Final Project Report

Artificial Intelligence I (CSCI 5511)

Submitted by:
Alex Besch (BESCH040)
Mohit Yadav (YADAV171)
Dec 19, 2023

Introduction:

For this project we developed the game Ultimate TicTacToe in python. Ultimate Tic-Tac-Toe (UTTT) is an advanced iteration of the traditional Tic-Tac-Toe game, introducing a more complex and strategically challenging environment due to its expanded search space. In Ultimate Tic-Tac-Toe, the game board comprises nine smaller Tic-Tac-Toe boards, and player moves determine which sub-board their opponent must play next, adding an additional layer of strategy compared to the classic game.

As this is not an official game, exact rules are not available. However, some common accepted rules are:

- There is a 3x3 master grid, each tile containing another sub 3x3 grid.
- Players take turns marking an X or an O to designate their move.
- Successor players must play in the tile of the master grid that their opponent played in the sub grid. If the master grid tile is full or complete, or it is the first move of the game, the player can play in any open tile of their choosing.
- The game is played until someone wins traditional Tic-Tac-Toe on the master grid.

Additional rules for tie in a sub-grid and for fully occupied sub-grid are considered similar to as done by Subrahmanya[1].

We have implemented the game from scratch in python. Players can play as human, random, and minimax agents, enabling gameplay between human-human, human-AI, or AI-AI. The minimax agent is enhanced with alpha-beta pruning to enable the code to look ahead up to 6 steps within a reasonable time, approximately around 7 seconds per move.

Heuristics:

For assessing the fitness of a state we came up with two heuristics for the gameplay which are described below:

Heuristic 1 : We assigned a importance score to every place on the board with our understanding of tic tac toe and accordingly gave the score to the players. The score distribution was as follows:

a. Acquiring the center spot on any of the miniboards gives you 5 points.*
b. Acquiring the corner spot on any of the miniboards gives you 4 points.*
c. Acquiring the edge spot on any of the miniboards gives you 3 points.*
d. Acquiring the center spot on the master board gives you 500 points.*
e. Acquiring the corner spot on the master board gives you 400 points.*
f. Acquiring the edge spot on the master board gives you 300 points.*

g. Winning the game is worth 100,000 points.
h. Losing the game is worth -100,000 points.
i. Tie is worth -1000 score.

*Note : This is a symmetric heuristic i.e., negative of the same score was awarded if your opponent does this.

Heuristic 2 : For this heuristics we modified the strategy to also give importance to a spot based on the items already present there i.e., if we have two spots occupied in a winnable configuration(all the sequences where you can occupy three places to win) we give it more score and similarly for three spots occupied in winnable configuration, where winnable configuration means the stretch of three continuous spots which can lead to winning a game. The score distribution for this heuristics are given below:

a. Acquiring a spot in a winnable configuration on the mini board with all other spots empty gives you 1 point.*
b. Acquiring two spots in a winnable configuration on the mini board with the third spot being empty gives you 10 points.*
c. Acquiring all three spots in a winnable configuration on the mini board gives you 100 points.*
d. Acquiring a spot in a winnable configuration on the master board with all other spots empty gives you 100 points.*
e. Acquiring two spots in a winnable configuration on the master board with the third spot being empty gives you 1000 points.*
f. Acquiring all three spots in a winnable configuration on the master board gives you 100,000 points.*

*Note : This is a symmetric heuristic i.e., negative of the same score was awarded if your opponent does this.

With these heuristics to arm our alpha beta pruning Minimax we simulated 50 games with various agents namely Random, Heuristic 1, Pulkit Mittal's[2] Heuristic and Heuristic 2. In the game of ultimate tic-tac-toe there is a slight advantage to the player who goes first[1], therefore we played 25 games and then switched players to remove the extra variable of a player going first. The results obtained by the simulation of games are shown and discussed in the next section.

Results:

When both agents playing the game are deterministic, meaning they consistently make the same move from a given state, the game always ends similarly. To introduce randomness into the game, we made each agent's first move random (except human player). This made each game different, and tested the agent's ability to deal with unideal conditions. We made the agents play against each other. The results of the playoffs are discussed next.

Game 1. Between “Random Player” and Alpha-Beta with “Heuristic 1”

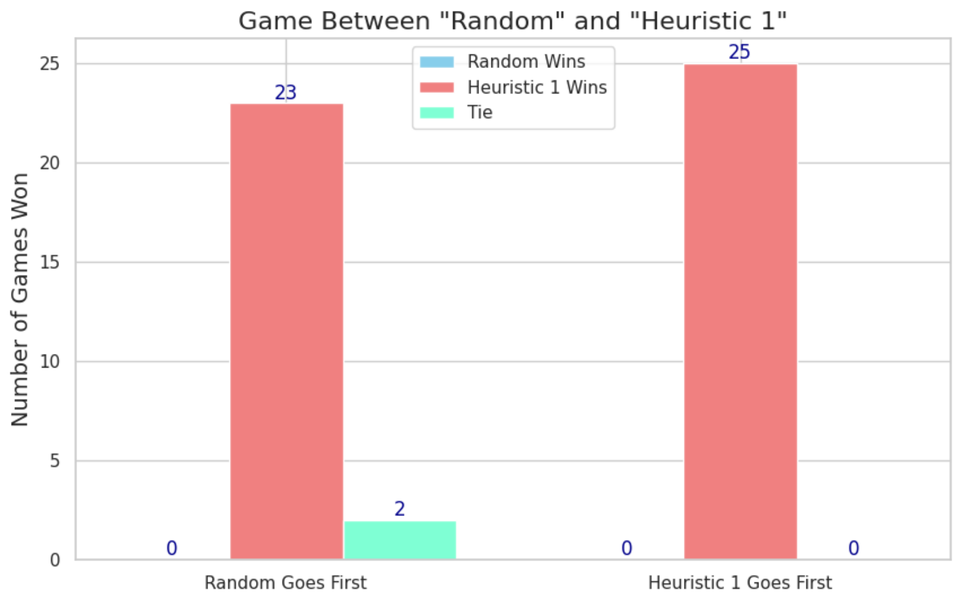


Figure 1: Results of Heuristic 1 vs Random playing agent.

Figure 1 shows the result of playing 50 games using our basic Heuristic 1 against a random agent. This was mostly a sanity check to ensure game mechanics were working properly. Figure 1 shows that heuristic 1 was far better than random, winning 48/50 games while 2 ended in a tie. Random was not able to win any game in these playouts.

Game 2. Between Alpha-Beta with “Heuristic 1” and Alpha-Beta with “Pulkit’s Heuristic”

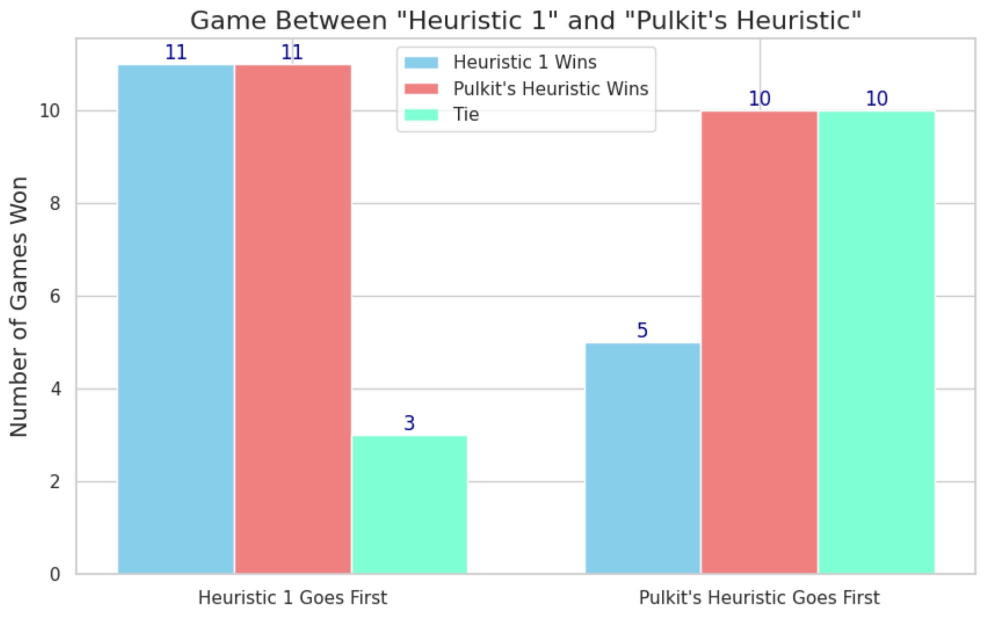


Figure 2: Results of Heuristic 1 vs Pulkit’s heuristic.

Figure 2 shows the result of playout between Heuristic 1 and Pulkit’s heuristics. Pulkit’s heuristic won 21/50 games, with 13/50 games ending in a tie and Heuristic 1 winning 16/50 games. Results showed that Pulkit’s method was marginally better than our heuristic 1.

Game 3. Between Alpha-Beta with “Heuristic 2” and Alpha-Beta with “Pulkit’s Heuristic”

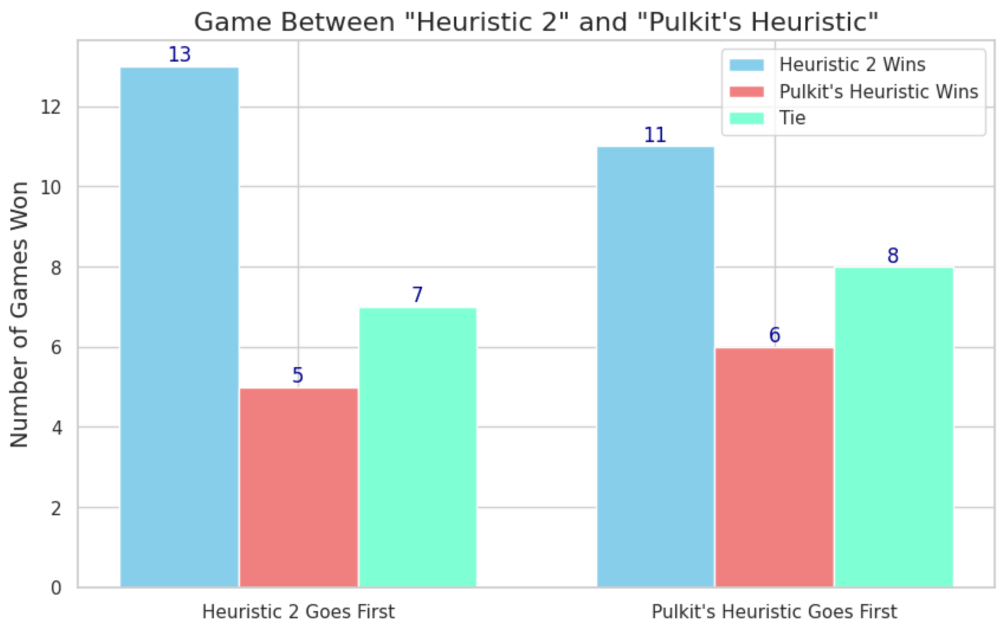


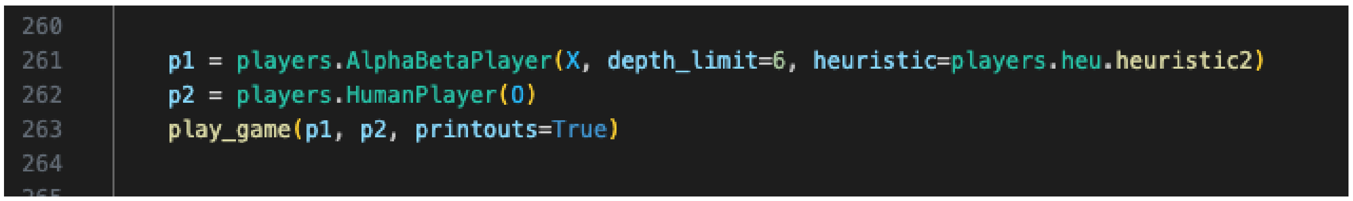
Figure 3: Results of Alpha-Beta with Heuristic 2 vs Pulkit’s heuristic.

The purpose of this project was to build a heuristic better than something publicly available. As our heuristic 1 proved worse than Pulkit's code we came up with heuristic 2 which took inspiration from Pulkit’s heuristics. We improved it by giving weightage to acquiring positions on the master board too. The results of Heuristic 2 playing against Pulkit’s heuristic is shown in figure 3. Heuristic 2 proved far better than Pulkit’s heuristic, winning 24/50 games, with 15/50 ending in a tie while Pulkit’s heuristic won 11/50 times.

Guideline to run the code:

Our project was completely made on python3 and therefore can run on terminal or any IDE of your choice. To run the program you need additional Numpy python packages installed in your environment. Once this is installed, code can be run by simply calling the python script gameuttt.py with players.py and heuristics.py files stored in the same directory as gameuttt.py.

By default the code will play against a human player using the Alpha Beta agent with depth of 6 and Heuristics 2. This can be changed in the main() function of gameuttt.py where you can adjust the agents playing the game and other parameters of the agent.



```
260
261     p1 = players.AlphaBetaPlayer(X, depth_limit=6, heuristic=players.heu.heuristic2)
262     p2 = players.HumanPlayer(O)
263     play_game(p1, p2, printouts=True)
264
```

Figure 4. Snippet of code inside main() where you can declare players. (line no. 261,262 in gameuttt.py)

Following agents are available for playing the game: RandomPlayer, MinimaxPlayer, HumanPlayer and AlphaBetaPlayer.

RandomPlayer and HumanPlayer take one positional argument as the player i.e., “X” or “O”. Whereas MinimaxPlayer and AlphaBetaPlayer take two extra named arguments of depth(default : 4) and heuristic(default : heuristic_2). Available heuristics for agents are heuristic_1, pulkit_github and heuristic_2.

You may also run multiple games between AI agents, similar to the tests run for this report. This can be found in the commented out section of main(). See readme file for further information.

Conclusion:

We observed that the heuristics, which prioritize winning tic-tac-toe on the master board along with giving priority to winning on mini boards, prove to be very effective for the game. This intuitively aligns with how a human would think about winning the game. The project highlighted the challenges of creating an artificial agent for games with vast state spaces and high branching factors. However, the project also demonstrated the computational power of computers in making highly informed decisions by considering thousands of states in mere seconds. This capability enabled us to develop an agent that can outperform most human players in the game of Ultimate Tic-Tac-Toe.

GitHub Repository:

As we needed to collaborate with team members on this project, we used GitHub for collaboration and version control. This project provided us with hands-on experience using tools like Git and GitHub. The GitHub repository link for the project is: https://github.com/TralexBeach/Ultimate-Tic-Tac-Toe. All the code for the project can be found in this repository.

References:

[1] Sis16. Subrahmanya Srivathsava Sista. Adversarial Game Playing Using Monte Carlo Tree Search. PhD thesis, University of Cincinnati, 2016.

[2] Maloo, P (2017) Ultimate-Tic-Tac-Toe (Version 1.0) [Source code]. https://github.com/pulkitmaloo/Ultimate-Tic-Tac-Toe