Team members

- Austin Anker (Team Lead)
- Qinqi Wang
- Ray Freeze
- Alif Maknojia

Purpose

The purpose of this project is to create a tic-tac-toe game for the Android platform that pits the user against an AI based on the minimax and A* algorithms.

Entities

1. GUI

The purpose of the GUI is to provide an aesthetically pleasing graphic that the user interfaces with when he or she plays our game. The GUI also allows our program to run on any android device or android emulator. The user will be able to choose the AI type and difficulty and start new games.

2. Minimax Algorithm

The minimax algorithm will plot out all possible positions (to a certain depth) that the AI could choose based on the decisions from the user. The algorithm will then use alpha-beta pruning to delete the positions that neither the user or the AI would choose. The resulting tree contains the "best" placements that would leave to an AI victory.

3. A* Algorithm

The A* algorithm will picks the shortest path to reach the destination node. It will pick the best position available for a move depending on the users move. It calculates the number of moves required for win by adding initial state to the number of state to reach the destination node. It basically uses the Dijkstra's algorithm to get the final node or the best possible move.

4. AI Extension

The AI Extension will allow the AI to run the minimax and A* searches while the player is coming up with a move. This will allow for the AI to make better decisions, since they are given more time to come up with an optimal move.

Low Level Design

1. GUI

- Usage
    - The GUI is used as an interface between the user and the program. It will allow the user to play our game with touch or mouse control. The GUI stores the game state internally and shows a graphic representation to the user. The internal state can be sent to separate algorithms in order to determine the next best move.
- Configuration
    - The GUI will first be set up to display the menus and empty game board. The GUI will then need to store information about each square of our board and update the graphics so that the correct stage in the game is displayed. The GUI will need to be able to interface with the C++ code of the other group members.
    - The Graphics will be codded in XML using the drag and drop functionality of Android Studio.
    - The Graphics will be given functionality with java.
    - The next best move calculation will be handled by an outside algorithm that is passed from our java program to a C++ program.

2. Minimax Algorithm

- Usage
    - The minimax algorithm will request the current board setup from the GUI in order to create a decision tree. It will then check the rows, columns, and diagonals for a win. If not, it will choose the best position for the AI and return the updated board setup to the GUI.
- Configuration
    - For initialization, the algorithm will use the starting position of the AI as determined by the game difficulty, which is chosen by the user.

3. A* Algorithm

- Usage
    - The A* algorithm will request the current board setup from the GUI same as the minimax algorithm. It first start with X in a node depending on the difficulty level the user picks. Once user makes his move it checks and see for a win, if not then it chooses the best move for the AI to either block the users win or keep putting X to make five in row, column or diagonally. After one move it returns back the board to GUI for the user's move.
- Configuration
    - It randomly puts X in a position depending the difficulty chosen. Then when the user makes his move it checks and see if user is not winning, go for five in row X. It basically see's which node is empty around the parents node. If its empty node then it puts it into the open list else in the close list. AI keeps doing this till it wins or it ties.
- Model

4. AI Extension

- Usage
    - The AI extension will run a separate thread while it is the player's turn. Once the player makes

    a decision, it will find the subtree that stems from the user's move and continue searching that subtree. After the AI reaches it's time limit, it will make a move and then run the search again as the player thinks on the AI's move. As shown in the diagram above, the decision tree will go through every possible player move first until the player makes a decision. Running the AI algorithms while the player is coming up with a move should improve the performance of the AI, but this only improves performance assuming the player takes time coming up with a move. If the player acts quickly, the improvement to AI ability will be minimal.

- Configuration
    - The AI extension for the Minimax algorithm will be different than that of the A* algorithm because they search for optimal moves with different methods. The Minimax algorithm uses a decision tree while the A* algorithm uses a mathematical function.

Interaction

Benefits, Risks, Assumptions

Benefits

- Developing for Android means that our program will work across a wide range of devices.
- Our program will be physically portable on small smartphones.
- Using the algorithms makes the process more efficient and more consistent.
- The AI extension will give the AI more time to make decisions

Risks

- Developing for Android means that we might run into issues that are only present on certain devices or emulators.

Assumptions

- We assume that if our program works properly on our emulator that it will work on any Android device.
- We assume that the player will take time making a move, giving the AI time to run it's algorithm