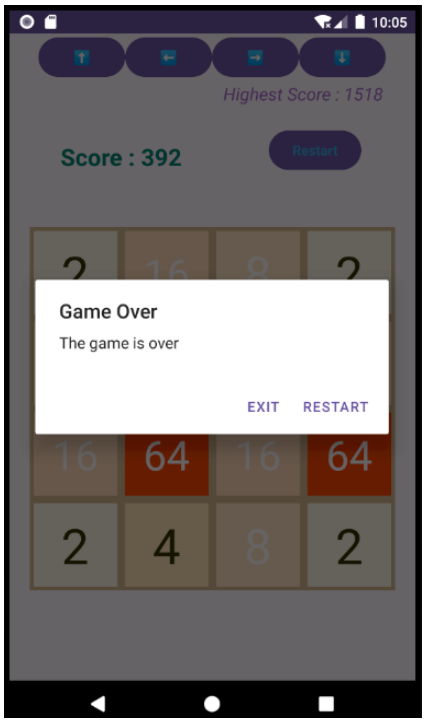

Figure 10: Game over

Figure 11: Choose “EXIT”

6 Evaluation

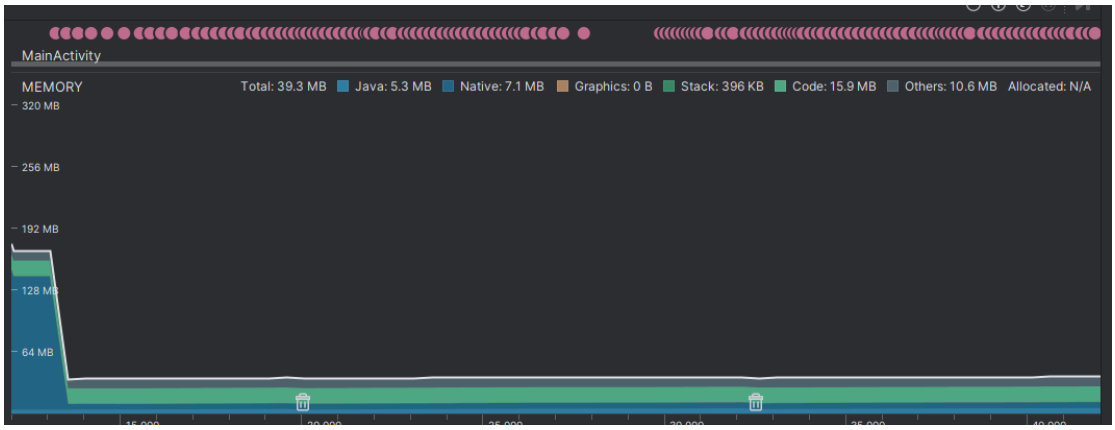

Figure 12: Android Profiler

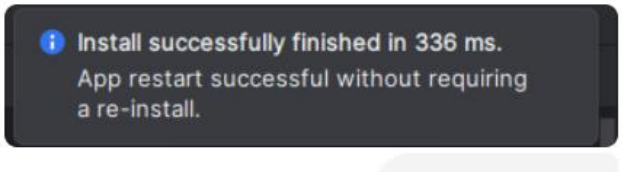

Figure 13: Run the MainActivity

The evaluation of the system's operational efficiency is based on the performance metrics observed during the execution of the application using Android Studio Profiler. The key metrics include response time, memory usage, and processing speed during different operations such as moving tiles in different directions.

7 Performance Metrics Analysis

- **Response Time:**
  The response time for user inputs (up, down, left, right movements) was measured by observing the timestamps before and after the processing of each move. The consistent distribution of the pink dots indicates frequent user interactions and system responsiveness to these inputs.
- **Memory** Usage**:**
  The memory usage was tracked throughout the application's lifecycle. The profiler captures the memory usage details as follows:
  - **Total Memory:** 39.3 MB
  - **Java Memory:** 5.3 MB
  - **Native Memory:** 7.1 MB
  - **Code Memory:** 15.9 MB
  - **Other Memory:** 10.6 MB
- **Memory Peaks and Garbage Collection:**
  There was a noticeable memory peak around 192 MB at the start, which then decreased to a stable usage of around 64 MB. This indicates that the application initially allocated a significant amount of memory, but it was able to release it, likely due to garbage collection.

- **CPU Utilization:**
  The continuous distribution of pink dots representing user inputs shows that the CPU was actively processing these inputs without significant delay, implying efficient CPU utilization.
- **Installation and Restart Efficiency:**
  The application installation completed successfully in 336 ms, and the app was able to restart without requiring a re-install. This quick installation and restart time indicates efficient handling of the application's lifecycle events.

8 Appendix

MainActivity.java

```
package com.example.game2048;

import androidx.appcompat.app.AppCompatActivity;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private final String FILE_NAME = "MyData";
    private final String HIGHEST_SCORE = "highest_score";

    private TextView textScore;
    private TextView textHighestScore;
    private Button buttonReplay;
    private Button buttonUp, buttonDown, buttonLeft, buttonRight;

    private int score = 0;
    private int highestScore = 0;

    public static MainActivity mainActivity;

    public MainActivity() {
        mainActivity = this;
    }

    public void addScore(int score) {
        this.score += score;
        textScore.setText("Score : " + this.score);
```

```
        ///update the highest score
        updateHighestScore(this.score);
    }

    private void updateHighestScore(int score) {
        if (score > highestScore) {
            highestScore = score;
            textHighestScore.setText("Highest Score : " + score);
            //store the highest score
            SharedPreferences shp = getSharedPreferences(FILE_NAME, MODE_PRIVATE);
            SharedPreferences.Editor editor = shp.edit();

            editor.putInt(HIGHEST_SCORE, highestScore);
            editor.apply();
        }
    }

    public void clearScore() {
        score = 0;
        textScore.setText("Score : " + 0);
    }

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

        textScore = findViewById(R.id.textScore);
        textHighestScore = findViewById(R.id.textHighestScore);
        buttonReplay = findViewById(R.id.buttonReplay);
        buttonUp = findViewById(R.id.buttonUp);
        buttonDown = findViewById(R.id.buttonDown);
        buttonLeft = findViewById(R.id.buttonLeft);
        buttonRight = findViewById(R.id.buttonRight);

        // get the highest score
        SharedPreferences shp = getSharedPreferences(FILE_NAME, MODE_PRIVATE);
        highestScore = shp.getInt(HIGHEST_SCORE, 0);
        textHighestScore.setText("Highest Score : " + highestScore);

        buttonReplay.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                GameView.gameView.replayGame();
```

```
                GameView.gameView.invalidate();
            }
        });

        buttonUp.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                GameView.gameView.swipeUp();
                GameView.gameView.invalidate()
            }
        });

        buttonDown.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                GameView.gameView.swipeDown();
                GameView.gameView.invalidate();                    }
        });

        buttonLeft.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                GameView.gameView.swipeLeft();
                GameView.gameView.invalidate()                    }
        });

        buttonRight.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                GameView.gameView.swipeRight();
                GameView.gameView.invalidate();                    }
        });

        // 设置游戏结束监听器
        GameView.gameView.setGameOverListener(new GameView.GameOverListener() {
            @Override
            public void onGameOver() {
                showGameOverDialog();
            }
        });
    }

    public void showGameOverDialog() {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

```
            builder.setTitle("Game Over");
            builder.setMessage("The game is over");
            builder.setPositiveButton("Restart", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    GameView.gameView.replayGame();
                }
            });
            builder.setNegativeButton("Exit", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    finish();
                }
            });
            builder.setCancelable(false);
            builder.show();
        }
    }
```

GameView.java

```
package com.example.game2048;

import android.content.Context;
import android.util.AttributeSet;
import android.util.DisplayMetrics;
import android.view.MotionEvent;
import android.view.View;
import android.widget.GridLayout;

public class GameView extends GridLayout {

    public static GameView gameView;

    private Card[][] cards = new Card[4][4];
    private GameLogic gameLogic;
    private AnimationManager animationManager;
    private GameOverListener gameOverListener;

    public GameView(Context context) {
        super(context);
        gameView = this;
        initGame(context);
    }

    public GameView(Context context, AttributeSet attrs) {
```

```
        super(context, attrs);
        gameView = this;
        initGame(context);
    }

    public GameView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        gameView = this;
        initGame(context);
    }

    public void initGame(Context context) {
        this.setBackgroundColor(getResources().getColor(R.color.gameViewBackgroundColor));
        setColumnCount(4);

        int cardWidth = getCardWidth();
        addCards(cardWidth, cardWidth);

        gameLogic = new GameLogic(cards, context);
        animationManager = new AnimationManager(cards, context);

        gameLogic.randomCreateCard(2);

        setListener();
    }

    public void replayGame() {
        MainActivity.mainActivity.clearScore();
        gameLogic.replayGame();
    }

    public void setGameOverListener(GameOverListener listener) {
        this.gameOverListener = listener;
    }

    private void setListener() {
        setOnTouchListener(new OnTouchListener() {
            private float startX, startY, endX, endY;

            @Override
            public boolean onTouch(View v, MotionEvent event) {
                switch (event.getAction()) {
                    case MotionEvent.ACTION_DOWN:
                        startX = event.getX();
```

```
                startY = event.getY();
                break;

            case MotionEvent.ACTION_UP:
                endX = event.getX();
                endY = event.getY();

                boolean swiped = false;

                if (Math.abs(endX - startX) > Math.abs(endY - startY)) {
                    if (endX - startX > 10) {
                        if (gameLogic.swipeRight()) {
                            swiped = true;
                        }
                    } else if (endX - startX < -10) {
                        if (gameLogic.swipeLeft()) {
                            swiped = true;
                        }
                    }
                } else {
                    if (endY - startY < -10) {
                        if (gameLogic.swipeUp()) {
                            swiped = true;
                        }
                    } else if (endY - startY > 10) {
                        if (gameLogic.swipeDown()) {
                            swiped = true;
                        }
                    }
                }

                if (swiped) {
                    gameLogic.randomCreateCard(1);
                    if (gameLogic.isGameOver() && gameOverListener != null) {
                        gameOverListener.onGameOver();
                    }
                }
                break;
        }
        return true;
    }
});
}
```

```
public void swipeLeft() {
    gameLogic.swipeLeft();
    if (gameLogic.isGameOver() && gameOverListener != null) {
        gameOverListener.onGameOver();
    }
}

public void swipeRight() {
    gameLogic.swipeRight();
    if (gameLogic.isGameOver() && gameOverListener != null) {
        gameOverListener.onGameOver();
    }
}

public void swipeUp() {
    gameLogic.swipeUp();
    if (gameLogic.isGameOver() && gameOverListener != null) {
        gameOverListener.onGameOver();
    }
}

public void swipeDown() {
    gameLogic.swipeDown();
    if (gameLogic.isGameOver() && gameOverListener != null) {
        gameOverListener.onGameOver();
    }
}

private void addCards(int width, int height) {
    Card c;
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            c = new Card(getContext());
            addView(c, width, height);
            cards[i][j] = c;
        }
    }
}

private int getCardWidth() {
    DisplayMetrics displayMetrics = getResources().getDisplayMetrics();
    return (int) ((displayMetrics.widthPixels * 0.9f) / 4);
}
```

```
    public interface GameOverListener {
        void onGameOver();
    }
}
```

GameLogic.java

```
package com.example.game2048;

import android.content.Context;
import java.util.Random;

public class GameLogic {

    private Card[][] cards;
    private Context context;

    public GameLogic(Card[][] cards, Context context) {
        this.cards = cards;
        this.context = context;
    }

    public void replayGame() {
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                cards[i][j].setNum(0);
            }
        }
        randomCreateCard(2);
    }

    public void randomCreateCard(int cnt) {
        Random random = new Random();
        for (int i = 0; i < cnt; i++) {
            int r = random.nextInt(4);
            int c = random.nextInt(4);

            while (cards[r][c].getNum() != 0) {
                r = random.nextInt(4);
                c = random.nextInt(4);
            }

            int rand = random.nextInt(10);
            if (rand >= 2) {
                rand = 2;
            } else {
```

```
                rand = 4;
            }

            cards[r][c].setNum(rand);
            AnimationManager.playCreateAnimation(cards[r][c]);
        }
    }

    public boolean canSwipe() {
        // Check if there are any empty tiles
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                if (cards[i][j].getNum() == 0) {
                    return true;
                }
            }
        }

        // Check if there are any tiles that can be merged
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                if ((i < 3 && cards[i][j].getNum() == cards[i + 1][j].getNum()) ||
                        (j < 3 && cards[i][j].getNum() == cards[i][j + 1].getNum())) {
                    return true;
                }
            }
        }

        // If there are no empty tiles and no tiles can be merged, the game is over
        return false;
    }

    public boolean swipeUp() {
        boolean flag = false;
        for (int j = 0; j < 4; ++j) {
            int ind = 0;
            for (int i = 1; i < 4; ++i) {
                if (cards[i][j].getNum() != 0) {
                    for (int ii = i - 1; ii >= ind; --ii) {
                        if (cards[ii][j].getNum() == 0) {
                            cards[ii][j].setNum(cards[i][j].getNum());
                            cards[i][j].setNum(0);
                            i--;
                            flag = true;
```

```
                        } else if (cards[ii][j].getNum() == cards[i][j].getNum()) {
                            cards[ii][j].setNum((cards[i][j].getNum() * 2));
                            cards[i][j].setNum(0);
                            flag = true;
                            ind = ii + 1;
                            MainActivity.mainActivity.addScore(cards[ii][j].getNum() / 2);
                            AnimationManager.playMergeAnimation(cards[ii][j]);
                            break;
                        } else {
                            break;
                        }
                    }
                }
            }
        }
        if (flag) {
            randomCreateCard(1);
        }
        return flag;
    }

    public boolean swipeDown() {
        boolean flag = false;
        for (int j = 0; j < 4; ++j) {
            int ind = 4;
            for (int i = 2; i >= 0; --i) {
                if (cards[i][j].getNum() != 0) {
                    for (int ii = i + 1; ii < ind; ++ii) {
                        if (cards[ii][j].getNum() == 0) {
                            cards[ii][j].setNum(cards[i][j].getNum());
                            cards[i][j].setNum(0);
                            flag = true;
                            i++;
                        } else if (cards[ii][j].getNum() == cards[i][j].getNum()) {
                            cards[ii][j].setNum((cards[i][j].getNum() * 2));
                            cards[i][j].setNum(0);
                            flag = true;
                            ind = ii;
                            MainActivity.mainActivity.addScore(cards[ii][j].getNum() / 2);
                            AnimationManager.playMergeAnimation(cards[ii][j]);
                            break;
                        } else {
                            break;
                        }
```

```
                    }
                }
            }
        }
        if (flag) {
            randomCreateCard(1);
        }
        return flag;
    }

    public boolean swipeLeft() {
        boolean flag = false;
        for (int i = 0; i < 4; ++i) {
            int ind = 0;
            for (int j = 1; j < 4; ++j) {
                if (cards[i][j].getNum() != 0) {
                    for (int jj = j - 1; jj >= ind; --jj) {
                        if (cards[i][jj].getNum() == 0) {
                            cards[i][jj].setNum(cards[i][j].getNum());
                            cards[i][j].setNum(0);
                            flag = true;
                            j--;
                        } else if (cards[i][jj].getNum() == cards[i][j].getNum()) {
                            cards[i][jj].setNum((cards[i][j].getNum() * 2));
                            cards[i][j].setNum(0);
                            flag = true;
                            ind = jj + 1;
                            MainActivity.mainActivity.addScore(cards[i][jj].getNum() / 2);
                            AnimationManager.playMergeAnimation(cards[i][jj]);
                            break;
                        } else {
                            break;
                        }
                    }
                }
            }
        }
        if (flag) {
            randomCreateCard(1);
        }
        return flag;
    }

    public boolean swipeRight() {
```

```
        boolean flag = false;
        for (int i = 0; i < 4; ++i) {
            int ind = 4;
            for (int j = 2; j >= 0; --j) {
                if (cards[i][j].getNum() != 0) {
                    for (int jj = j + 1; jj < ind; ++jj) {
                        if (cards[i][jj].getNum() == 0) {
                            cards[i][jj].setNum(cards[i][j].getNum());
                            cards[i][j].setNum(0);
                            flag = true;
                            j++;
                        } else if (cards[i][jj].getNum() == cards[i][j].getNum()) {
                            cards[i][jj].setNum((cards[i][j].getNum() * 2));
                            cards[i][j].setNum(0);
                            flag = true;
                            ind = jj;
                            MainActivity.mainActivity.addScore(cards[i][jj].getNum() / 2);
                            AnimationManager.playMergeAnimation(cards[i][jj]);
                            break;
                        } else {
                            break;
                        }
                    }
                }
            }
        }
        if (flag) {
            randomCreateCard(1);
        }
        return flag;
    }

    public boolean isGameOver() {
        // Check if there are any empty tiles
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                if (cards[i][j].getNum() == 0) {
                    return false;
                }
            }
        }

        // Check if there are any tiles that can be merged
        for (int i = 0; i < 4; ++i) {
```

```
                for (int j = 0; j < 4; ++j) {
                    if ((i < 3 && cards[i][j].getNum() == cards[i + 1][j].getNum()) ||
                            (j < 3 && cards[i][j].getNum() == cards[i][j + 1].getNum())) {
                        return false;
                    }
                }
            }

            // If there are no empty tiles and no tiles can be merged, the game is over
            return true;
        }
}
```

Card.java

```
package com.example.game2048;

import android.content.Context;
import android.view.Gravity;
import android.widget.FrameLayout;
import android.widget.TextView;

import androidx.annotation.NonNull;

import java.util.HashMap;
import java.util.Map;

public class Card extends FrameLayout {

    TextView textView;

    private int num;

    static Map<Integer, Integer> backgroundColorIdMap = new HashMap<>();
    static Map<Integer, Integer> textColorIdMap = new HashMap<>();

    static {
        textColorIdMap.put(0, R.color.textColor0);
        textColorIdMap.put(2, R.color.textColor2);
        textColorIdMap.put(4, R.color.textColor4);

        backgroundColorIdMap.put(0, R.color.backgroundColor0);
        backgroundColorIdMap.put(2, R.color.backgroundColor2);
        backgroundColorIdMap.put(4, R.color.backgroundColor4);
        backgroundColorIdMap.put(8, R.color.backgroundColor8);
        backgroundColorIdMap.put(16, R.color.backgroundColor16);
```

```
        backgroundColorIdMap.put(32, R.color.backgroundColor32);
        backgroundColorIdMap.put(64, R.color.backgroundColor64);
        backgroundColorIdMap.put(128, R.color.backgroundColor128);
        backgroundColorIdMap.put(256, R.color.backgroundColor256);
        backgroundColorIdMap.put(512, R.color.backgroundColor512);
        backgroundColorIdMap.put(1024, R.color.backgroundColor1024);
    }

    public Card(@NonNull Context context) {
        super(context);

        textView = new TextView(context);
        textView.setGravity(Gravity.CENTER);
        textView.setText(getNum() + "");
        textView.setTextSize(50);

        setNum(0);

        LayoutParams lp = new LayoutParams(-1, -1);
        lp.setMargins(10, 10, 10, 10);

        addView(textView, lp);
    }

    public int getNum() {
        return num;
    }

    public void setNum(int num) {
        this.num = num;
        textView.setText(num + "");
        changeColor(num);
        changeSize(num);
    }

    private void changeSize(int num){
        if(num >= 1024){
            textView.setTextSize(25);
        } else if(num >= 128){
            textView.setTextSize(35);
        } else if(num >= 16){
            textView.setTextSize(42);
        } else{
            textView.setTextSize(50);
```

```
            }
        }

        private void changeColor(int num){
            if(num >= 8){
                textView.setTextColor(getResources().getColor(R.color.textColorCommon));
            } else{
                textView.setTextColor(getResources().getColor(textColorIdMap.get(num)));
            }
            if(num >= 2048){

textView.setBackgroundColor(getResources().getColor(R.color.backgroundColorBiggerThan2048));
            } else{
                textView.setBackgroundColor(getResources().getColor(backgroundColorIdMap.get(num)));
            }
        }
}
```

AnimationManager.java

```
package com.example.game2048;

import android.content.Context;
import android.view.animation.Animation;
import android.view.animation.AnimationSet;
import android.view.animation.LinearInterpolator;
import android.view.animation.RotateAnimation;
import android.view.animation.ScaleAnimation;

public class AnimationManager {

    private static Card[][] cards;
    private static Context context;

    public AnimationManager(Card[][] cards, Context context) {
        this.cards = cards;
        this.context = context;
    }

    public static void playCreateAnimation(Card card) {
        AnimationSet animationSet = new AnimationSet(true);

        RotateAnimation anim = new RotateAnimation(0, 360, RotateAnimation.RELATIVE_TO_SELF,
0.5f, RotateAnimation.RELATIVE_TO_SELF, 0.5f);
        anim.setDuration(250);
        anim.setRepeatCount(0);
```

```
        anim.setInterpolator(new LinearInterpolator());

        ScaleAnimation anim2 = new ScaleAnimation(0, 1, 0, 1, Animation.RELATIVE_TO_SELF, 0.5f,
Animation.RELATIVE_TO_SELF, 0.5f);
        anim2.setDuration(250);
        anim2.setRepeatCount(0);

        animationSet.addAnimation(anim);
        animationSet.addAnimation(anim2);

        card.startAnimation(animationSet);
    }

    public static void playMergeAnimation(Card card) {
        ScaleAnimation anim = new ScaleAnimation(1, 1.2f, 1, 1.2f, Animation.RELATIVE_TO_SELF,
0.5f, Animation.RELATIVE_TO_SELF, 0.5f);
        anim.setDuration(150);
        anim.setRepeatCount(0);

        anim.setRepeatMode(Animation.REVERSE);

        card.startAnimation(anim);
    }
}
```

colors.xml

```
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="gameViewBackgroundColor">#D2B48C</color>
    <color name="backgroundColor0">#DEB887</color>
    <color name="textColor0">#00000000</color>
    <color name="textColor2">#333300</color>
    <color name="textColor4">#333300</color>
    <color name="textColorCommon">#FFFAFA</color>
    <color name="backgroundColor2">#FDF5E6</color>
    <color name="backgroundColor4">#FFE4B5</color>
    <color name="backgroundColor8">#FFE4C4</color>
    <color name="backgroundColor16">#FFDAB9</color>
    <color name="backgroundColor32">#FF7F50</color>
    <color name="backgroundColor64">#FF4500</color>
    <color name="backgroundColor128">#FF8C00</color>
    <color name="backgroundColor256">#FFD700</color>
    <color name="backgroundColor512">#9ACD32</color>
```

```
    <color name="backgroundColor1024">#483D8B</color>
    <color name="backgroundColor2048">#4B0082</color>
    <color name="backgroundColorBiggerThan2048">#000000</color>
</resources>
```

Strings.xml

```
<resources>
    <string name="app_name">Game2048</string>
    <string name="score">Score : 0</string>
    <string name="highest_score">Highest Score : 0</string>
    <string name="button_replay">Restart</string>
</resources>
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textScore"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/score"
        android:textColor="@color/colorPrimary"
        android:textSize="24sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/guideline4"
        app:layout_constraintEnd_toStartOf="@+id/guideline2"
        app:layout_constraintStart_toStartOf="@+id/guideline6"
        app:layout_constraintTop_toTopOf="@+id/guideline3" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_percent="0.5" />

    <Button
        android:id="@+id/buttonReplay"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="@string/button_replay"
        android:textColor="@android:color/holo_blue_light"
        app:layout_constraintBottom_toTopOf="@+id/guideline4"
        app:layout_constraintEnd_toStartOf="@+id/guideline7"
        app:layout_constraintHorizontal_bias="0.628"
        app:layout_constraintStart_toStartOf="@+id/guideline2"
        app:layout_constraintTop_toTopOf="@+id/guideline3"
        app:layout_constraintVertical_bias="0.351" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.12" />

    <TextView
        android:id="@+id/textHighestScore"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/highest_score"
        android:textColor="@android:color/holo_purple"
        android:textSize="18sp"
        android:textStyle="italic"
        app:layout_constraintBottom_toTopOf="@+id/guideline3"
        app:layout_constraintEnd_toStartOf="@+id/guideline7"
        app:layout_constraintHorizontal_bias="0.49"
        app:layout_constraintStart_toStartOf="@+id/guideline2"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.89" />

    <com.example.game2048.GameView
        android:id="@+id/gameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@+id/guideline5"
        app:layout_constraintEnd_toStartOf="@+id/guideline7"
        app:layout_constraintStart_toStartOf="@+id/guideline6"
        app:layout_constraintTop_toTopOf="@+id/guideline4" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline4"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.26" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.9" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_percent="0.05109489" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_percent="0.95" />

    <!-- Add buttons to simulate sliding -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:gravity="center_horizontal">

        <Button
            android:id="@+id/buttonUp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="⬆"/>

        <Button
            android:id="@+id/buttonLeft"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="⬅"/>
```

```
        <Button
            android:id="@+id/buttonRight"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="➡"/>

        <Button
            android:id="@+id/buttonDown"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="⬇"/>

    </LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

AndroidMainfest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Game2048"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```