DOCUMENTATION ON TICTACTOE MINI-PROJECT

Aim: The aim of the project is to understand the development of a Tic-Tac-Toe for Android devices. The project involves creating a mobile app that allows players to engage in the classic two-player Tic-Tac-Toe game as well as play against an AI opponent with varying levels of difficulty.

- **The main features of this project are:**

1. **Two-Player Gameplay**: The app allows two players to take turns playing Tic-Tac-Toe on the same device.

2. **Single-Player Mode with AI:** The app provides an option for a single player to play against an AI opponent. The AI is implemented with different levels of difficulty: easy and hard.

3. **User Interface:** The project includes well-designed user interfaces using XML layouts, with buttons and animations to enhance the user experience.

4. **Player Name Customization:** The players' names can be customized, and their scores are tracked as they win rounds and games.

5. **Pause and Reset:** The game includes functionality to pause the game, reset the current round, start a new game, and view credits.

6. **Animations:** The app utilizes Lottie animations to provide visual feedback to players, including win animations and celebratory effects.

7. **State Management:** The game retains its state during device rotation or other interruptions using ‘onSaveInstanceState’ and ‘onRestoreInstanceState’.

8. **Dialogs:** Custom dialogs are used for pause menus, reset, and displaying credits.

9. **Intents and Activities:** The app uses intents and activities to navigate between different screens, such as game options and the main game board.

Requirements:

Minimum Hardware/software requirements: Any device running Android 11(API LEVEL 30) with a method of installing offline '.apk' files.

Recommended Hardware/software requirements: Any device running Android 13(API LEVEL 33) with a method of installing offline '.apk' files.

Internet connectivity: not required

Version history TTT3

0.1-TTT1

0.2-TTT2

0.3-TTT3(continued this)

1.1-buttons and text design

1.2-base game logic and linking buttons

1.3-base design implementations and modifications

2.1-loading screen 1st implementation

2.2- loading screen modifications and design changes

2.3-loading bar implementation and final design loading screen changes

3.1-pause menu implementation

3.2-credit screen implementation

3.3-college logo addition with link to college website

4.1-overall win system introduction

5.1-game start menu creation with more game options

6.1-introduction to one player and two player mode

7.1-animations for round wins and overall wins

8.1- Added options for 2 difficulties for the bot, easy and hard(1 player mode)

9.0-QC with final modifications and small changes

Basic explanation of the use of xml files and java files:

The XML layout serves as the visual representation of the game options screen, while the Java class handles the logic and behaviour associated with user interactions on that screen. The XML layout and Java class are linked through the 'setContentView' call and the use of 'findViewById' to reference UI elements. This connection enables the user interface to interact with the underlying code and data.

References:

1) LoadingActivity.java and activity_loading.xml: Loading screen when the user first opens the app.

2) GameoptionsActivity.java and activity_game_options.xml: The game options screen the user gets after the loading is complete, allowing the user to choose the options for the game.

3) MainActivity.java and activity_main.xml: The main user playable game after the game options menu.

4) Pause_menu.xml: An extra clickable and interactive dialog which has extra game options during a running game.

5) Credits_dialog.xml: A secondary dialog which can be activated from the game starting screen or the pause menu.

Android manifest file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
```

//This section defines the XML namespace declarations for the Android manifest file.

```
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="TicTacToe"
    android:supportsRtl="true"
    android:theme="@style/Theme.TTT3"
    tools:targetApi="33">
```

//The `<application>` tag defines the properties and components of your Android application.

- `android:allowBackup="true"`

//indicates that the app is allowed to participate in the backup and restore process.

- `android:dataExtractionRules="@xml/data_extraction_rules"`

//specifies a set of rules for data extraction during backup.

- `android:fullBackupContent="@xml/backup_rules"`

//defines a set of rules for specifying which app data should be backed up.

- `android:icon="@mipmap/ic_launcher"`

//sets the app icon.

- `android:label="TicTacToe"`

//sets the app's display label.

- `android:supportsRtl="true"`

//indicates that the app supports right-to-left layout for languages that require it.

- `android:theme="@style/Theme.TTT3"`

//sets the app's theme to the style defined in `Theme.TTT3`.

- `tools:targetApi="33"`

//specifies the target API level for tools.

```
<activity
  android:name=".LoadingActivity"
  android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

//This defines an activity named `LoadingActivity`, which is the entry point of the app.

- `android:name=".LoadingActivity"`

//specifies the class name of the activity.

- `android:exported="true"`

//indicates that this activity is accessible to other apps.

- The `<intent-filter>` section defines the action and category for this activity. In this case, it specifies that this activity is the main entry point (launcher) for the app.

```
<activity
    android:name=".GameOptionsActivity"
    android:exported="true"/>
```

//This defines an activity named `GameOptionsActivity`.

- `android:name=".GameOptionsActivity"` specifies the class name of the activity.

- `android:exported="true"` indicates that this activity is accessible to other apps.

```
<activity
    android:name=".MainActivity"
    android:exported="true" />
```

//This defines an activity named `MainActivity`.

- `android:name=".MainActivity"`

//specifies the class name of the activity.

- `android:exported="true"`

//indicates that this activity is accessible to other apps.

```
</application>
```

//Closing tag for the `<application>` element.

```
</manifest>
```

//Closing tag for the `<manifest>` element.

In summary,

this Android manifest file defines the properties and components of your app, including its activities, entry points, icons, labels, backup settings, and more. It specifies the activities' names, whether they are accessible to other apps, and their intent filters. The manifest plays a crucial role in providing essential information about your app to the Android operating system.

Gradle build file:

```
plugins {
    id("com.android.application")
}
```

//This section specifies the plugins used in the build process. Here, the `com.android.application` plugin is applied, indicating that this is an Android application project.

```
android {
    namespace = "com.mpan.ttt3"
    compileSdk = 33
```

//The `android` block contains configuration settings related to the Android build process.

- `namespace = "com.mpan.ttt3"` specifies the namespace for the app's package.

- `compileSdk = 33` sets the Android SDK version that the app is compiled against.

```
    defaultConfig {
        applicationId = "com.mpan.ttt3"
        minSdk = 30
        targetSdk = 33
        versionCode = 1
        versionName = "9.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }
```

//The `defaultConfig` block defines default configuration settings for the app.

- `applicationId = "com.mpan.ttt3"` sets the unique identifier for the app package.

- `minSdk = 30` specifies the minimum Android API level required to run the app.

- `targetSdk = 33`

//indicates the target Android API level the app is designed for.

- `versionCode = 1`

//is an integer used to represent the version of the app.

- `versionName = "9.0"`

//is a user-friendly string representing the version of the app.

- `testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"`

//specifies the test runner for running unit tests.

```
    buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
```

//The `buildTypes` block defines different build types, such as `release` and `debug`.

- Inside the `release` build type, `isMinifyEnabled = false` indicates that code minification is disabled.

- `proguardFiles(...)` specifies ProGuard rules files for code shrinking and obfuscation.

```
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }
}
```

The `compileOptions` block configures Java compatibility settings for the app's source and target compatibility. Here, it specifies compatibility with Java 8.

```
dependencies {
    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.9.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    implementation ("com.airbnb.android:lottie:3.7.0")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
}
```

The `dependencies` block lists the external libraries and dependencies used by the app.

- `implementation("androidx.appcompat:appcompat:1.6.1")`, `implementation("com.google.android.material:material:1.9.0")`, and `implementation("androidx.constraintlayout:constraintlayout:2.1.4")` are implementations of AndroidX libraries used for UI components.

- `implementation ("com.airbnb.android:lottie:3.7.0")` is a library for adding Lottie animations to the app.

- `testImplementation("junit:junit:4.13.2")` specifies the JUnit library for unit testing.

- `androidTestImplementation("androidx.test.ext:junit:1.1.5")` and `androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")` are dependencies for Android instrumentation tests.

In summary,

this Gradle build file contains configuration settings for the Android build process, including SDK versions, dependencies, build types, Java compatibility, and more. It defines the app's dependencies on various AndroidX libraries, material design components, and external libraries like Lottie for animations. The file plays a crucial role in specifying how the app is built, packaged, and assembled.

Loading activity xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:background="@android:color/black">

    <!-- ImageView for the Logo -->
    <ImageView
        android:id="@+id/logo_image"
        android:layout_width="120dp"
        android:layout_height="120dp"
        android:layout_centerInParent="true"
        android:layout_marginBottom="32dp"
        android:src="@mipmap/ic_launcher"
        tools:ignore="ContentDescription,ImageContrastCheck" />

    <!-- ProgressBar for the Loading Spinner -->
    <ProgressBar
        android:id="@+id/progressBar1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/logo_image"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp" />
</RelativeLayout>
```

Explanation:

1. The root element of the layout is a `RelativeLayout`, which allows you to arrange its child views relative to each other.

2. `xmlns:android="http://schemas.android.com/apk/res/android"` specifies the XML namespace for Android attributes.

3. `xmlns:tools="http://schemas.android.com/tools"` is a namespace declaration for attributes used by the Android development tools, which are only used at design time and ignored at runtime.

4. `android:layout_width="match_parent"` and `android:layout_height="match_parent"` indicate that the layout should match the width and height of its parent container.

5. `android:gravity="center"` specifies that the content inside the `RelativeLayout` should be centered both horizontally and vertically.

6. `android:background="@android:color/black"` sets the background color of the layout to black.

7. The `<ImageView>` element with `android:id="@+id/logo_image"` is used to display an image (logo) in the center of the layout. Here's what the attributes mean:

- `android:layout_width="120dp"` and `android:layout_height="120dp"` set the dimensions of the ImageView to 120 density-independent pixels (dp).

- `android:layout_centerInParent="true"` centers the ImageView both horizontally and vertically within the parent layout.

- `android:layout_marginBottom="32dp"` adds a bottom margin of 32dp to create some space between the ImageView and the ProgressBar.

- `android:src="@mipmap/ic_launcher"` sets the image source to the app's launcher icon (mipmap/ic_launcher). This is just a placeholder; you would typically replace it with your actual logo.

8. The `<ProgressBar>` element with `android:id="@+id/progressBar1"` is used to display a loading spinner below the logo. Here's what the attributes mean:

- `android:layout_below="@id/logo_image"` positions the ProgressBar below the ImageView with the ID `logo_image`.

- `android:layout_centerHorizontal="true"` centers the ProgressBar horizontally within the parent layout.

- `android:layout_marginTop="16dp"` adds a top margin of 16dp to create some space between the ImageView and the ProgressBar.

9. The `tools:ignore` attributes are used to specify tools that should ignore certain warnings or checks during layout editing in Android Studio. In this case, they are used to ignore warnings related to content description and image contrast checks.

Loading activity java

```
package com.mpan.ttt3;
// Import necessary classes and libraries
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.Looper;
import android.view.View;
import android.widget.ImageView;
import android.widget.ProgressBar;
import androidx.appcompat.app.AppCompatActivity;

public class LoadingActivity extends AppCompatActivity {

    private static final long DELAY_DURATION = 2500;

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

        // Get references to UI elements
        ProgressBar spinner = findViewById(R.id.progressBar1);
        ImageView logoImage = findViewById(R.id.logo_image);

        // Create a new HandlerThread for background processing
        HandlerThread handlerThread = new HandlerThread("LoadingThread");
        handlerThread.start();
        // Get the Looper and create a Handler for the new thread
        Looper looper = handlerThread.getLooper();
        Handler handler = new Handler(looper);

        // Schedule a delayed task on the Handler
        handler.postDelayed(new Runnable() {
          @Override
          public void run() {
            // This code runs on the background thread

            // Switch to the UI thread to update UI elements
            runOnUiThread(new Runnable() {
              @Override
              public void run() {
                // This code runs on the UI thread

                // Hide the spinner and logo
                spinner.setVisibility(View.GONE);
                logoImage.setVisibility(View.GONE);

                // Create an Intent to start the GameOptionsActivity
                Intent intent = new Intent(LoadingActivity.this, GameOptionsActivity.class);
                startActivity(intent);
                finish(); // Finish the LoadingActivity to prevent going back to it
              }
            });
          }
        }, DELAY_DURATION); // Delay for 2500 milliseconds (2.5 seconds)
    }
}
```

Explanation:

1. `LoadingActivity` is an `AppCompatActivity` that serves as a loading screen before starting the main game activity.

2. In the `onCreate` method, the layout specified in `activity_loading.xml` is set as the content view of the activity.

3. The `ProgressBar` and `ImageView` UI elements are obtained from the layout using their respective IDs.

4. A `HandlerThread` named "LoadingThread" is created. This is a background thread that can be used to perform tasks off the main UI thread.

5. The `Looper` of the `HandlerThread` is obtained, and a `Handler` is created using that `Looper`. This allows us to post and schedule tasks on the background thread.

6. A delayed task is posted to the background thread's `Handler`. This task will run after a delay of `DELAY_DURATION` milliseconds (2.5 seconds).

7. Inside the delayed task, the code runs on the background thread. However, to update the UI elements, we switch to the UI thread using the `runOnUiThread` method.

8. In the UI thread, the spinner (progress bar) and logo are hidden (`setVisibility(View.GONE)`).

9. An `Intent` is created to start the `GameOptionsActivity`.

10. The `startActivity` method is called with the `Intent` to switch to the `GameOptionsActivity`.

11. The `finish` method is called to finish the `LoadingActivity`. This prevents the user from returning to the loading screen by pressing the back button.

In summary,

This XML layout defines a simple loading screen with a centered logo image and a loading spinner below it. The use of relative positioning allows you to create a visually appealing layout that adapts to different screen sizes and orientations.

In java this code creates a loading screen with a spinner and logo, delays for a short duration, and then switches to the main game activity (`GameOptionsActivity`) while hiding the loading screen. The use of a background thread (`HandlerThread`) and switching to the UI thread (`runOnUiThread`) ensures that UI updates and task scheduling are performed correctly.

Game Options xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:background="@android:color/black">

    <!-- TextView for Game Mode Selection -->
    <TextView
        android:id="@+id/text_view_mode"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select Game Mode:"
        android:textColor="@android:color/white"
        android:textSize="18sp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"
        tools:ignore="HardcodedText" />

    <!-- ToggleButton for Game Mode 1 Player/2 Players -->
    <ToggleButton
        android:id="@+id/toggle_mode1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/text_view_mode"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"
        android:textOff="1 Player"
        android:textOn="2 Players"
        tools:ignore="HardcodedText" />

    <!-- ToggleButton for Game Difficulty Easy/Hard -->
    <ToggleButton
        android:id="@+id/toggle_mode2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/toggle_mode1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"
        android:textOff="Easy"
        android:textOn="Hard"
        tools:ignore="HardcodedText" />

    <!-- EditText for Player 1 Name -->
    <EditText
        android:id="@+id/edit_text_player1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/toggle_mode2"
        android:layout_marginTop="16dp"
        android:hint="Player 1 Name"

tools:ignore="Autofill,HardcodedText,TextContrastCheck,TextFields,TouchTargetSizeCheck,VisualLintTextFieldSize" />

    <!-- EditText for Player 2 Name (Initially Invisible) -->
    <EditText
        android:id="@+id/edit_text_player2"
        android:layout_width="match_parent"
```

```xml
        android:layout_height="wrap_content"
        android:layout_below="@id/edit_text_player1"
        android:layout_marginTop="8dp"
        android:hint="Player 2 Name"
        android:visibility="gone"
        tools:ignore="Autofill,HardcodedText,TextFields" />

    <!-- Button to Start the Game -->
    <Button
        android:id="@+id/button_start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/edit_text_player2"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"
        android:text="Start Game"
        tools:ignore="HardcodedText" />

    <!-- Button to View Credits -->
    <Button
        android:id="@+id/button_credits"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button_start"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="8dp"
        android:text="Credits"
        tools:ignore="HardcodedText" />

</RelativeLayout>
```

Explanation:

1. The root element of the layout is a `RelativeLayout`, which allows you to arrange its child views relative to each other.

2. Inside the layout, there are various UI components defined:

- `<TextView>` with `android:id="@+id/text_view_mode"` is used to display the instruction for game mode selection. It's centered horizontally and has a top margin of 32dp.
- `<ToggleButton>` with `android:id="@+id/toggle_mode1"` is for selecting the number of players (1 or 2 players). It's positioned below the TextView and centered horizontally.
- `<ToggleButton>` with `android:id="@+id/toggle_mode2"` is for selecting the game difficulty (Easy or Hard). It's positioned below the previous ToggleButton and centered horizontally.
- Two `<EditText>` fields with `android:id="@+id/edit_text_player1"` and `android:id="@+id/edit_text_player2"` are used to input player names. `edit_text_player2` is initially set to be invisible using `android:visibility="gone"`.
- Two `<Button>` components with `android:id="@+id/button_start"` and `android:id="@+id/button_credits"` are used to start the game and view credits, respectively.

3. The layout uses `tools:ignore` attributes to suppress certain warnings or checks during layout editing in Android Studio, like hardcoded text, autofill, and touch target size.

4. `android:layout_below` attribute is used to position components below others.

5. `android:textOff` and `android:textOn` attributes in ToggleButtons specify the text for the OFF and ON states of the toggle.

6. `android:hint` attribute in EditTexts provides a hint to users about what to enter.

7. The `android:visibility` attribute controls the visibility of the EditText for player 2. Initially, it's set to `gone`, meaning it won't be visible. It will become visible programmatically when needed.

Game options java

```
package com.mpan.ttt3;

// Import necessary classes and libraries

import android.app.Activity;

import android.app.Dialog;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.view.ViewGroup;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ToggleButton;


public class GameOptionsActivity extends Activity {


    private ToggleButton toggleMode1;

    private ToggleButton toggleMode2;

    private EditText editTextPlayer1;

    private EditText editTextPlayer2;


    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_game_options);


        // Initialize UI components

        toggleMode1 = findViewById(R.id.toggle_mode1);

        toggleMode2 = findViewById(R.id.toggle_mode2);

        editTextPlayer1 = findViewById(R.id.edit_text_player1);

        editTextPlayer2 = findViewById(R.id.edit_text_player2);
        Button buttonStart = findViewById(R.id.button_start);
        Button buttonCredits = findViewById(R.id.button_credits);

        // Button to start the game
        buttonStart.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Get player names and game options
                String player1Name = editTextPlayer1.getText().toString();
                String player2Name = toggleMode1.isChecked() ? editTextPlayer2.getText().toString() :
"BOT";
                boolean isTwoPlayersMode = toggleMode1.isChecked();
                boolean isHardDifficulty = toggleMode2.isChecked();

                // Start the main game activity and pass data through intent
                Intent intent = new Intent(GameOptionsActivity.this, MainActivity.class);
                intent.putExtra("player1Name", player1Name);
                intent.putExtra("player2Name", player2Name);
                intent.putExtra("isTwoPlayersMode", isTwoPlayersMode);
                intent.putExtra("isHardDifficulty", isHardDifficulty);
                startActivity(intent);
                finish(); // Close the current activity
            }
        });

        // Button to show credits dialog
        buttonCredits.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                showCredits();
            }
        });

        // ToggleButton listener to handle visibility of player 2 input based on game mode
        toggleMode1.setOnCheckedChangeListener((buttonView, isChecked) -> {
            if (isChecked) {
                toggleMode2.setVisibility(View.GONE); // Hide difficulty toggle when in 1 player mode
                editTextPlayer2.setVisibility(View.VISIBLE); // Show player 2 input
            } else {
                toggleMode2.setVisibility(View.VISIBLE); // Show difficulty toggle in 2 players mode
                editTextPlayer2.setVisibility(View.GONE); // Hide player 2 input
            }
        });
    }

    // Method to show the credits dialog
    private void showCredits() {
        Dialog creditsDialog = new Dialog(this);
        creditsDialog.setContentView(R.layout.credits_dialog); // Set custom layout for the dialog
        creditsDialog.setCancelable(true);
        creditsDialog.getWindow().setLayout(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT);

        creditsDialog.show(); // Show the dialog
    }
}
```

Explanation:

1. The `GameOptionsActivity` class extends `Activity`, indicating that it's an Android activity responsible for managing the game options screen.

2. The `onCreate` method is the entry point of the activity, called when the activity is created. Here's what it does:

- It sets the content view to the layout defined in `activity_game_options.xml`.
- Initializes UI components such as `ToggleButton`, `EditText`, and `Button` views by finding them using their IDs.

3. The `buttonStart` click listener handles the logic when the "Start Game" button is clicked. It does the following:

- Retrieves player names and game options (1 player mode, hard difficulty) from UI components.
- Creates an intent to start the `MainActivity` and passes the game data as extras in the intent.
- Starts the `MainActivity` and finishes the current `GameOptionsActivity`.

4. The `buttonCredits` click listener displays a dialog to show credits when the "Credits" button is clicked. It uses a custom dialog layout defined in `credits_dialog.xml`.

5. The `toggleMode1` listener (`setOnCheckedChangeListener`) toggles the visibility of `toggleMode2` (difficulty toggle) and `editTextPlayer2` (player 2 input) based on the selected game mode (1 or 2 players).

6. The `showCredits()` method creates and displays a custom dialog using a layout defined in `credits_dialog.xml`.

7. Note the use of `findViewById` to get references to UI elements by their IDs.

In summary,

This xml layout provides a user interface for selecting game mode, difficulty, entering player names, and starting the game or viewing credits. The use of relative positioning allows for a structured and visually appealing arrangement of UI components.

In this java code `GameOptionsActivity` class handles the user interface and interaction for selecting game options, including the number of players and difficulty level. It also provides a mechanism to show credits in a dialog.

Main activity Xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <!-- LottieAnimationView for Thumbs Up -->
    <com.airbnb.lottie.LottieAnimationView
        android:id="@+id/lottie_thumbs_up"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="gone"
        app:lottie_autoPlay="true"/>

    <!-- LottieAnimationView for Party Emoji -->
    <com.airbnb.lottie.LottieAnimationView
        android:id="@+id/lottie_party_emoji"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="gone"
        app:lottie_autoPlay="true"/>

    <!-- LottieAnimationView for Sad Face 1 -->
    <com.airbnb.lottie.LottieAnimationView
        android:id="@+id/lottie_sad_face"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="gone"
        app:lottie_autoPlay="true"/>

    <!-- LottieAnimationView for Sad Face 2 -->
    <com.airbnb.lottie.LottieAnimationView
        android:id="@+id/lottie_sad_face2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="gone"
        app:lottie_autoPlay="true"/>

    <!-- Linear Layout for Game UI -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <!-- Relative Layout for Player Info and Pause Button -->
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <!-- Player 1 Score Text View -->
            <TextView
                android:id="@+id/text_view_p1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:freezesText="true"
                android:text="Player 1: 0"
                android:textSize="30sp"
                tools:ignore="HardcodedText" />
        <!-- Player 2 Score Text View -->
        <TextView
            android:id="@+id/text_view_p2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/text_view_p1"
            android:freezesText="true"
            android:text="Player 2: 0"
            android:textSize="30sp"
            tools:ignore="HardcodedText" />

        <!-- Pause Button -->
        <Button
            android:id="@+id/button_pause"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentEnd="true"
            android:layout_centerVertical="true"
            android:layout_marginEnd="33dp"
            android:text="Pause"
            tools:ignore="HardcodedText" />
    </RelativeLayout>

    <!-- Grid of Buttons for Game Board -->
    <!-- 3x3 grid of Buttons arranged in 3 rows (LinearLayouts) -->
    <!-- Each row contains 3 buttons (Buttons) -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">
```

```
        <!-- Row 1 -->
        <Button
            android:id="@+id/button_00"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:freezesText="true"
            android:textSize="60sp"
            tools:ignore="ButtonStyle,NestedWeights,VisualLintButtonSize,SpeakableTextPresentCheck"
/>

        <Button
            android:id="@+id/button_01"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:freezesText="true"
            android:textSize="60sp"
            tools:ignore="ButtonStyle,SpeakableTextPresentCheck,VisualLintButtonSize" />

        <Button
            android:id="@+id/button_02"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:freezesText="true"
            android:textSize="60sp"
            tools:ignore="ButtonStyle,SpeakableTextPresentCheck,VisualLintButtonSize" />
    </LinearLayout>

    <!-- ... Rows 2 and 3 (similar structure as above) ... -->
```

~~~
    </LinearLayout>
</FrameLayout>
```
~~~

Explanation:

1. The XML layout is wrapped in a `FrameLayout`, which is a simple layout manager that stacks its children on top of each other.

2. `LottieAnimationView` elements are used to display animation graphics for various states, such as thumbs up, party emoji, and sad faces. They are set to be initially invisible with `android:visibility="gone"` and set to auto-play their animations.

3. A `LinearLayout` is used to organize the game UI elements vertically.

4. Inside the `LinearLayout`, a `RelativeLayout` contains player score `TextView` and a pause `Button`.

5. The player score `TextView`s (`text_view_p1` and `text_view_p2`) show the scores for Player 1 and Player 2. They are initially set to "Player 1: 0" and "Player 2: 0" with `android:text="Player 1: 0"`.

6. The pause `Button` (`button_pause`) is aligned to the end of the parent layout and has the text "Pause".

7. Another set of `LinearLayout`s is used to create a 3x3 grid of `Button` elements, representing the game board. Each button is given an ID like `button_00`, `button_01`, and so on. These buttons are intended to be used for gameplay.

Main activity java

```java
// Import necessary classes and libraries
import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.app.Dialog;
import android.content.Intent;
import android.graphics.Color;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import java.util.Random;
import com.airbnb.lottie.LottieAnimationView;
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
   // Declare class-level variables
   private Button[][] buttons = new Button[3][3];
   private boolean player1Turn = true;
   private int roundCount;
   private int player1Points;
   private int player2Points;
   private TextView textViewPlayer1;
   private TextView textViewPlayer2;
   private Dialog pauseDialog;
   private String player1Name;
   private String player2Name;
   private boolean isTwoPlayersMode;
   private boolean isPlayerWinsGameAnimationPlaying = false;
   private boolean isGameWinAnimationPlaying = false;
   private boolean isPlayer1Starting = true;
   private boolean isOriginalPlayer1Starting = true;
   private boolean isEasyMode = false;
   private boolean isHardDifficulty = false;
   // This method opens a website when a button is clicked
   public void openKCWebsite(View view) {
       // Create an Intent to open a URL in a web browser
       Uri uri = Uri.parse("https://kccemsr.edu.in/");
       Intent intent = new Intent(Intent.ACTION_VIEW, uri);
       startActivity(intent);
   }

   // This method resets the game board with a delay
   private void resetBoardWithDelay() {
       // Post a delayed action using a Handler to reset the board after a delay
       new Handler(Looper.getMainLooper()).postDelayed(new Runnable() {
           @Override
           public void run() {
               resetBoard();
           }
       }, 1000); // Delay of 1000 milliseconds (1 second)
   }

   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```java
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

// Initialize TextViews for player scores and set background colors
textViewPlayer1 = findViewById(R.id.text_view_p1);
textViewPlayer2 = findViewById(R.id.text_view_p2);
textViewPlayer1.setBackgroundColor(Color.parseColor("#FF0000"));
textViewPlayer2.setBackgroundColor(Color.parseColor("#0000FF"));

// Initialize Button views for the game board and set listeners
// This loop iterates through the 3x3 grid of buttons
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        // Construct the resource ID for each button using its index
        String buttonID = "button_" + i + j;
        int resID = getResources().getIdentifier(buttonID, "id", getPackageName());
        buttons[i][j] = findViewById(resID);

        // Set background color and click listener for each button
        buttons[i][j].setBackgroundColor(Color.parseColor("#FF8C00"));
        buttons[i][j].setOnClickListener(this);
    }
}

// Get data from the intent that started this activity
Intent intent = getIntent();
player1Name = intent.getStringExtra("player1Name");
player2Name = intent.getStringExtra("player2Name");
isTwoPlayersMode = intent.getBooleanExtra("isTwoPlayersMode", true);

// Update TextViews with player names and points
```

```
        textViewPlayer1.setText(player1Name + ": " + player1Points);

        textViewPlayer2.setText(player2Name + ": " + player2Points);


        // Set a click listener for the "Pause" button

        Button buttonPause = findViewById(R.id.button_pause);

        buttonPause.setOnClickListener(new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                pauseDialog.show(); // Show the pause dialog

            }

        });


        // Initialize the pause dialog with its layout and options

        pauseDialog = new Dialog(this);

        pauseDialog.setContentView(R.layout.pause_menu);

        pauseDialog.setCancelable(false);

        pauseDialog.getWindow().setLayout(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT);


        // Set click listeners for buttons within the pause dialog

        Button buttonResume = pauseDialog.findViewById(R.id.button_resume);

        Button buttonResetGame = pauseDialog.findViewById(R.id.button_reset);

        Button buttonNewGame = pauseDialog.findViewById(R.id.button_new_game);

        Button buttonCredits = pauseDialog.findViewById(R.id.button_credits);


        // ... Set click listeners and actions for the buttons


        // Determine if player 1 starts or use the original setting

        player1Turn = isPlayer1Starting;


    }
    // This method displays the credits dialog with a clickable logo button
    private void showCredits() {
        Dialog creditsDialog = new Dialog(this);
        creditsDialog.setContentView(R.layout.credits_dialog);
        creditsDialog.setCancelable(true);
        creditsDialog.getWindow().setLayout(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT);

        // Get the logo button and set a click listener to open a website
        ImageButton logoButton = creditsDialog.findViewById(R.id.logoButton);
        logoButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                openKCWebsite(v);
            }
        });

        creditsDialog.show();
    }

    // This method is called when a button on the game board is clicked
    @Override
    public void onClick(View v) {
        // Check if the clicked button is empty, if not, return
        if (!((Button) v).getText().toString().equals("")) {
            return;
        }

        // Update the button's text and appearance based on the current player's turn
        if (player1Turn) {
            ((Button) v).setText("X");
            v.setBackgroundColor(Color.RED);
            textViewPlayer1.setBackgroundColor(Color.parseColor("#FF0000"));
            textViewPlayer2.setBackgroundColor(Color.parseColor("#0000FF"));
        } else if (isTwoPlayersMode) {
            ((Button) v).setText("O");
            v.setBackgroundColor(Color.BLUE);
            textViewPlayer1.setBackgroundColor(Color.parseColor("#FF0000"));
            textViewPlayer2.setBackgroundColor(Color.parseColor("#0000FF"));
        } else {
            // For single-player mode against AI
            v.setEnabled(false);
            // ... Delayed AI move logic (computerMoveEasy() or computerMoveHard())
            textViewPlayer1.setBackgroundColor(Color.parseColor("#FF0000"));
            textViewPlayer2.setBackgroundColor(Color.parseColor("#0000FF"));
        }

        // Increment round count
        roundCount++;

        // Check for a win or draw
        if (checkForWin()) {
            if (player1Turn) {
                player1Wins();
            } else {
                player2Wins();
            }
        } else if (roundCount == 9) {
            draw();
        } else {
            // Switch turns if the game is not over
            player1Turn = !player1Turn;
```

```
        // If it's AI's turn, make the AI move
        if (!player1Turn && !isTwoPlayersMode) {
            if (isHardDifficulty) {
                computerMoveHard();
            } else {
                computerMoveEasy();
            }
        }
    }
}

// This method checks for a win by examining the game board
private boolean checkForWin() {
    // Create a 2D array to hold the current state of the buttons
    String[][] field = new String[3][3];

    // Populate the array with the text from the buttons
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            field[i][j] = buttons[i][j].getText().toString();
        }
    }

    // Check for horizontal, vertical, and diagonal wins
    // ... Code to check for win conditions
}

// This method is called when player 1 wins a round
private void player1Wins() {
    // Update player 1's score and display animation
    player1Points++;
    updatePointsText();
    resetBoardWithDelay();
    checkOverallWin();

    // Display animation for player 1's win
    if (!isPlayerWinsGameAnimationPlaying && !isGameWinAnimationPlaying) {
        LottieAnimationView thumbsUpAnimation = findViewById(R.id.lottie_thumbs_up);
        // ... Code to play thumbs up animation
    }
}

// This method is called when player 2 (or the bot) wins a round
private void player2Wins() {
    // Update player 2's score and display animation
    player2Points++;
    updatePointsText();
    resetBoardWithDelay();
    checkOverallWin();

    // Display animation for player 2's win (or bot's win)
    if (!isPlayerWinsGameAnimationPlaying && !isGameWinAnimationPlaying) {
        if (!isTwoPlayersMode) {
            LottieAnimationView sadfaceAnimation = findViewById(R.id.lottie_sad_face);
            // ... Code to play sad face animation for bot's win
        } else {
            LottieAnimationView thumbsUpAnimation = findViewById(R.id.lottie_thumbs_up);
            // ... Code to play thumbs up animation for player 2's win
        }
    }
}
```

```
* Handles the situation when the game ends in a draw.
 * Displays a toast message indicating a draw and then resets the game board with a delay.
 */
private void draw() {
    // Display a toast message to inform the players about the draw.
    Toast.makeText(this, "Draw!", Toast.LENGTH_SHORT).show();

    // Reset the game board after a short delay to prepare for the next round.
    resetBoardWithDelay();
}
    // This method updates the text views to display the current scores of player 1 and player 2
    private void updatePointsText() {
        textViewPlayer1.setText(player1Name + ": " + player1Points);
        textViewPlayer2.setText(player2Name + ": " + player2Points);

        // Set background colors of text views based on the current player's turn
        textViewPlayer1.setBackgroundColor(Color.parseColor("#FF0000"));
        textViewPlayer2.setBackgroundColor(Color.parseColor("#0000FF"));
    }

    // This method resets the game board with a delay after a round
    private void resetBoardWithDelay() {
        new Handler(Looper.getMainLooper()).postDelayed(new Runnable() {
            @Override
            public void run() {
                resetBoard();
            }
        }, 1000); // Delay of 1000 milliseconds (1 second)
    }

    // This method resets the entire game
```

```
private void resetGame() {
   player1Points = 0;
   player2Points = 0;
   isPlayer1Starting = isOriginalPlayer1Starting; // Reset the starting player
   updatePointsText();
   resetBoard(); // Reset the game board
}

// This method checks if a player has won the overall game
private void checkOverallWin() {
   if (player1Points >= 4) {
      playerWinsGame(1); // Player 1 wins the game
      isPlayerWinsGameAnimationPlaying = false;
   } else if (player2Points >= 4) {
      playerWinsGame(2); // Player 2 (or bot) wins the game
      isPlayerWinsGameAnimationPlaying = false;
   }
}

// This method handles a player winning the entire game
private void playerWinsGame(int player) {
   // Display a toast with the winner's message
   String winnerMessage;
   if (!isTwoPlayersMode && player == 2) {
      winnerMessage = "Bot wins the game!";
   } else {
      winnerMessage = "Player " + player + " wins the game!";
   }
   Toast.makeText(this, winnerMessage, Toast.LENGTH_SHORT).show();

   // Reset scores and board for a new game
```

```
        isGameWinAnimationPlaying = true;
        player1Points = 0;
        player2Points = 0;
        updatePointsText();
        resetBoardWithDelay();

        // Display a celebration animation
        LottieAnimationView partyEmojiAnimation = findViewById(R.id.lottie_party_emoji);
        partyEmojiAnimation.bringToFront();
        partyEmojiAnimation.setVisibility(View.VISIBLE);
        partyEmojiAnimation.setAnimation(R.raw.lottie_party_pop);
        partyEmojiAnimation.playAnimation();
        partyEmojiAnimation.addAnimatorListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                partyEmojiAnimation.setVisibility(View.GONE);
                isGameWinAnimationPlaying = false;
            }
        });
    }

    // This method handles the easy-level AI move
    private void computerMoveEasy() {
        if (!player1Turn && !isTwoPlayersMode) {
            // ... AI move logic for easy mode
        }
    }

    // This method handles the hard-level AI move
    private void computerMoveHard() {
        if (!player1Turn && !isTwoPlayersMode) {
            // ... AI move logic for hard mode
        }
    }

    // This method is called when the activity is being saved (e.g., during screen orientation changes)
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        // Save game state data
        outState.putInt("roundCount", roundCount);
        outState.putInt("player1Points", player1Points);
        outState.putInt("player2Points", player2Points);
        outState.putBoolean("player1Turn", player1Turn);
    }

    // This method is called when the activity is being restored after being saved
    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);

        // Restore game state data
        roundCount = savedInstanceState.getInt("roundCount");
        player1Points = savedInstanceState.getInt("player1Points");
        player2Points = savedInstanceState.getInt("player2Points");
        player1Turn = savedInstanceState.getBoolean("player1Turn");
    }
}
// This method handles the easy-level AI move
private void computerMoveEasy() {
    if (!player1Turn && !isTwoPlayersMode) {
```

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (buttons[i][j].getText().toString().equals("")) {
            // Simulate player's move to check for win
            buttons[i][j].setText("O");
            if (checkForWin()) {
                buttons[i][j].setText("");
                buttons[i][j].setText("O");
                buttons[i][j].setBackgroundColor(Color.BLUE);
                roundCount++;
                player2Wins();
                return;
            } else {
                buttons[i][j].setText("");
            }

            buttons[i][j].setText("X");
            if (checkForWin()) {
                buttons[i][j].setText("");
                buttons[i][j].setText("O");
                buttons[i][j].setBackgroundColor(Color.BLUE);
                roundCount++;
                player1Turn = !player1Turn;
                return;
            } else {
                buttons[i][j].setText("");
            }
        }
    }
}
```

```
        Random random = new Random();
        int row, col;
        do {
            row = random.nextInt(3);
            col = random.nextInt(3);
        } while (!buttons[row][col].getText().toString().equals(""));

        buttons[row][col].setText("O");
        buttons[row][col].setBackgroundColor(Color.BLUE);

        roundCount++;

        if (checkForWin()) {
            player2Wins();
        } else if (roundCount == 9) {
            draw();
        } else {
            player1Turn = !player1Turn;
        }
    }
}
// This method handles the hard-level AI move
private void computerMoveHard() {
    if (!player1Turn && !isTwoPlayersMode) {
        // Check for winning moves
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (buttons[i][j].getText().toString().equals("")) {
                    // Simulate player's move to check for win
                    buttons[i][j].setText("O");
                    if (checkForWin()) {
```

```
                buttons[i][j].setText("");
                buttons[i][j].setText("O");
                buttons[i][j].setBackgroundColor(Color.BLUE);
                roundCount++;
                player2Wins();
                return;
            } else {
                buttons[i][j].setText("");
            }
        }
    }
}

// Check for blocking moves
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (buttons[i][j].getText().toString().equals("")) {
            // Simulate opponent's move to check for potential win
            buttons[i][j].setText("X");
            if (checkForWin()) {
                buttons[i][j].setText("O");
                buttons[i][j].setBackgroundColor(Color.BLUE);
                roundCount++;
                player1Turn = !player1Turn;
                return;
            } else {
                buttons[i][j].setText("");
            }
        }
    }
}
```

```
    // Choose a random available move
    Random random = new Random();
    int row, col;
    do {
      row = random.nextInt(3);
      col = random.nextInt(3);
    } while (!buttons[row][col].getText().toString().equals(""));

    buttons[row][col].setText("O");
    buttons[row][col].setBackgroundColor(Color.BLUE);

    roundCount++;

    if (checkForWin()) {
      player2Wins();
    } else if (roundCount == 9) {
      draw();
    } else {
      player1Turn = !player1Turn;
    }
  }
}
}
```

Explanation:

1. The `MainActivity` class extends `AppCompatActivity` and implements the `View.OnClickListener` interface to handle button clicks.

2. The `Button[][] buttons` array is used to store references to the buttons in the 3x3 grid of the game board.

3. `player1Turn` is a boolean variable that keeps track of which player's turn it is.

4. `roundCount` keeps track of the number of rounds played in the game.

5. `player1Points` and `player2Points` store the scores of the two players.

6. `textViewPlayer1` and `textViewPlayer2` are `TextView` elements that display the scores of the players.

7. The `openKCWebsite(View view)` method is called when a specific button is clicked to open a website using an implicit intent.

8. `resetBoardWithDelay()` method resets the game board with a delay of 1 second using a `Handler`.

9. In the `onCreate` method:

   - The layout is set using `setContentView(R.layout.activity_main)`.

   - Views are initialized, listeners are set, and data is retrieved from the intent that started the activity.

   - The pause dialog is created, and click listeners are set for its buttons.

10. The `onClick(View v)` method is where the main game logic is implemented. It handles button clicks and contains code for updating the game state, checking wins, and displaying animations.

11. `showCredits()`: This method displays a dialog with credits and a clickable logo button that opens a website when clicked.

12. `onClick(View v)`: This method is the heart of the game logic. It's called when a button on the game board is clicked. It handles updating the game state based on player moves, checking for wins or draws, switching turns, and making AI moves in single-player mode.

13. `checkForWin()`: This method checks the current state of the game board to determine if a player has won. It examines rows, columns, and diagonals to find matching symbols.

14. `player1Wins()`: This method is called when player 1 wins a round. It updates player 1's score, triggers animations, and handles overall game wins.

15. `player2Wins()`: This method is called when player 2 (or the bot) wins a round. It updates player 2's score, triggers animations, and handles overall game wins. The animation displayed depends on the game mode.

16.'draw()': This method is called when the game ends in a draw, which means that all available spaces on the game board have been filled, but neither player has won. In this case, a toast message is displayed briefly to inform the players that the game has ended in a draw. The Toast.makeText() method is used to create a short-duration popup message at the bottom of the screen with the "Draw!" message.

17. `updatePointsText()`: This method updates the text views that display the scores of player 1 and player 2. It also changes the background colors of the text views based on the current player's turn.

18. `resetBoardWithDelay()`: This method resets the game board with a delay of 1 second after a round. It uses a `Handler` to achieve the delay.

19. `resetGame()`: This method resets the entire game. It resets scores, the starting player, updates the points text, and resets the game board.

20. `checkOverallWin()`: This method checks if a player has won the overall game (reached 4 points). If so, it calls `playerWinsGame()` to handle the game win.

21. `playerWinsGame(int player)`: This method is called when a player wins the entire game. It displays a toast message with the winner's message, triggers a celebration animation, resets scores, and resets the game board.

22. `computerMoveEasy()`: This method handles the AI's move in easy mode. It determines a move to make based on simple logic.

23. `computerMoveHard()`: This method handles the AI's move in hard mode. It uses more advanced logic to determine a move that can block the opponent or lead to a win.

24. `onSaveInstanceState(Bundle outState)`: This method is called when the activity is being saved, typically during screen orientation changes. It saves essential game state data to the provided `Bundle`.

25. `onRestoreInstanceState(Bundle savedInstanceState)`: This method is called when the activity is being restored after being saved. It restores the game state data from the `Bundle`.

26.`computerMoveEasy()`: This method handles the AI's move in easy mode. The AI's strategy is relatively simple: it tries to block the player from winning or attempts to make a random valid move.

a. The AI iterates over each cell of the game board (a 3x3 grid) using nested loops.

b. For each empty cell, it simulates placing an "O" in that cell and checks if that move would result in a win for the AI. If so, the AI plays that move to win the game.

c. If the AI's winning move is not available, it simulates placing an "X" in the same cell and checks if that move would block the player from winning. If so, the AI plays that move to block the player.

d. If neither a winning nor a blocking move is possible, the AI makes a random move by generating random row and column indices until it finds an empty cell.

e. After making a move, the AI updates the game board, checks for a win, draw, or continues the game by toggling the player's turn.

27. `computerMoveHard()`: This method handles the AI's move in hard mode. The AI's strategy is more advanced than in easy mode. It first checks for potential winning moves, then for potential blocking moves to prevent the player from winning, and if neither is available, it makes a random move.

a. The AI first checks for any winning moves by iterating over each cell of the game board, similar to `computerMoveEasy()`. If a winning move is available, the AI plays that move.

b. If no winning move is available, the AI iterates over each cell again to check for potential blocking moves. It simulates the player's move and checks if the player could win in the next turn. If a blocking move is possible, the AI plays that move to block the player.

c. If neither a winning nor a blocking move is possible, the AI makes a random move, similar to `computerMoveEasy()`.

d. After making a move, the AI updates the game board, checks for a win, draw, or continues the game by toggling the player's turn.

Summary

This xml layout is designed to create a simple Tic-Tac-Toe game interface. It includes animations and UI elements to display scores and control the game. The `FrameLayout` acts as a container for all the UI elements, and the `LinearLayout` is used to vertically stack the game components. The layout hierarchy is well-structured to create an organized and visually appealing game board.

This java code is for a two-player Tic-Tac-Toe game with options for single-player mode against a bot (AI). It handles player turns, updating scores, checking for wins and draws, showing animations, and more. The comments within the code provide explanations for each section and function.

These methods continue the implementation of game logic, score tracking, animations, and win/draw conditions. The comments within the code help explain the purpose of each method and the logic it contains. The `MainActivity` class effectively manages the gameplay and user interactions in your Tic-Tac-Toe game.

These methods complete the implementation of the Tic-Tac-Toe game. They handle game resets, AI moves, saving and restoring game state during orientation changes, and overall game wins. The comments provide insights into the purpose and functionality of each method.

These methods provide the AI's decision-making process for making moves in the Tic-Tac-Toe game. The `computerMoveEasy()` method is more focused on making valid moves, while `computerMoveHard()` implements a more strategic approach by considering potential winning and blocking moves.

Pause menu xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical"
  android:padding="16dp">
<Button
  android:id="@+id/button_resume"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Resume"
  tools:ignore="HardcodedText,VisualLintButtonSize" />
<Button
    android:id="@+id/button_reset"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Reset Game"
    tools:ignore="HardcodedText,VisualLintButtonSize" />

  <Button
    android:id="@+id/button_new_game"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="New Game"
    tools:ignore="HardcodedText,VisualLintButtonSize" />
  <Button
    android:id="@+id/button_credits"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Credits"
    tools:ignore="HardcodedText,VisualLintButtonSize" />
</LinearLayout>
```

Explanation:

This XML starts with a `LinearLayout` element, which is a common layout used to arrange child views in a linear manner (either horizontally or vertically). Here, the orientation is set to `"vertical"` to stack the child views vertically. The `padding` attribute adds some spacing around the content of the layout.

Next, there are four **`Button`** elements representing the buttons within the pause menu dialog:-

Each button has an `id` attribute to uniquely identify it. The `layout_width` is set to `"match_parent"` to make the button occupy the full width of the dialog, and the `layout_height` is set to `"wrap_content"` to adjust the height based on the button's content. The `text` attribute sets the text displayed on the button.

The `tools:ignore` attribute is used to indicate that lint checks for hardcoded text and button size should be ignored during development.

You have similar `Button` elements for "Reset Game," "New Game," and "Credits" with similar attributes.

The layout ends with the closing tags for the `LinearLayout` and the root element of the XML.

Summary

This XML layout defines a simple pause menu dialog for your Android game. The dialog contains four buttons stacked vertically: "Resume," "Reset Game," "New Game," and "Credits." Players can interact with these buttons to perform actions such as resuming the game, resetting the game, starting a new game, or viewing credits. This custom dialog enhances the user experience by providing a convenient and visually appealing way to manage game options while the game is paused.

Credit menu dialog xml

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:orientation="vertical"
   android:padding="16dp">

<TextView
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Credits"
   android:textSize="20sp"
   android:textStyle="bold"
   android:gravity="center"
   android:padding="8dp"
   tools:ignore="HardcodedText" />

<TextView
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Developed/Coded/Debugged by: \n Armaan Nakhuda B-02 \n Sushant Navle B-05 \n Hritvik Saigaonkar B-19 \n\n Special credit for helping:\n Samay Pandey"
   android:textSize="16sp"
   android:gravity="center"
   android:padding="8dp"
   tools:ignore="HardcodedText" />
```

```
<Space
    android:layout_width="match_parent"
    android:layout_height="16dp" /> <!-- Small gap -->
```

Finally, an `ImageButton` is included to display the game's logo:

```
<ImageButton
    android:id="@+id/logoButton"
    android:layout_width="20mm"
    android:layout_height="20mm"
    android:src="@drawable/logo"
    android:contentDescription="KC Logo"
    android:background="@null"
    android:layout_gravity="center"
    android:padding="8dp"
    android:scaleType="fitCenter"
    tools:ignore="HardcodedText,InOrMmUsage" /> <!-- Clickable logo button -->
</LinearLayout>
```

Explanation:

This XML starts with a `LinearLayout` element, similar to the previous code. The `orientation` is set to `"vertical"` to stack the child views vertically. The `padding` attribute adds some spacing around the content of the layout.

Next, there are two `TextView` elements for displaying the credits information:

The first `TextView` displays the title "Credits." The attributes `textSize`, `textStyle`, `gravity`, and `padding` are used to style and format the text. The `tools:ignore` attribute is used to suppress lint warnings about hardcoded text during development.

The second `TextView` displays the detailed credits information. It lists the individuals who contributed to the development, coding, and debugging of the game. The format includes names and special credits. The `tools:ignore` attribute is again used to suppress lint warnings about hardcoded text.

A `Space` element is added to create a small gap between the text and the logo

The `ImageButton` uses an `id` to identify it. The `layout_width` and `layout_height` are set using millimeters (`mm`) to specify the size of the button. The `src` attribute points to the drawable resource for the logo. The `contentDescription` provides an accessible description of the logo. The `background` attribute is set to `@null` to remove any background, making it appear like an image. The `layout_gravity` centers the button horizontally within the layout. The `padding` adds some spacing around the button. The `scaleType` is set to `"fitCenter"` to ensure the logo fits within the button without distortion. The `tools:ignore` attribute suppresses lint warnings about hardcoded text and usage of millimeters during development.

Summary

This XML layout defines a custom dialog for displaying the credits section of the Android game. It includes a title, a detailed list of contributors, a small gap, and an image button displaying the game's logo. The layout is designed to provide a clear and visually appealing presentation of the game's credits, along with a clickable logo for additional interactivity.