Katrina Qiu

Nicky Mosharaf

Zainab Imadulla

Professor Rose Sloan

CS4100 Artificial Intelligence

4 December 2024

Final Paper

I. Introduction

For our final project, we chose to create an agent capable of playing Connect 4, a two-player board game that will provide the framework for implementing the minimax algorithm with alpha beta pruning. The minimax algorithm is utilized for decision making in two-player, zero-sum games, where the agent attempts to maximize its score and assumes the opponent is trying to minimize it. Alpha-beta pruning enhances the algorithm by pruning off branches of the game tree that cannot influence the outcome. The minimax algorithm alongside alpha-beta pruning serves to ensure that our agent chooses the most optimal move that results in the best outcome for itself (a victory or a draw). Carl Felstiner's writing on this topic, "Alpha Beta Pruning," served as key inspiration, as he discusses implementation of the minimax algorithm and alpha-beta pruning in agents that play Tic Tac Toe, Hex, and 3D Tic Tac Toe. Both Tic Tac Toe and Connect 4 are two-player, zero-sum, and involve grid-style boards, with win conditions based on aligning tokens. There are differences that lead to Connect 4 having a larger branching factor than Tic Tac Toe, such as larger board dimensions. In the article, Felstiner provides some background on this

topic, explaining how computers build a tree of possible moves and outcomes for a game and work through it to find the move that will result in the best outcome. Though in games with larger game trees, he notes that “the challenge for the computer is then to find ways to avoid searching in certain parts of the tree” (Felstiner 1). The data in the article clearly demonstrates that alpha beta pruning can significantly reduce the nodes to be explored in Tic Tac Toe, Hex, and 3D Tic Tac Toe, thus improving the efficiency of the minimax algorithm. Felstiner’s writing was able to deepen our understanding of optimizing the decision-making process of our agent and serve as guidance for the implementation. Our project will observe the results of using different heuristics in our algorithm to demonstrate how they may affect the accuracy and decisions of the agent.

II. Methods

In order to begin implementing the minimax algorithm in a Connect 4 game, we created a Graphical User Interface that first displays a main menu that is a button panel of game modes. These options include player vs. player, player vs. minimax, minimax vs. minimax, and minimax vs. random. Once a game is started, the program displays the board that the two opponents would play on. The player vs. player and player vs. minimax options involve user interaction, while the minimax vs. minimax and minimax vs. random were primarily used for evaluation of the heuristics used. The GUI also features ‘Restart’ and ‘Back’ buttons that allow users to restart a round or go back to the main menu respectively. In the game modes that do not involve the user playing, there is a ‘Begin Game!’ button that initiates the game between the opposed agents. The CustomTkinter library in python was used for the interface, constructing a board with tokens and empty slots for each player. The user simply needs to click on the column they want to place their token in, and the program updates the board to reflect the new game state. The program

handles choices made by the players and other changes to the game state (checking win conditions, moving validity, etc.). The minimax algorithm determines the optimal move by simulating the possible moves up to a depth of three (recursive depth-limited search), alternating between maximizing the agent's score during the agent's turn and minimizing the agent's score during the opponent's turn. A depth limit of three was chosen to achieve a balance between computational efficiency and depth. We wanted to avoid too large of an exponential increase in exploration without sacrificing the ability of our agent. The algorithm calls the evaluation function on terminal states when they are found (victory for the player, loss for the player, the board is filled, or the depth limit is reached). Alpha-beta pruning improves the efficiency of the algorithm by skipping exploration of branches of the game tree that do not influence the final decision (i.e. a move that is already guaranteed to be worse than a move that has already been explored).

The evaluation function represents the heuristic used in our algorithm. In this project, we studied the effect of using three different heuristics on the agent. The first heuristic uses center cell prioritization, where cells are given higher weights the closer they are to the center of the board, as control of the center often leads to a more favorable outcome. The second heuristic also uses center cell prioritization, though it also accounts for the number of two-in-a-row, three-in-a-row, and four-in-a-row configurations for both the player and the opponent (with a priority towards defensive moves). Higher weights are given to longer chains of tokens: a four-in-a-row configuration is given the highest weight as it leads to a victory, followed by three-in-a-row, then two-in-a-row. We made the player prioritize defensive moves by giving more weight to an opponent's three-in-a-row configuration than a three-in-a-row in the player's configuration. This makes it so that if the agent is given a choice between completing its own four-in-a-row

sequence or blocking the opponent from achieving a four-in-a-row sequence, it will choose the latter. The third heuristic is identical to the second heuristic except that it prioritizes offensive moves over defensive moves. The player's three-in-a-row configuration is given a higher weight than the opponent's three-in-a-row configuration and thus it will choose to make a winning configuration for itself when presented with the opportunity to either make a winning configuration for itself or block a winning configuration of the opponent. The assignment of three-in-a-row weights can be viewed in the weigh_current_connections method in board.py. In our program, the default heuristic setting is the offensive heuristic, but the minimax method takes in a parameter signifying which heuristic should be used.

In addition to the varying heuristics, we wanted to observe how well the agent did when competing against another agent using the same minimax algorithm and an agent that chooses random but valid moves. We will discuss the results of the three heuristics and the agent's performance against different types of opponents in the 'Results' section. To evaluate the code, unit tests were written to ensure the functionality of the methods. These tests achieved this by simulating scenarios such as adding a token in specific columns, forming winning token sequences, filling a board completely, etc. and confirming that the program was responding correctly and accurately. Tests were also written for the evaluation function to ensure that correct scores were being returned for the heuristics. Given we are interested in the results of different heuristics, we tested the three heuristics against human players, another minimax agent, and an agent that plays randomized moves. We then kept track of victories, losses, and draws to analyze the effectiveness of the heuristics.

III. Results

We initially hypothesized that the heuristic made up solely by center cell prioritization weighting would yield the lowest performance, while the remaining two heuristics that accounted for token configuration were expected to perform at similar levels. This is due to the center heuristic primarily accounting for token placement relative to the center of the board, while defensive and offensive heuristics account for proximity to the center of the board and additionally, the token chain lengths. The added factor of the token configurations was assumed to reduce the number of losses.

Based on our results, we conclude that the heuristic that prioritizes offensive moves was the most effective of the three, followed by defensive, and lastly center cell prioritization. We began with observations of the minimax agent vs. minimax agent results. In games between the center prioritization heuristic and the defensive prioritization heuristic, the defensive prioritization heuristic emerges victorious every round (see Figure 1). When we ran the center prioritization heuristic agent against the offensive prioritization heuristic, the offensive prioritization heuristic emerged victorious every round (see Figure 2). Lastly, between the defensive and offensive heuristics, the agent using the offensive heuristic won every single round (see Figure 3). We can state that these agents were able to emerge victorious over others with surety, as any minimax agent vs. minimax agent rounds resulted in identical boards no matter how many rounds were run. Lastly, because the agent that plays only randomized valid moves is not making optimized plays, the minimax agent emerges victorious, as the probability of the randomized agent winning is extremely slim (see Figure 4).

Next, we tested the agent with the heuristics against human players. The center heuristic agent lost against a human player three times out of a total of 40 games, the defensive heuristic agent lost against a human player two times out of a total of 40 games, and the offensive

heuristic agent did not lose against a human player in a total of 40 games. The size of the human player set was seven individuals, and we acknowledge that this is far from a representative set. It is possible that a larger and more varied sample size could result in different conclusions. The results of our human player vs. minimax agent games provide additional support to the results demonstrated in the minimax agent vs. minimax agent games.

Overall, the results indicate an alignment with the initial hypothesis. The center heuristic agent was the weakest, but we did not initially expect the defensive heuristic agent to be weaker than the offensive heuristic agent. As the center heuristic prioritizes stacking, at times it misses opportunities to win. The defensive heuristic agent proved to be more strategic than the center heuristic agent, and its only recognizable flaw was the case that was addressed earlier: where the agent is presented with the opportunity to make a winning move for itself or block the winning move of its opponent, it will choose to block the opponent. In general, all three heuristics proved to be very strong against human players.

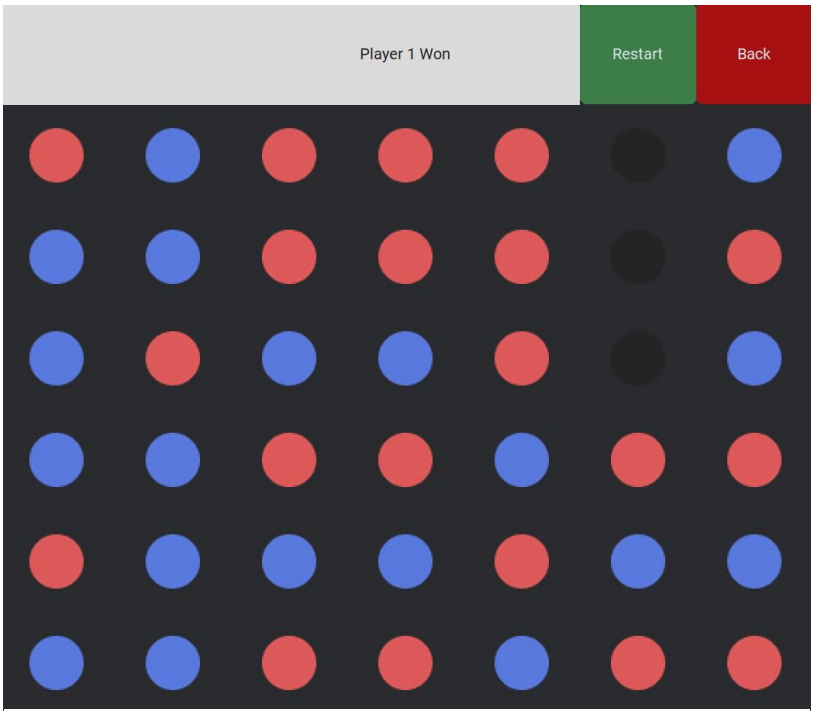

Figure 1: Player 1 (red) is the defensive heuristic agent and Player 2 (blue) is the center heuristic agent. This is an example of one of the configurations of the board once the game has ended.

Figure 3: Player 1 (red) is the offensive heuristic agent and Player 2 (blue) is the center heuristic agent. This is an example of one of the configurations of the board once the game has ended.

Figure 2: Player 1 (red) is the defensive heuristic agent and Player 2 (blue) is the offensive heuristic agent. This is an example of one of the configurations of the board once the game has ended.

Figure 4: Player 1 (red) is the offensive heuristic agent and Player 2 (blue) is the randomized move agent. This is an example of one of the configurations of the board once the game has ended. Note how the game did not progress as far as Player 2's moves were not optimized.

IV. Conclusion

In terms of strengths, the implementation of minimax with alpha-beta pruning was an efficient and effective approach towards decision making in Connect 4. This approach allowed the agent to perform well using the different heuristics even with the constraints of a depth limit of three. While we chose a depth limit of three as a reasonable number that would not compromise too much computational power or strategy, there is possibly a different depth limit that could be more effective and still balance the computational power. This could be addressed with dynamic depth adjustment so that the agent can analyze further in more critical game states while still being able to conserve some computational power in less critical situations. Another strength of our approach was the exploration of three distinct heuristics, which provided insight

into the influence on the agent's performance. Additionally, while the agent performed well against human players, the data coming from a smaller sample size of individuals is not a comprehensive representation of its effectiveness across diverse playing styles. Another way to improve our approach is a more thorough and comprehensive testing process, whether it involves human players, or another Connect 4-playing agent.

Works Cited:

Felstiner, Carl. *Alpha-Beta Pruning*. 2019.

Schimansky, Tom. “CustomTkinter UI-Library.” *GitHub*, 29 July 2022, github.com/TomSchimansky/CustomTkinter.