Cubic Tic-Tac-Toe Alpha Beta Search

Tyler Brinkley,[a] Susan Foster,[a] Alexander Neumyer,[a] Michael Tang,[a] Melinda Yan,[a] Julie Yang[a]

[a] University of Maryland, College Park, Professor James Reggia, CMSC421: Introduction to Artificial Intelligence

Abstract. The goal of our project is to apply the alpha-beta pruning variant of the minimax game tree search algorithm (shortened to alpha-beta search) and analyze its results for our innovative version of the classic game Tic-Tac-Toe. In this study, we designed and implemented a cubic version of Tic-Tac-Toe with a unique set of rules inspired by the Rubik's Cube. We modified an existing heuristic for Tic-Tac-Toe to be usable for our variant and created a new heuristic as well. Our hypothesis is as follows: if we use our own heuristic created for this game, then we will receive a higher win percentage and/or more wins achieved with fewer moves than if we use the adapted heuristic of the original game. This hypothesis is based on the unique rules of our variant and the fact that each side of the cube is its own Tic-Tac-Toe game.

1 Introduction

Our goal is to analyze the results of applying an alpha-beta search on a cubic version of Tic-Tac-Toe with its own unique set of rules (Appendix). Motivations for this project stem from the classic game of Tic-Tac-Toe being solved. Small variations of the classic game, such as using a 4x4 or 5x5 board, are already known to be solved because of the small solution space. However, using a 3D board (specifically 3x3x3) is vastly different from the classic game and thus, presents a curious opportunity to investigate if it is solvable. There exists a version of a 3D Tic-Tac-Toe game presented by having 3 horizontal layers of Tic-Tac-Toe boards but we investigated a different game board design, inspired by the Rubik's Cube.

We investigated whether our cubic version of Tic-Tac-Toe can be solved as the classic game of Tic-Tac-Toe is known to be solved. The game Tic-Tac-Toe has many existing admissible heuristics, with one well-known to be the most efficient. We modified this efficient heuristic to be usable for our 3D version of the game and tested whether the modified heuristic or a new heuristic we created would yield a higher percentage of wins and/or more efficient wins.

Our hypothesis is if we use our own heuristic created for this game we will receive a higher win percentage and/or more efficient wins (more wins achieved with fewer moves) because not only are the rules different, there are more to consider than just one Tic-Tac-Toe game.

In the event that we conclude that our version cannot be solved, we aim to find the heuristic that leads us to a higher possibility of winning. To compare the efficiency of the heuristics we have to know how fast the players can win the game or how many steps it takes to win. In implementing this game, we used a python environment for its ease of use and object-oriented nature. Python's dynamic nature also helps in the inevitable case of the alpha-beta search yielding potentially large integer values, and help avoid integer overflows. After we collected results from running several experiments, we discussed the pros and cons of each heuristic and came to a conclusion as to which heuristic was better.

2 Literature Review

Minimax algorithm is a backtracking algorithm used in decision making and game theory. The idea is to determine a player's optimal move, assuming the opponent moves optimally. For two players, MAX and MIN, the purpose of the algorithm is for MAX to optimize their chance of winning the game and for MIN to minimize the opponent's chances of winning. This method has been used to solve games, such as Tic-Tac-Te. In implementing the minimax algorithm, we first construct a game tree and apply an evaluation function to the leaf nodes. Then, we recurse up the tree, selecting the best move for each player type. For MAX, select the move with the highest score. For Min, select the move with the lowest score. Once at the root node, select the node with the highest value as its next move. In the original Tic-Tac-Toe, a common evaluation function or the heuristic is

$$E = (number\ of\ open\ wins\ for\ MAX) - (number\ of\ open\ wins\ for\ Min)$$

where MAX represents X and MIN represents O. If the move is a win for MAX, then $E = \infty$. If the move is a win for MIN, then $E = -\infty$.

Alpha-beta pruning is an optimized algorithm of the minimax algorithm. It seeks to reduce the number of nodes visited to find the best move possible for a player. Unlike the minimax algorithm, Alpha-beta pruning stops searching for MAX if the score for a move of MIN results in a value equal to or greater than the largest score (alpha) it has seen so far, so called creating an alpha cutoff. For MIN, the algorithm stops searching if the score for a move of MAX results in a value equal to or less than the smallest score (beta) it has seen so far, so called creating a beta cutoff. Thus, the algorithm reduces computational time and enables a faster search that possibly goes into deeper levels in the game tree. With alpha and beta cutoffs, the algorithm cuts off branches of the game tree that do not need to be searched if a better move exists.

In research from Zain et al. (2020), the paper outlined steps taken to develop standard 3x3 Tic-Tac-Toe games using artificial intelligence heuristics. The process was broken down into three parts: pre-production, production, and post-production. In the first phase, pre-production, researchers focused on designing a proper prototype for the game interface, including incorporation of rules and heuristics. This step acted as a brainstorming step to solidify ideas and gain a better understanding of how the game will be run. In the production stage, researchers implemented the game design and heuristics. Lastly, researchers performed post-production development to maintain and test their game design. Maintenance in this last step is crucial as it is necessary for programmers to recheck to code and fix any bugs produced after executing the game. Testing is also a major component as this is where programmers will work to optimize the code and increase efficiency of the game, in terms of runtime and memory. While this paper discusses how researchers develop a 3x3 Tic-Tac-Toe game, the same concepts and ideas of the game development process are of significance. This process can be viewed as an outline as we implement our version of Tic-Tac-Toe using heuristics.

In 2007, Cranenburgh and Smid developed Prolog code to simulate playing Tic-Tac-Toc of varying size and number of dimensions. They explore the use of minimax and heuristics by implementing three heuristics and varying search depth. Of interest is their results for the 3x3x3 game. The results show that the first player wins around 70% of games. When performing with their first heuristic, the first player wins 85% of games. When varying search depth, either the starting player or the one with a higher depth wins the games. While our version of 3x3x3 Tic-Tac-Toe involves a distinct set of heuristics and rules, this paper gives us insight into results to expect in terms of who wins, effectiveness of heuristics, and impact of search depth.

3 Methodology

The implementation of our study primarily consisted of three separate programs (1) the representation of the cubic Tic-Tac-Toe game board and rules, (2) a *random bot* that would make a move by randomly picking an open space, (3) a *minimax bot* that would utilize minimax pruning (alpha-beta search) with predetermined heuristics to make a move. Additionally, we also implemented a runner program to play and test the game as well as a shell script to expedite and automate experiments. All programs except for the shell script (bash) were written in python.

We applied two separate heuristics to the minimax bot. Our first (old) heuristic was derived directly from the original game of Tic-Tac-Toe where we decided to take the *sum of the scores calculated with the original heuristic on each side of the cube*. If a face is won by MAX, add 10. If a face is won by MIN, subtract 10. This best represents the original heuristic altered with our game because even though scores are calculated separately for each side, in an overall view of the entire game board, we are still counting the possible wins of MAX - the possible wins of MIN. Our second (new) heuristics was formed after some rule based study. We discovered some strategies we can use that are different from the original Tic-Tac-Toe game after playing a few games with the new game board. For example, in our game board setting, corners are very important, since in the beginning all of them represent three possible moves that can be taken with one move. However, we came to a conclusion that the best move still depends on whether a move results in more possible wins. Therefore, our new heuristic was formed with three categories: whether sides have been won, possible open wins with two in a row in place, and possible open wins with just one in a row in place. Our new heuristic function:

$$\begin{aligned} &50 * ([sides\ won\ by\ max\ -\ sides\ won\ by\ min]) \\ &\quad + 2 * (number\ of\ 2\ in\ a\ rows\ that\ max\ has\ -\ what\ min\ has) \\ &\quad + 1 * (number\ of\ 1\ in\ a\ rows\ max\ has\ -\ what\ min\ has) \end{aligned}$$

See Appendix for an example of both heuristics being applied.

In this study, we repeatedly had the random bot and the minimax bot compete in a game of cubic Tic-Tac-Toe. For each experiment, we evaluated the performance of the bots on various parameters depending on the experiment. In general, we considered how many games a bot won, the time it took for a bot to decide on a move, how many moves a bot won in, and how the depth of the search tree of the minimax bot affects the previously mentioned factors. Due to time constraints and computational limitations, only depths 3, 4, and 5 were examined.

4 Results

A total of 5 unique experiments were conducted, the first studied the win rate of the minimax bot versus the random bot at different depths and if player order significantly affected any bot's win rate. At depths 4 and 5, games were conducted 100 times total, 50 where the minimax bot played first and 50 where the random bot went first. Below shows the outcome:

Depth	Random Bot Wins	Minimax Bot Wins	Game ties	Win Rate (%)
4	0	100	0	100%
5	0	96	4	96%

Figure 1: Experiment 1 results for the minimax bot using our new heuristic against the random bot.

Regardless of which bot played first, the minimax bot consistently performed better than the random bot. Any ties that occurred could be due to the random bot picking the most optimal move by pure chance based off of how the bot was created. Finally, against a random player, minimax consistently outperformed regardless of play order.

The second experiment performed was testing two different heuristics against one another using the minimax alpha-beta pruning algorithm. The win rate and time it took for each heuristic and bot to make a move were recorded to identify any significant differences. The mean time to make a move and standard deviation were recorded across 100 games.

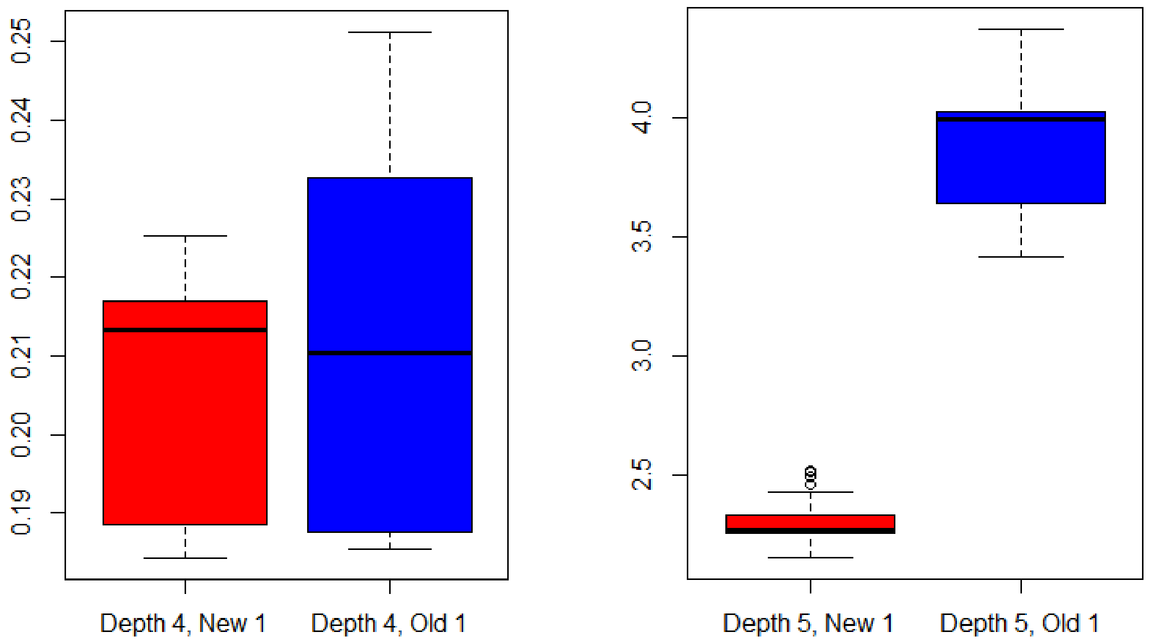


Figure 2: Experiment 2 results on average move times between the new and old heuristics for the minimax bot.

	MMBNew (first)	MMBNew (second)	MMBOld (first)	MMBOld (second)
Mean, Depth 4	0.217053889	0.193198873	0.233757569	0.187809667
SD, Depth 4	0.001459121	0.00801795	0.010026038	0.002112191
Mean, Depth 5	2.298743198	2.288271068	4.075548045	3.660910886
SD, Depth 5	0.086281265	0.05161641	0.097832005	0.015850025

Figure 2a: Experiment 2 results on average move times between the new and old heuristic for the minimax bot, and the standard deviation, along with how play order affected each.

The third experiment examined the average time to make a move over a game. The results are shown below

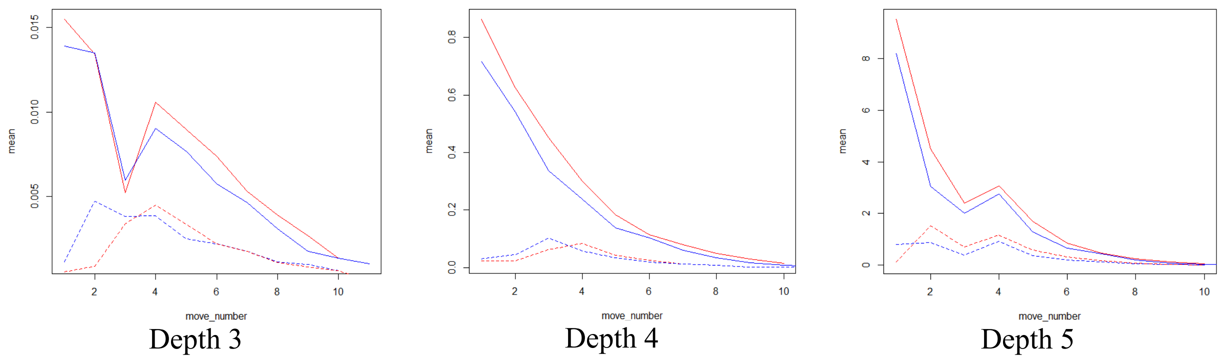
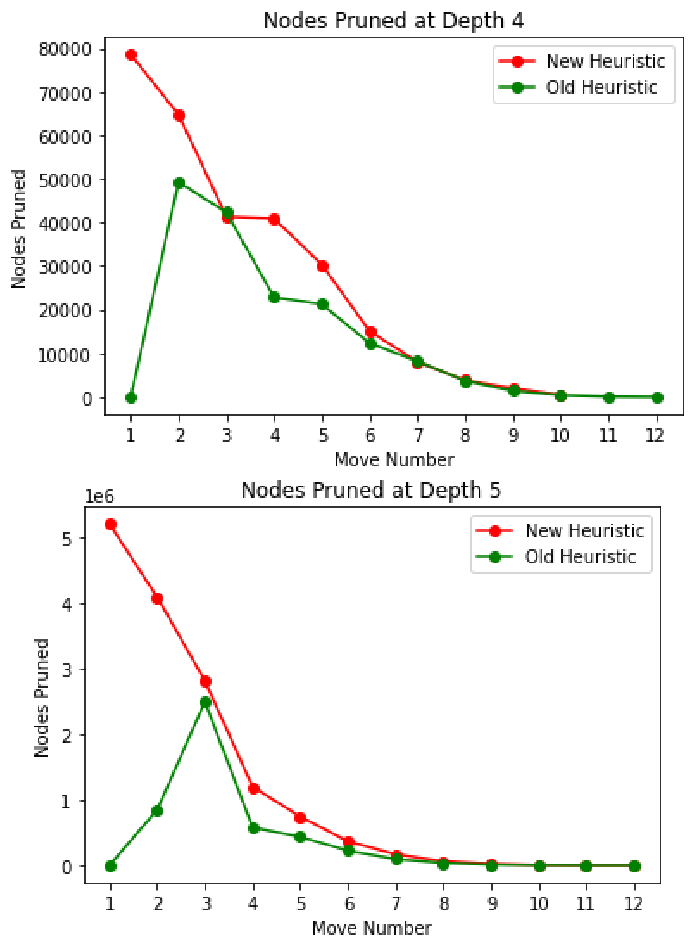


Figure 3: Experiment 3 results on average move time per move for the new heuristic from depths 3 to 5. The data for the minimax bot going first is shown in red, the data for the minimax bot going second is shown in blue, the solid lines are the mean move times, and the dotted lines are the standard deviations of the move times.

The time it takes the minimax bot to choose a move is larger when the minimax bot goes first. This could be due to the fact that the minimax bot has to consider more moves on the game board as there are more open spaces for the minimax algorithm to evaluate.

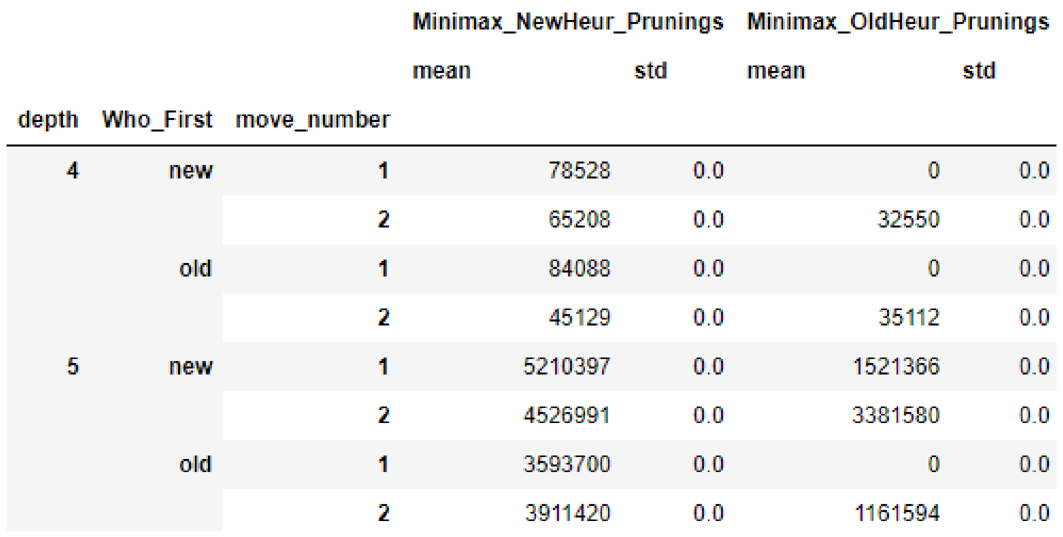


The fourth experiment investigated the amount of node prunings per move by both heuristics. The minimax bot played against a random bot for 50 games with the minimax bot going first.

The minimax algorithm pruned significantly more nodes with both heuristics at depth 5 compared to depth 4. The new heuristic caused more nodes to be pruned in the first two moves than the old heuristic for both depths; after that, the curves follow a similar pattern. This experiment also originally involved 50 games with the minimax bot going second, however the results were extremely similar to what was found already, so they were omitted for brevity.

Figure 4: Experiment 4 results on average pruning per move by both heuristics against a random bot.

Experiment 5 explored average node prunings per move for both heuristics. Each heuristic played against the other heuristic, going first for 5 games, and going second for 5 games. This was repeated for two depths, 4 and 5 to eliminate depth-specific biases.



			Minimax_NewHeur_Prunings		Minimax_OldHeur_Prunings	
			mean	std	mean	std
depth	Who_First	move_number				
4	new	1	78528	0.0	0	0.0
		2	65208	0.0	32550	0.0
	old	1	84088	0.0	0	0.0
		2	45129	0.0	35112	0.0
5	new	1	5210397	0.0	1521366	0.0
		2	4526991	0.0	3381580	0.0
	old	1	3593700	0.0	0	0.0
		2	3911420	0.0	1161594	0.0

Figure 5: Experiment 5 results on average node prunings per move for both heuristics. Separated into multiple groups to show how the data was collected, with depth, who went first, and move number as important attributes.

5 Discussion

Regarding efficiency, we found that at depth 4, the old heuristic had similar move calculation speed to the new heuristic, but for depth 5 the new heuristic was much faster. We suspect this is attributed to the amount of pruning done by the minimax bot using each algorithm. The new heuristic was able to give a larger range of values than the old heuristic, which resulted in more prunings at higher depth, saving more time overall. This may not have been a factor at depth 4 due to the way the game played out. The minimax bot has no randomness, so every game at the same depth between the two heuristics has the same sequence of moves. The data we collected in Figure 4 on the minimax bot's pruning per move at depths 4 and 5 against a random bot shows the new heuristic to be pruning significantly more nodes than the old heuristic for moves 1 and 2. Pruning more nodes means the algorithm takes less time to complete due to a smaller tree to traverse. Thus, the new heuristic calculates moves quicker than the old heuristic at higher depths.

We claim that two minimax bots with different heuristics playing against each other will play out any number of games in the exact same way (assuming their play order isn't swapped), and this is supported by the data in Figure 5. While the difference between depths in average node prunings for the first two moves is interesting, the focal point is the standard deviation. We found exactly 0 deviation in prunings, which should only occur when equivalent moves are being performed in the same order. Since the number of node prunings is a quantitative value with a large range, it should be very unlikely for two different trees of this magnitude to have the same prunings. Coupling this data with experiences playing against the minimax bot and knowledge of its inner processes, we hold the claim we made above to be true.

We determined the success of this project based on the results of the matches between the minimax bot and its opponents (random-move bot or human players). We defined a full success for this project as creating an algorithm that can beat human players consistently; in failing that, we defined a partial success as creating an algorithm that can consistently beat a random-move bot.

Based on the data in Figure 1, we have sufficient evidence to conclude that this project is at least a partial success, as the minimax bot we designed with the heuristic we created achieved a 98% win rate against a random bot regardless of depth. Since the random bot does not use depth, this variable only affects the minimax bot. This may suggest that the heuristic performed worse at depth 5 than at depth 4, but we hypothesize that this was due to the random bot finding lucky moves to tie the games. The minimax bot never lost to the random bot, which helps support the idea that the heuristic is still performing well, as if it performed much worse at depth 5, we would likely see a few losses.

Pitting human players against the new heuristic always led to ties when the human player was given ample time to think and planned out multiple moves in the future. If the human player focused on obtaining immediate side wins, the new heuristic would usually beat them. This aligns with the data found in experiment 2, as the new heuristic was able to beat the old heuristic in all cases except when the old heuristic went first at depth 5, where every game was a tie. These results suggest that the new heuristic is better than its opponents at lower depths, but increasing the depths leads to more ties. Based on experience playing this variant, as well as an understanding of classic Tic-Tac-Toe, the positive correlation between overall game depth (both players at the same depth) and number of ties is likely due to finding the optimal way of playing the game. Classic Tic-Tac-Toe is solved and always ends in a tie under optimal strategies, and we believe this applies to our variant as well. Our results leave us unable to conclude a full success based on our definition of a full success above. However, this new knowledge about our variant leads us to believe a full success was never possible, as a player cannot consistently win in a solved-for-a-tie game.

6 References

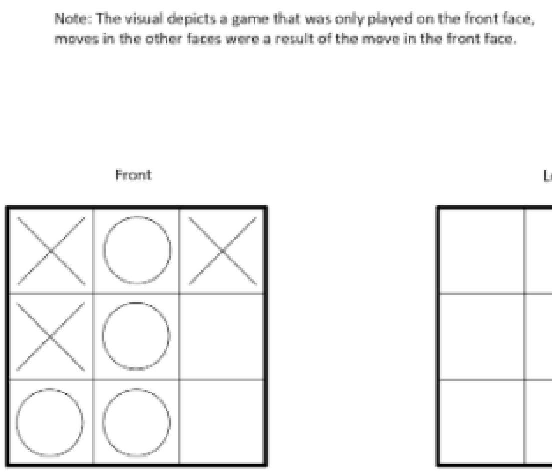
Cranenburgh, A.V., Smid, R. (2007). *Tic-Tac-Toe* [Unpublished manuscript]. University of Amsterdam.

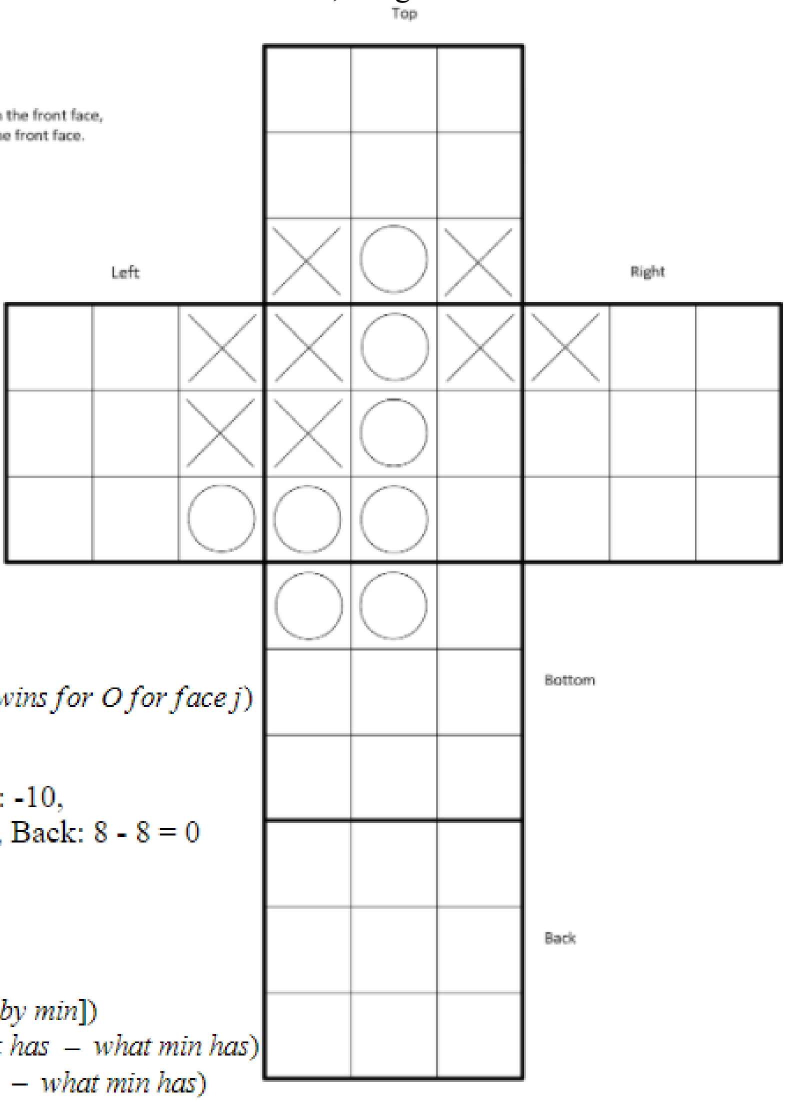
Zain, A., Chai, C., Goh, C., Lim, B., Low, C., & Tan, S. (2020). Development of Tic-Tac-Toe Game Using Heuristic Search. *IOP Conference Series: Materials Science and Engineering*, *864*(1), 012090. https://doi.org/10.1088/1757-899x/864/1/012090

Appendix

Cubic Tic-Tac-Toe Game Rules

1. Each player goes one at a time placing X's or O's like how original Tic-Tac-Toe is played
2. If a player places a move on an edge or corner square, any (directly) attaching edges (on other faces of the cube) are also counted as the player's current move.
3. Each face of the cube is a separate Tic-Tac-Toe game. Winning a Tic-Tac-Toe game on that face means winning that face of the cube.
4. When a player wins 4 faces or all faces have been won, the game ends.


Note: The visual depicts a game that was only played on the front face, moves in the other faces were a result of the move in the front face.
Front



Using 1st (old) heuristic:

$$\sum_j (\textit{open wins for X for face } j - \textit{open wins for O for face } j)$$

Top: 6 - 3 = 3, Left: 5 - 4 = 1, Front: -10,
Right: 8 - 5 = 3, Bottom: 4 - 8 = - 4, Back: 8 - 8 = 0
Total: -7

Using 2nd (new) heuristic:

$$50 * ([\textit{sides won by max} - \textit{sides won by min}])$$
$$+ \ 2 * (\textit{number of } 2 \textit{ in a rows that max has} - \textit{what min has})$$
$$+ \ 1 * (\textit{number of } 1 \textit{ in a rows max has} - \textit{what min has})$$

50 * (0 - 1)
+ 2 * (0 - 1) (O or MIN has a 2 in a row in the bottom face)
+ 1 * [(4 from Top, 3 Left, 3 Right for X) - (1 from Top, 2 Left, 3 Bottom for O)]
= -50 - 2 + (10 - 6) = - 4

Contribution Statement

Tyler - 16%
Tyler was tasked with coding the basics of our 3D Tic-Tac-Toe game. This included designing the data structure for the gameboard that scripts depended on and coding functions for making a move and checking for wins on each side. He also worked on code optimization, code review, and running experiments. For our final paper, Tyler proofread and edited the paper to ensure everything flowed well.

Susan - 18%
Susan designed a random bot that chooses random moves. She wrote the basis of data collection Python code (runExperiment and autoExperiment) and R code to produce graphs and charts included in our final paper. Susan was heavily involved in optimizing our code and analyzing the results of our experiments. She contributed to the Methods and Discussion sections of our Final Report and created graphs to visualize our results.

Alex - 18%
Alex designed a Minimax bot with heuristics. He worked on large optimizations of minimaxBot, headed early performance testing, and conducted code review. He was also invested in improving and running our experiments. Alex stepped up throughout the project to assign roles for each team member, lead weekly group meetings, and ensure that we followed a timeline to meet the project deadline. For the final report, Alex aggregated our data so that we could seamlessly incorporate it into the paper.

Michael - 16%
Michael was tasked to write unit tests to check all Python scripts work together and write a driver that runs a two player game for as many iterations as we need. He aided in running our experiments of Tic-Tac-Toe. As we wrapped up our project, Michael worked on data aggregation of results and performed code review. He contributed to the results and methods section of our paper.

Melinda - 16%
Melinda worked on modifying a classic heuristic for Tic-Tac-Toe to be usable in our variant of the game. She also developed a new heuristic for our variant of the game. She and Julie would play multiple rounds of the game to learn strategies that would be useful in our heuristic. She also looked at the code written by others to gain an understanding of the game. For the final report, she assigned parts of the paper for each team member and contributed to the introduction, literature review, and methodology.

Julie - 16%
Julie also worked on modifying a classic heuristic for Tic-Tac-Toe to be usable in our variant of the game. She also developed a new heuristic for our variant of the game. She and Melinda would play multiple rounds of the game to learn strategies that would be useful in our heuristic. Additionally, Julie participated in code optimization. For the final report, she contributed to the introduction, literature review, methodology, and appendix sections of the paper.