# Tic-Tech-Toe

A Project Report
by
Sang Luong

Group: Javachip

University of Colorado Denver
Dr.Samil Lakhani
CSCI 3810 001

November 18, 2023

# Project Description:

Overview:
"Tic-Tech-Toe" is an innovative and interactive mobile application that reimagines the classic game of Tic-Tac-Toe. This project makes use of Android development frameworks to create a user-friendly and engaging gaming experience. Through the develomplemnt of this project, it integrates various features such as multiplayer options, game history tracking, music integration making it a versatile and entertaining version of Tic-Tac-Toe.

Key Features:

Multiplayer Gameplay: The core of Tic-Tech-Toe is its multiplayer functionality. The 'AddPlayers.java' module allows users to enter player names, creating a personalized gaming experience. This feature enhances the traditional game, allowing for a more social and interactive play. The second main feature is the addition of AI. There is two bots modes that are available, that being an easy bot and a hard bot. The hard bot uses the Minimax algorithm with basically makes the bot unbeatable. AI bots that can be selected will allow players to have a more competitive and more available gameplay as if the players does not have a second real life player.

Game History Management: 'GameHistory.java' and 'HistoryActivity.java' work together to provide a comprehensive history management system. There is a recycler view created to do the layout in which games are displayed. This implemented feature records each game played, enabling players to view their past matches.

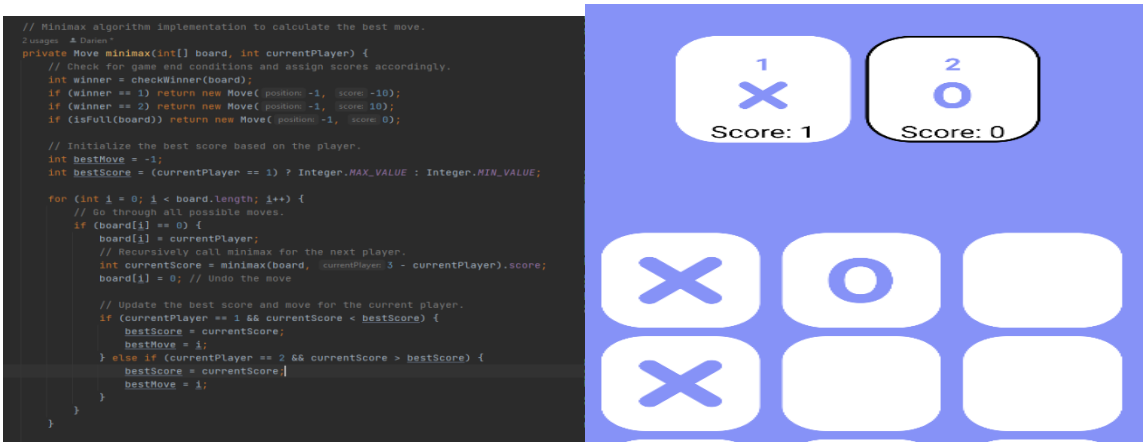
Interactive User Interface: The 'StartingScreen.java' and 'AddPlayers.java' modules lay the foundation for an intuitive user interface. With easy navigation and responsive design, the app ensures a seamless user experience from the start screen to the gameplay. Also, allowing players multiple options for customizations. The main screen has a menu button which is listed as “Settings.” It give the polayer navigation to the game history, about page, and sound settings.

In-Game Music Service: 'MusicService.java' and ‘SoudEffects.java’ adds an immersive element to the game by integrating background music and soud effects into the app. This service enhances the gaming atmosphere and can be toggled on or off as per the user's preference.

Result Display and Navigation: 'ResultDialog.java' displays the outcome of each game in a clear and concise manner. It also provides options to navigate back to the main menu or replay, ensuring a smooth transition between games.

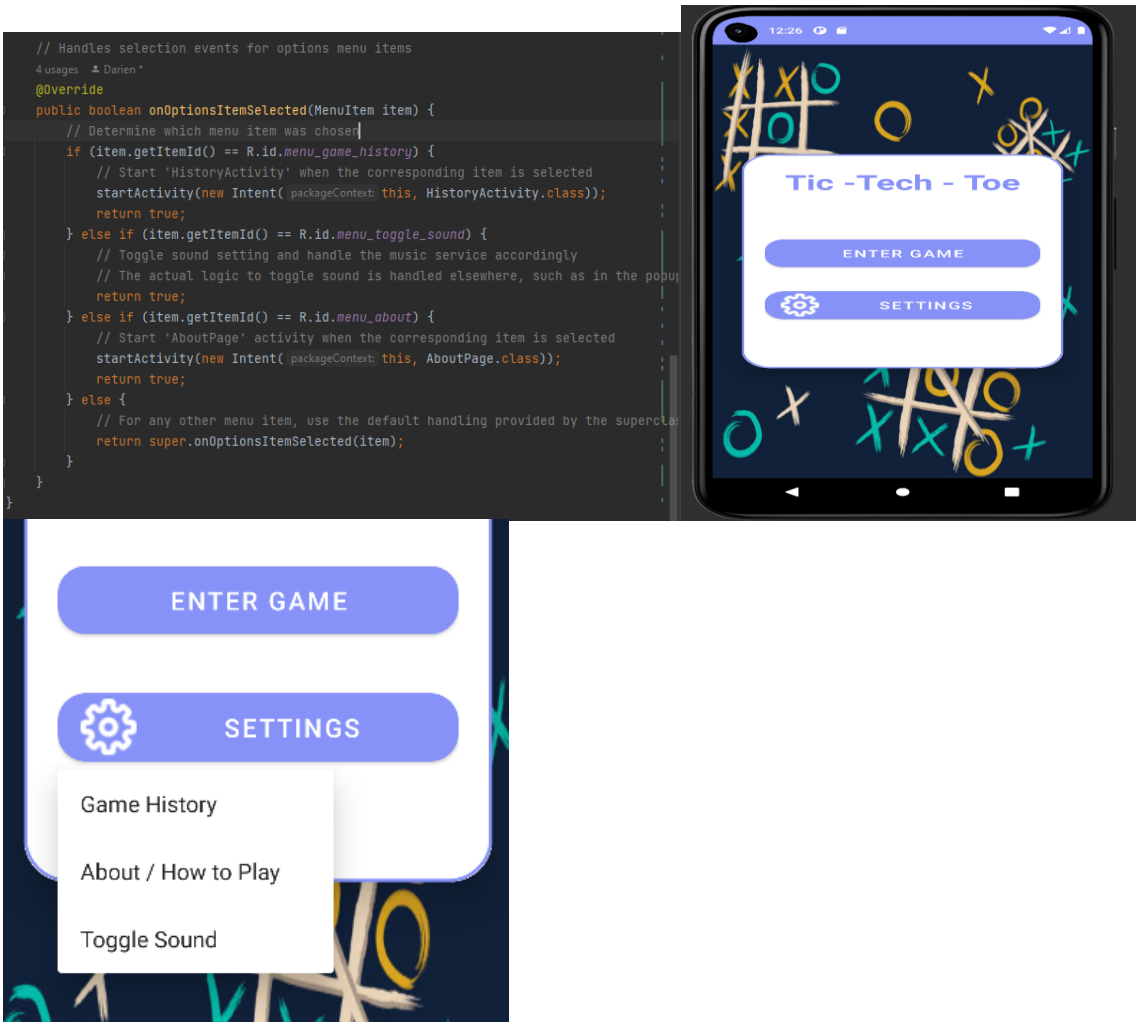
## Detailed Technical Descriptions

MainActivity.java is the central hub of the game, handling the core Tic-Tac-Toe gameplay logic. The first part I would like to mention is I did try setting a loop for the onClickListeners for binding the images on the grid. Though when I did my loop it broke my game, so I went back to the original code which is less clean but it is functioning. But the main functions to look at is performAction which will did all the main function to play the Tic-Tac-Toe game. It will update player's score, check for board for winning combinations, and save the game if the player's score reaches 3.This function also will handel the bot's movement and turn, based on the boolean value that is sent from AddPlayer.java. To explain the Minimax algorithm, private Move minimax(int[] board, int currentPlayer): This method calculates the best move for the current player given the current state of the board. board is an array representing the game board, and currentPlayer indicates who is making the move. checkWinner will check if the game has reached a terminal state (win, lose, or draw). bestScore is initialized differently based on the current player. for player 1, it starts with Integer.MAX_VALUE and for Player 2 with Integer.MIN_VALUE. This initialization supports the optimization of minimizing/maximizing the score based on the player. For each possible move, it simulates the move by setting board[i] = currentPlayer, then recursively calls minimax, then returns the 'bestMove.'




To continue on explaining some more codes, there are two functions one for resetting the board and one for resetting the game.

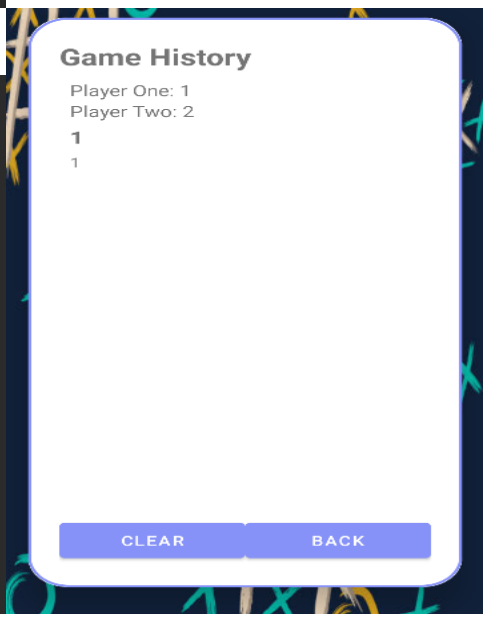
StartingScreen.java serves as the entry point of the app, offering navigation to game settings and the start of a new game. These settings leads to the game history, about page, and the option to toggle on and off sounds.The image belows shows the option handling for the menu.




HistoryActivity.java is dedicated to displaying the game history, utilizing a RecyclerView to present past matches in an organized manner. The image that is being provided is displaying the code that manages clearing the game history. As of now, the game only has the option to clear all game history, but I am now thinking that it might be better to allows players the options to delete history based on where the id is on the RecyclerView, but that can be added into later implementation of the game. GameHistory.java and Match.java are pivotal in storing and retrieving game history data. SharedPreferences is used for persisting user preferences and game history, ensuring data retention across sessions.

```java
// Activity class for displaying game history using a RecyclerView
4 usages  Darien *
public class HistoryActivity extends AppCompatActivity {

    // RecyclerView for displaying the match history list
    3 usages
    private RecyclerView recyclerView;
    // Adapter for the RecyclerView to bind Match data to the view
    3 usages
    private MatchAdapter adapter;
    // List to store the history of past matches
    3 usages
    private List<Match> pastMatches; // Declare here as an instance variable

    Darien *
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Set the content view to the layout defined in 'activity_history.xml'
        setContentView(R.layout.activity_history);

        // Initialize the 'Clear History' button
        Button clearHistoryButton = findViewById(R.id.clearButton);
        // Initialize the 'Back' button
```

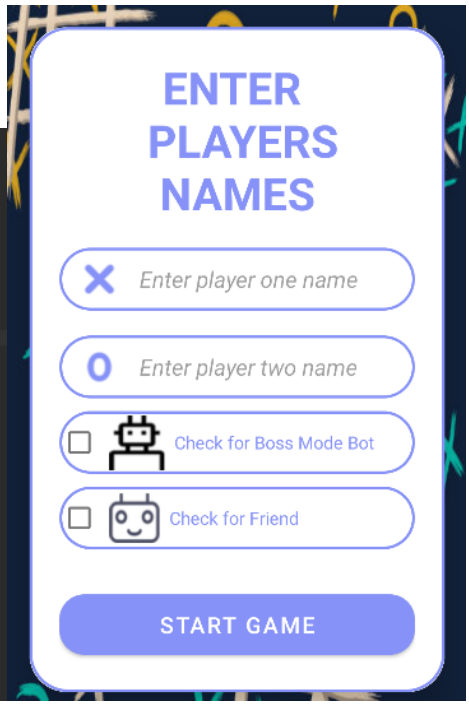
```java
Darien *
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Set the content view to the layout defined in 'activity_history.xml'
    setContentView(R.layout.activity_history);

    // Initialize the 'Clear History' button
    Button clearHistoryButton = findViewById(R.id.clearButton);
    // Initialize the 'Back' button
    Button backButton = findViewById(R.id.backButton);

    // Set an OnClickListener to clear the game history when the clear button is clicked
    Darien *
    clearHistoryButton.setOnClickListener(new View.OnClickListener() {
        Darien *
        @Override
        public void onClick(View v) {
            // Create an instance of GameHistory to interact with SharedPreferences
            GameHistory gameHistory = new GameHistory(getApplicationContext());
            // Clear the stored match history
            gameHistory.clearHistory();

            // Clear the current list of past matches and notify the adapter of the change
            pastMatches.clear();
            adapter.notifyDataSetChanged();
        }
    });
```




AddPlayers.java provides the functionality to enter player names, enhancing the multiplayer experience. It also provides the funcationalitly to add in bots as a stand-in for the second player. Here though, the player's get two options of the bots where one is easy and one is hard. The image provided below displays the checks for the boxes. For example, if the easy bot is selected you cannot select the other bot or enter in player 2's

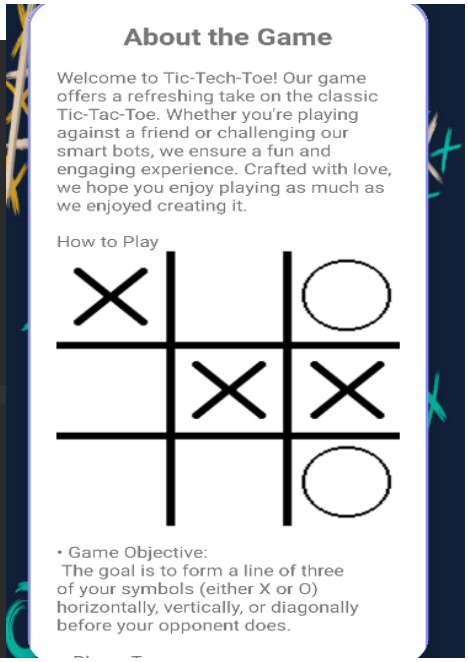
name.

```
// Define EditText fields for player names and CheckBoxes for game options
EditText playerOne = findViewById(R.id.playerOne);
EditText playerTwo = findViewById(R.id.playerTwo);
CheckBox botCheckBox = findViewById(R.id.botCheckBox);
CheckBox isEasyMode = findViewById(R.id.isEasyMode);
Button startGameButton = findViewById(R.id.startGameButton);

// Set listener for isEasyMode checkbox to disable playerTwo input and botCheckBox if easy mode is selected
isEasyMode.setOnCheckedChangeListener((buttonView, isChecked) -> {
    if (isChecked) {
        // Disable input fields for playerTwo and botCheckBox when Easy Mode is on
        botCheckBox.setEnabled(false);
        playerTwo.setEnabled(false);
    } else {
        // Enable input fields for playerTwo and botCheckBox when Easy Mode is off
        botCheckBox.setEnabled(true);
        playerTwo.setEnabled(true);
        playerTwo.setText(""); // Clear text field when Easy Mode is deselected
    }
});

// Set listener for botCheckBox to disable playerTwo input and isEasyMode if bot is selected
botCheckBox.setOnCheckedChangeListener((buttonView, isChecked) -> {
    if (isChecked) {
        // Disable isEasyMode CheckBox and playerTwo input field when Bot is selected
        isEasyMode.setEnabled(false);
        isEasyMode.setChecked(false); // Uncheck isEasyMode when Bot is selected
        playerTwo.setEnabled(false);
    } else {
        // Enable isEasyMode CheckBox and playerTwo input field when Bot is not selected
        isEasyMode.setEnabled(true);
        playerTwo.setEnabled(true);
        playerTwo.setText(""); // Clear text field when Bot is deselected
    }
});
```



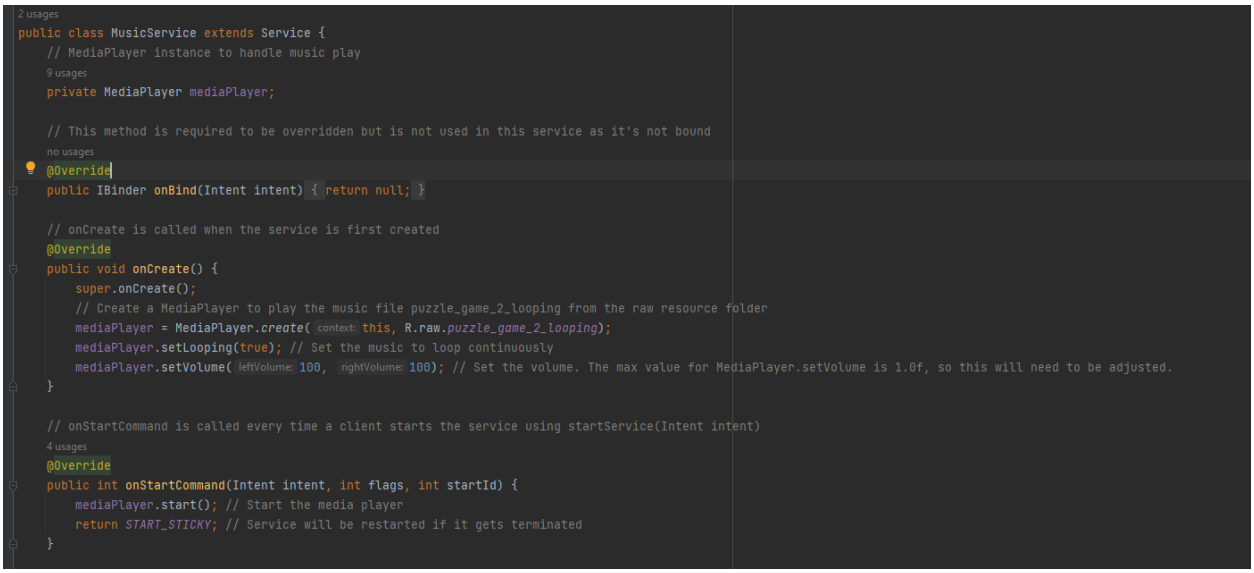

AboutPage.java is used to display game information to players. The about page includes descriptions of the the game, as well as instructions on how to play the game. The interesting new thing I had to input into this code was the use of Glide to display a gif asset I wanted to use to display how tic-tac-toe works.

```
// AboutPage Activity to display the about page of the Tic-Tech-Toe game
4 usages
public class AboutPage extends AppCompatActivity {

    // Declaration of the back button
    2 usages
    private Button backButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Set the content of the activity to use the activity_about_page.xml layout file
        setContentView(R.layout.activity_about_page);

        // Find the ImageView from the layout
        ImageView gifImageView = findViewById(R.id.animatedGifImageView);
        // Glide to load a GIF into the ImageView. The R.drawable.ttt_animated is a resource id for gif
        Glide.with( activity: this).asGif().load(R.drawable.ttt_animated).into(gifImageView);

        // Initialize the back button from the layout
        backButton = findViewById(R.id.backButton);

        // Set an onClickListener on the back button to respond to user clicks
        backButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Close the AboutPage activity and return to the previous activity in the stack
                finish();
            }
        });
    }
}
```




Music and Audio Features:

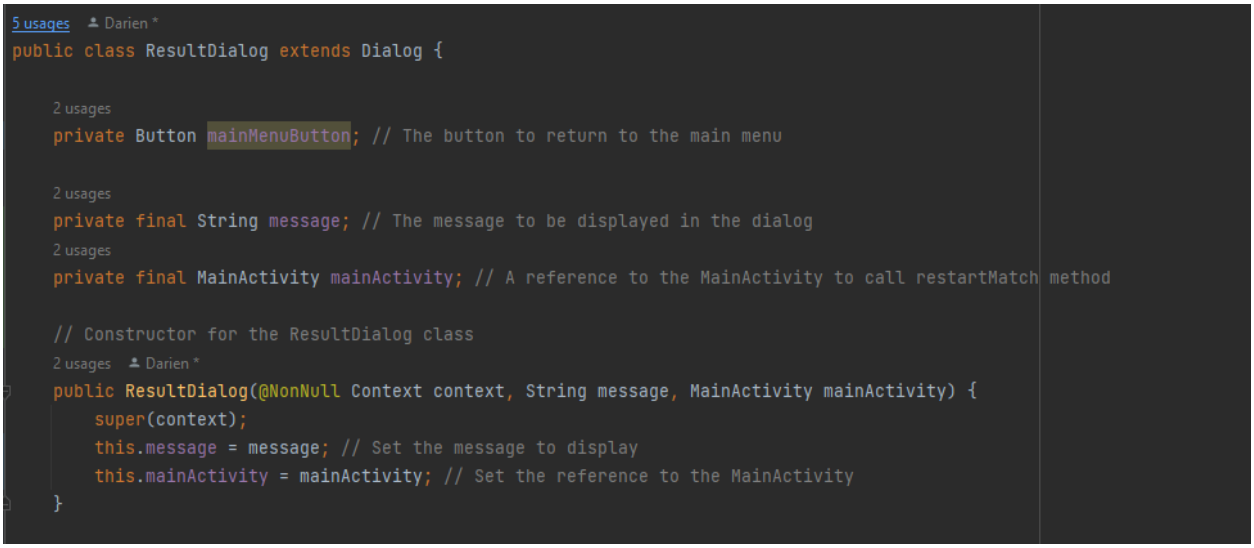
MusicService.java and SoundEffects.java implements a background music service, sound effects on clicks, enhancing the game's ambiance. It uses MediaPlayer for audio playback.



```
2 usages
public class MusicService extends Service {
    // MediaPlayer instance to handle music play
    9 usages
    private MediaPlayer mediaPlayer;

    // This method is required to be overridden but is not used in this service as it's not bound
    no usages
    @Override
    public IBinder onBind(Intent intent) { return null; }

    // onCreate is called when the service is first created
    @Override
    public void onCreate() {
        super.onCreate();
        // Create a MediaPlayer to play the music file puzzle_game_2_looping from the raw resource folder
        mediaPlayer = MediaPlayer.create( context: this, R.raw.puzzle_game_2_looping);
        mediaPlayer.setLooping(true); // Set the music to loop continuously
        mediaPlayer.setVolume( leftVolume: 100,  rightVolume: 100); // Set the volume. The max value for MediaPlayer.setVolume is 1.0f, so this will need to be adjusted.
    }

    // onStartCommand is called every time a client starts the service using startService(Intent intent)
    4 usages
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        mediaPlayer.start(); // Start the media player
        return START_STICKY; // Service will be restarted if it gets terminated
    }
```

Custom Dialogs and User Feedback:

ResultDialog.java presents the game outcome in a dialog, offering a user-friendly way to display results and navigate post-game options.



```
5 usages   Darien *
public class ResultDialog extends Dialog {

    2 usages
    private Button mainMenuButton; // The button to return to the main menu

    2 usages
    private final String message; // The message to be displayed in the dialog
    2 usages
    private final MainActivity mainActivity; // A reference to the MainActivity to call restartMatch method

    // Constructor for the ResultDialog class
    2 usages   Darien *
    public ResultDialog(@NonNull Context context, String message, MainActivity mainActivity) {
        super(context);
        this.message = message; // Set the message to display
        this.mainActivity = mainActivity; // Set the reference to the MainActivity
    }
```


Instructions (To Start A Game):
1. When app opens, you will be on Starting Screen. Here you have the option to view game history, toggle sounds, go to an about page, and start a game.

2. You would click Start and be lead to a Add Player screen where you would get the option to select your opponent. You must input a name for player 1 which is you and select from the wide range of options. (Note: You cannot select a bot and enter a player name, and vice-versa. Either deselect the bot or remove the player's name to change options)

3. Once done you can press start game to begin the match.

4. Depending on your options you will always go first, but player 2's play would be dependent on the options of player you selected. To play though, you would click on one of the white squares that have not been occupied by a X or O symbol. This step will be repeated by PLayer 1 and 2 till the board is filled up.

5. Once a player's score reaches 3 points, a custom dialog box will pop up announcing the winner of the game and then saving the match to the game history. The dialog will offer the options to return to the starting screen, or to restart the game with the same settings.