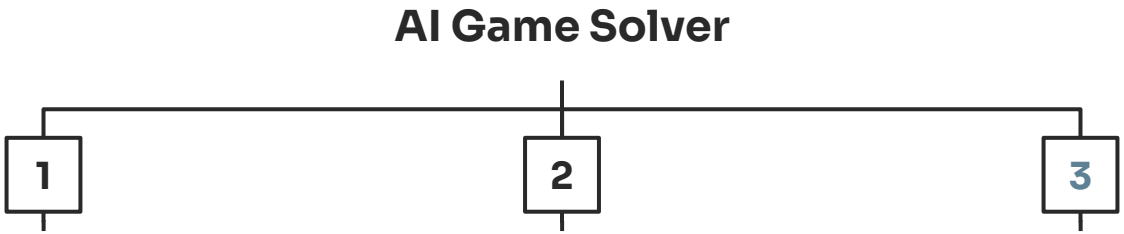
Tic-Tac-Toe AI Solver

Rida Khan, George Mills, Samirah Huda,
Kyle Deng, David George

Initial Goals



Maze Solver

A. Uninformed/informed search methods, inspired by the one of the provided project ideas
B. Had implementation of A* search from before
C. But wasn't that interesting

Chess

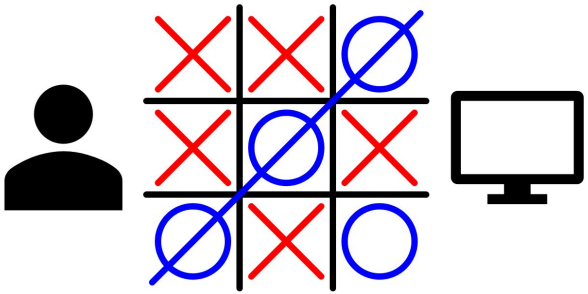
A. Minimax algorithm
B. We all liked the idea
C. But very complex because of the large number of possible game states and piece movements

Tic-Tac-Toe

A. Minimax algorithm
B. Relatively simple compared to chess
C. Manageable number of game states and player actions

Objectives

- We aimed to build an AI agent that is able to play Tic-Tac-Toe and win against the opponent
- Our AI agent will use the minimax algorithm to determine the optimal actions to take to win the game
- Our final product will be presented in a GUI
- The user will be able to select X or O, choose the level of difficulty that the AI agent will play by, compete against the agent, and the winner will be displayed on the screen



Methodology: The Minimax Algorithm

- Used minimax to model and select maximal board states
- Created a score function that would return 100 (X winning final board), 0 (a drawing final board), and -100 (O winning final board)
- Adjusted for depth so that solutions that achieve a winning state in less turns are preferred
- Generated every possible board state from a given board state on every minimax iteration because it's computationally feasible for 3x3 Tic-Tac-Toe
- Informed by *Analysis of Game Tree Search Algorithms Using Minimax Algorithm and Alpha-Beta Pruning* by Shevtekar, Malpe, and Bhaila

Solved Issues

- Traditional minimax implementations is a tree of values, but we needed to return a boardspace and its value
- ➢ Generate list of possible immediate moves and corresponding boards, run minimax on each board, compile terminal values in a list, pick min/max, use that index to return future move from list
- Original implementation of minimax resulted in too many optimal states because each of them could eventually result in the winning terminal state
- ➢ Make values of win/loss terminal states 100/-100, keep track of depth in minimax, subtract/add final value by depth to prioritize solutions that end the game faster

Methodology:
The Minimax Algorithm for 4x4

- 3x3 algorithm can be used on a 4x4 board without modification
- However, without optimizations, the time it takes to run is too long
- Tested potential optimizations using various combinations of depth limits and heuristics
- Went with a heuristic that randomly chose an adjacent tile to a previously placed letter
- Best solution used this heuristic on the first 3 turns, then traditional minimax to determine the best moves thereafter
- Depth limits appear to be detrimental

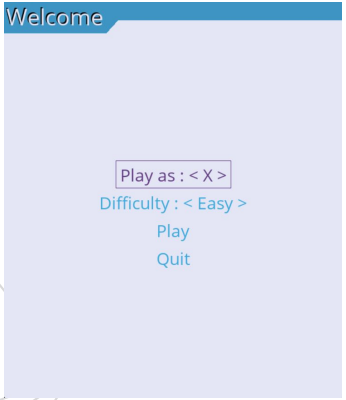
Outcomes

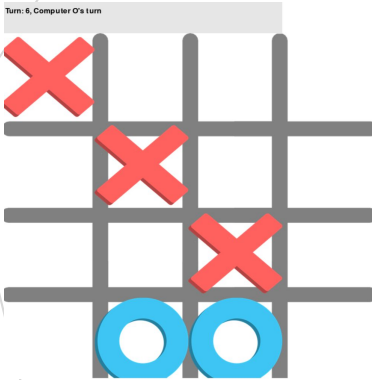
- We were able to build an AI agent that is able to play Tic-Tac-Toe optimally (win or tie) on a 3x3 board using the minimax algorithm
- The user is able to interact with a UI that enables them to choose a letter, the level of difficulty, and compete against the AI agent
- When a player wins, the game ends and the winner is displayed on the screen
- Created another game mode for a 4x4 board in order to better explore the minimax algorithm
    - 4x4 board presented the challenge of balancing depth/breadth limitation for time efficiency and strategy

Areas for Improvement

- Accounting for players who do not make optimal moves and make the game more interesting by using expectimax over minimax
- While efficiency for 3x3 is not a concern on our hardware, alpha-beta pruning might make the game more computationally accessible to worse/older hardware
- Implementing more rigorous heuristics, more informed breadth/depth limits, and pruning to improve 4x4

Game Images





Quantitative Metrics