CS 221 Project Final Report - 3D FourPlay AI Agent

Timothy Lee (timothyl), Edwin Park (edpark), Anthony Ma (akma327)

Introduction

FourPlay, also commonly known as Connect Four, is a two-player board game in which each player takes a turn in dropping a colored disc into a 6-by-7 vertically suspended grid. The objective of this game is to connect four of one's own pieces next to each other horizontally, vertically, or diagonally. Given the adversarial, zero-sum game nature of our project, we will use AI algorithms to have a computer learn to play Connect Four.

We have designed our project based on a 4x4x4 3-D version of Connect Four. In this case, there is a 4-by-4 base grid, and pieces can be stacked upon each base tile to win in a similar fashion (horizontally, vertically, and diagonally in 2-D and even 3-D!).

Related Work in 2D Connect Four

The 2D Connect Four has been a standard application of principles of Artificial Intelligence. Some common approaches found in related literature have been the recursive minimax algorithm with alpha-beta pruning, TD Learning, A* influence maps, and neural networks[1]. Minimax algorithms have been implemented with alpha-beta pruning up to depth 5, and they have been largely successful and fast primarily because the search space is exponentially smaller than the 3D version of Connect Four.[4] These AI agents have been effectively win against human players because of their ability to analyze all board configurations at once and compute optimal movements efficiently.

Furthermore, Giuliano Bertoletti has built Velena,[7] a freeware expert system for playing 2D version of Connect Four perfectly, based on eight mathematical rules that were proven in L. Victor Allis's master thesis, "A knowledge-based Approach of Connect Four". Using proof-number (PN) search, Velena reduces the game complexity from 7.1 * 10^13 to just 60,000, allowing the system to virtually precompute all possible winning solutions. Given that the search space is exponentially greater in the 3D than the 2D version, using proof-number search is still infeasible.

In this paper, we apply minimax with alpha-beta pruning as well as TD learning to the 3D variation of the standard connect four game. Given that the game is most typically played in the 2D version, not many studies have attempted to apply AI approaches for the 3D version, which involves a greater search space. In our 3D variation of the game, we can use the similar methodologies explored above to implement an AI agent using minimax algorithm with alpha-beta pruning and TD Learning, while managing a more challenging task of handling large state space.

Model

We modeled FourPlay as an adversarial game with the following structure:

Players(s): {1,2} which denotes Player 1 and Player 2.

State: A 4*4*4 matrix (that can be extended to an i*j*k cube) with values of 0 denoting an empty spot, 1 as occupancy of player 1 and 2 as occupancy of player 2. We will use the terminology “base column” to denote the j dimension and the “vertical column” to denote the k dimension in which the tile drops.

Actions(s): An (i, j) tuple denoting the ith row and jth base column of the state to place a new tile by the agent. If the vertical column is fully occupied at the (i,j) position, meaning it already has k tiles, then (i, j) is not a valid action.

IsEnd(s): Returns True when either player 1 or player 2 have a set of 4 contiguous tiles (contiguous in any straight direction: vertical, horizontal, and diagonal) that are its own. It also returns true if there are no legal possible actions left. Else, false is returned.

Utility: Returns +inf if player 1 has won, or -inf if player 2 has won, or zero if no one has won yet. When there are no legal actions possible, the utility is still zero.

Implementation

Random Agent (First baseline algorithm)

The random agent retrieves the list of all legal moves for a given state, then randomly choose a move out of these legal actions to make the next move. Therefore, the random agent does not consider the current state of the board and does not consider the possibility of winning or losing given a game state.

For a random agent, there are n^2 possible successor states that the random agent could reach on a given state, with a uniform probability. Even if the random agent is at a game state where it could win, the probability of making the right move to win is extremely low. Consequently, the random agent can almost never win. We found out that using the random agent as a baseline algorithm fails to give us meaningful results for our error analysis, so we decided on an improved baseline algorithm: reflex agent.

Reflex Agent (Second baseline algorithm)

The reflex agent first looks at all possible successor states, which can be reached by placing a piece in one of n^2 slots. Then, it determines the optimal policy by using the evaluation function to evaluate all the successor states. In Connect Four, a player can only win on the player’s turn, and only lose on the adversary’s turn.

For example, if the reflex agent recognizes that the current state has a possible winning move, then it places a piece in that position to win. Similarly, if the reflex agent recognizes that the next move made by the adversary can lead to a loss, it must block the position by placing a piece.

Minimax Agent (with Alpha-Beta Pruning)

We have implemented the Minimax agent, which picks an action by choosing the legal action that has a maximum value. The value is found by recursing down the search tree for only a set depth. We then evaluate the state at this depth if isEnd(state) is False, using the evaluation function. For trials where two alpha beta agents are playing against each other, we have implemented the minimax agent to be either initialized as the agent that seeks to maximize the score or the agent that seeks to minimize it. We currently only run the minimax agent on a depth of one, because of the high latency of each move at higher depths.

Evaluation Function

The evaluation function works by counting the number of rows (in any direction) that have 2 contiguous tiles of the same player along with at least 2 free spaces around, and rows that have 3 contiguous tiles of the same player with 1 free space around. Each count is multiplied by a weight, designated such that rows with more contiguous tiles have a higher absolute weight since they contribute to a more likely win. In congruence to the fact that player 1 maximizes the score while player 2 minimizes the score, the weights associated with rows of player 1 tiles are positive and the weights associated with rows of player 2 are negative. It was assumed that the weights between player 1 and player 2 have the same absolute value since this is a zero sum game, and consequently, patterns from the first player equally affected the chance of winning for first player as it did the chance of losing for the second player.

Oracle

Our oracle is a web application called Connect 3D set at the most difficult gameplay standard called “Hard” out of the possible settings {Beginner, Easy, Medium, Hard}. Connect 3D is one of the best 3 dimensional connect four AI agents available online and can be found here. https://www.mathsisfun.com/games/connect3d.html

Automation of Gameplay Between AI Agent and Oracle

In order to accurately evaluate our model and implementation against the oracle, we required an automated way to have the AI agent play against the oracle.We embedded the oracle web application into a bare bone html, with only the game board interface. To recognize the oracle’s move every iteration, we performed image subtraction of the screen shots of the board between current and previous iteration. The tile detection performs at 100% accuracy. We then use a mouse click event to input our AI agent’s calculated move.

TD Learning

With this automated way of playing against the oracle, it was feasible to obtain enough game plays in order to learn the weights needed for evaluation function that approximates the favorability of winning given the state. First, we played against the oracle with an alpha-beta

agent with an exploration probability of 0.2. This signifies that 20% of the time, a random legal action rather than the action picked by the alpha-beta agent was used to explore other possible states instead of the state that would seem to be the best possible. Learning rate was decreased after each game transcript from its initial value of 0.01 by dividing by the number of games seen. The initial weight values with respect to the feature of number of rows with 2 consecutive unblocked tiles of the same player and the feature of number of rows containing 3 unblocked consecutive tiles of the same player were equal. We wanted to test the hypothesis that as explained in the *Evaluation Function* portion of the report, rows containing 3 unblocked consecutive tiles were more important than those with just two, thus having a higher absolute weight. We played the oracle this way 100 times.

Next, by the assumption that weights associated with the player that maximizes utility should have same absolute value, but opposite value, as the weights associated with the second player that minimizes utility (see *Evaluation Function*), we updated the weights, while maintaining shared weights with the TD learning algorithm. We modified the update rule for weights of features corresponding to the other player, since they needed to be its opposite value. For weights of features of the player maximizing the final utility, we have the regular update rule:

$$w \leftarrow w - \eta[V(s;w) - (r + \gamma V(s';w)]\nabla_w V(s;w)$$

equation
(latex for edwin: $w \leftarrow w - \eta [V(s;w) - (r+ \gamma V(s^{'};w)] \nabla_wV(s;w)$\\)
where w is the feature weights, eta is the learning rate, gamma is the discount factor, blahblah

For weights of corresponding features of the other player who is minimizing the final utility, we have the same update rule as above:

$$w \leftarrow w + \eta[V(s;w) - (r + \gamma V(s';w)]\nabla_w V(s;w)$$

(latex for edwin: $- w \leftarrow -w - \eta [V(s;w) - (r+ \gamma V(s^{'};w)] \nabla_wV(s;w)$\\)
This comes from the reason that the negative of the original weight is updated like this:

$$-w \leftarrow -w - \eta[V(s;w) - (r + \gamma V(s';w)]\nabla_w V(s;w)$$

(latex for edwin: $ w \leftarrow w + \eta [V(s;w) - (r+ \gamma V(s^{'};w)] \nabla_wV(s;w)$\\)

$- w \leftarrow -w - \eta [V(s;w) - (r+ \gamma V(s^{'};w)] \cdot - \phi(s)$\\
$V(s;w) = w\cdot \phi(s)$\\
$V(s;w) = -w \cdot \phi(s)$\\
$\nabla_w V(s;w) = \phi(s)$\\
$\nabla_w V(s;w) = -\phi(s)$\\

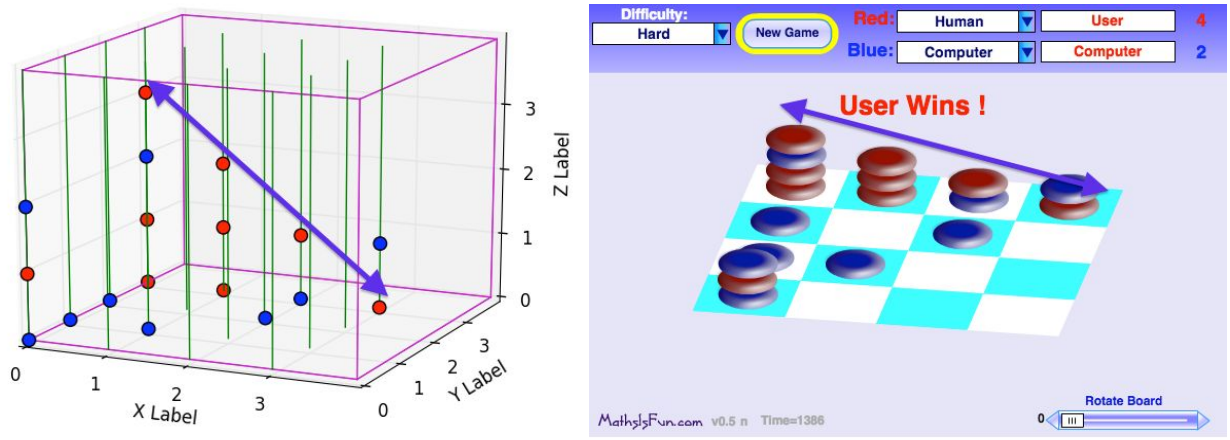

Figure 1: On the left we have our implementation of FourPlay and on the right is the interface for Connect 3D. We have the computer for Connect 3D represent the human decision (blue) for FourPlay, and the AlphaBeta minimax agent be the human decision (red) for Connect 3D. This particular game shows the outcome where our AI agent defeats the online oracle.

Experimental Results

1) Random agent vs Oracle

- The random agent won 0/50 games, with an average winning rate of 0%
- Average length of a game (all losing) is 8.0 moves.

2) Reflex Agent vs Oracle (Hard)

- The reflex agent won 2/50 games, with an average winning rate of 4%
- Average length of a winning game is 60 moves.
- Average length of a losing game is 39.5 moves.
- Average length of a game is 40.8 moves.

3) Reflex Agent vs Medium Oracle

- The reflex agent won 20/50 games, with an average winning rate of 40%
- Average length of a winning game is 39.2 moves.
- Average length of a losing game is 38.0 moves.
- Average length of a game is 38.5 moves.

4) Alpha-beta pruning minimax agent - Depth 1 vs Oracle

- The minimax agent at depth 1 won 75/100 games, with an average winning rate of 75%
- Average length of a winning game is 40.8 moves.
- Average length of a losing game is 26.4 moves.
- Average length of a game is 37.2 moves.

5) Alpha-beta pruning minimax agent - Depth 1 + TD Learning vs Oracle

- The minimax agent at depth 1 + weights from TD Learning won 41/50 games, with an average winning rate of 82 %
- Average length of a winning game is 41.5 moves.
- Average length of a losing game is 34.17 moves.
- Average length of a game is 39.9 moves.

6) Alpha-beta pruning minimax agent - Depth 2 vs Oracle

- The minimax agent at depth 2 won 9/10 games, with an average winning rate of 90%
- Average length of a winning game is 35.3 moves.
- Average length of a losing game is 41.0 moves.
- Average length of a game is 36.1 moves.

Analysis

With this project, our group was able to design, implement, and create a fully functional 3D Connect Four AI agent and a graphical interface to achieve our goal efficiency and winning accuracy against the current best oracle available online. With a depth 1 minimax agent we were already able to beat the oracle 75% of the time, and by increasing depth to 2, our agent is winning 90% of the time. Compared to our random agent baseline of 0% winning rate and reflex agent's 4% win rate, our implementation of minimax makes orders of magnitudes of improvement. Our implementation was able to do better than the reflex agent baseline because of the inherent strategy to make runs of 2 or 3 tiles and block the opponent's runs rather than reacting to just an opponent's possible winning move.

It would be expected, that having further improvement to computational efficiency and allowing greater depths would help our agent achieve even higher winning rates. From our results, however we already see that success rate is highly correlated with greater depth and the usage of TD learning to optimize weights for the evaluation function. Ultimately, we are very satisfied with our results because we have significantly cut down completion time with variable depth within a 3D search space to be able to utilize depth 2 conditions efficiently. More importantly. we have achieved performances beyond current state of the art implementations for 3D Connect Four.

Reinforcement learning techniques such as TD Learning helped us pinpoint appropriate weight values for our rudimentary features in the score evaluation function. Currently our two features would be the presence of length 2 and length 3 chains. Even with these two basic features, TD Learning resulted in a 7% overall improvement to winning rate (75% → 82 %) against the oracle. However we expect having a better refinement in our definitions of our current features, and the creation of other more sophisticated features would yield even greater improvements across different depths.

As discussed in the Implementation section, we hypothesized that chains of length 3 should have a higher importance than chains of length 2 due to their closer proximity to a

winning move. As such, chains of length 3 should have a higher absolute weight. This was observed after learning the feature weights, where the weight corresponding to the number of chains of length 2 was 4554 while that of length 3 was 6986, substantially higher than the former.

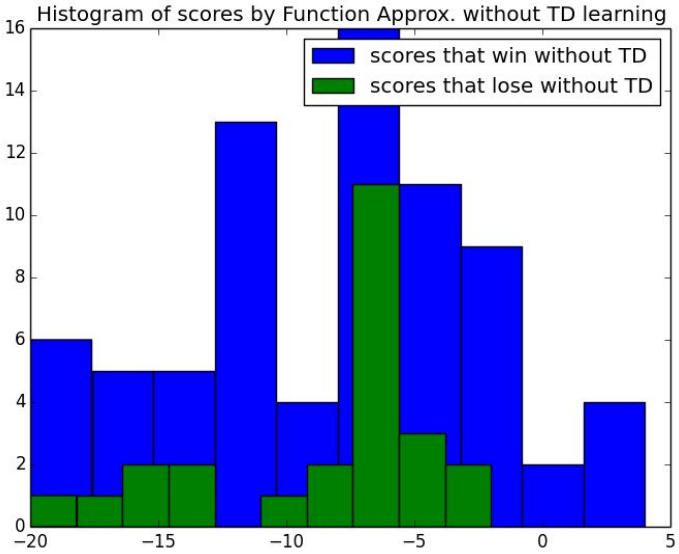

Figure 2: A histogram for score distribution without TD learning.

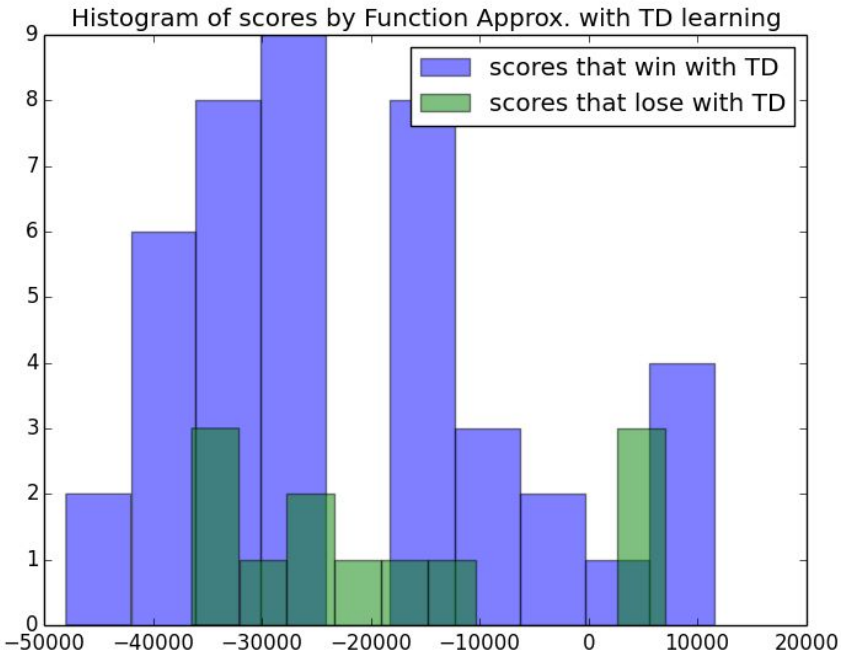

Figure 3: A histogram with the score from the evaluation function of the final state that led the minimax agent (a utility minimizer) to a win or loss.

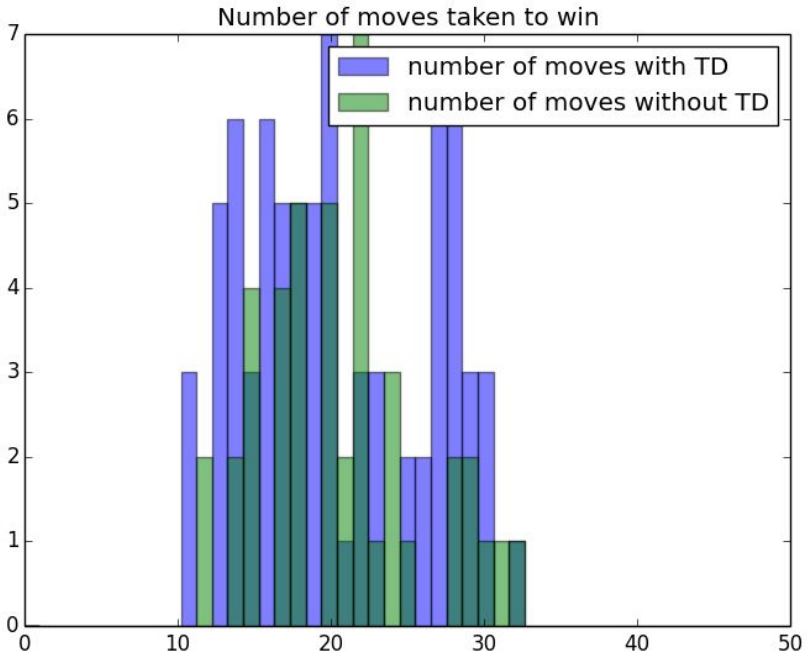

Figure 4: Distribution of number of moves to winning state for our AI agent with and without TD learning.

Despite the improvement of our algorithm that the features provide, our features from Figures 2 and 3 do not seem to be very predictive of a win or loss. These histograms show the score from the evaluation function of the final state where the AI agent (a minimax agent that minimizes utility) for each game. Figure 2 comes from trials without the learned weights (instead initialized at 2) while Figure 3 comes from trials with the learned weights from TD learning. The huge overlap between the score distributions between wins and losses shows that the features themselves cannot differentiate very well between a winning and a loss. In addition, the faults of this set of features may be also reflected in the number of moves taken to win. In general, a better algorithmic approach would be able to move to a state that wins faster, and require less moves. Figure 4 shows the histogram of number of moves taken to win. The high overlap between the distributions shows no significant improvement in terms of number of moves taken to win. We will describe the possible problems with our current set of features and ways to improve them.

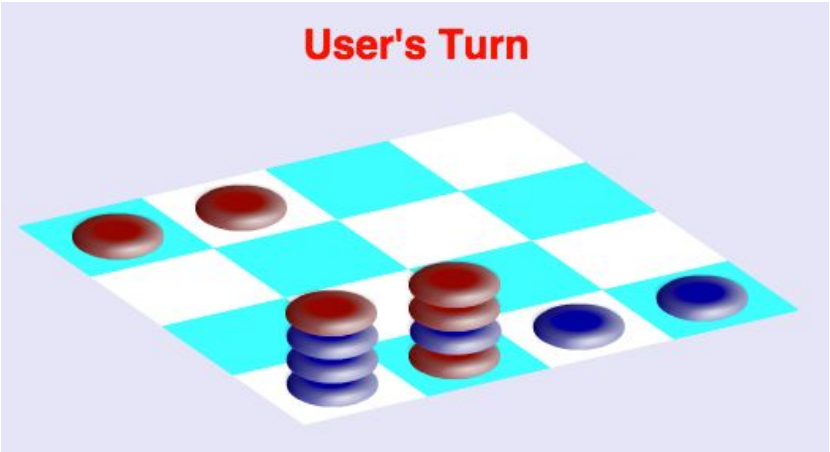

Figure 5: Forming a chain of 3 red tiles in the first layer upper row, and on the fourth layer bottom row are considered with equal importance. In reality, the former requires fewer moves and should be given higher priority.

Currently, one of the main issues we have with defining features simply as length n chains is that there is no differentiation between the placement of the chains. For example, as shown in Figure 5, a row of 3 red tiles on the first layer is of the game board is much easier to achieve than have a row of 3 red on the fourth level because the latter requires other pieces be placed in order to build up the columns. Because of this lack of differentiation, our AI agent would have equal priority of spending moves to create a length 3 chain in the fourth layer, when it could have created this length 3 chain in the first layer with fewer moves. This sort of inefficiency is reflected in our results, because the average length of game is 39.9 moves for depth 1 minimax with TD learning and 37.2 moves without. Introducing new features to encapsulate the geometric placement of these chains and the context around each row would most likely improve our performance.

Conclusion and Further Work

Our 3D Connect Four agent performs very well in terms of winning rate against the best available oracle. Even with a very high 3D search space, our implementation was able to run on multiple depths and achieve reasonable run times. There are, however, many improvements that can still be integrated into our engine.

First we would like to focus on further computational optimizations, to make it feasible to run minimax upon depths 3 and greater. To do this we can utilize more efficient infrastructure such as cython, caching state transitions computed in previous iterations to avoid redundant computations, and identifying and excluding low probability successor states. Furthermore, we can utilize parallelization to identify winning states in columns, rows, and diagonals in the three dimensions independently. Since identification of winning states and counting of features for the evaluation are the greatest bottlenecks in our computation time, parallelization would be expected to make major improvements to our efficiency. From a more algorithmic standpoint, we

saw a 250x difference in the number of leaves between two consecutive depths due to the high branching factor of 16. Thus, implementing a BEAM search to reduce our minimax tree search space would also be very helpful when trying to optimize our implementation to higher depth searches.

Currently, we only have the fundamentals of a simple evaluation function by considering length two and three chains for opponent and our own tile pieces. Running TD Learning to compute the best weights for these two features alone improved our alpha beta minimax by 7% in winning rate. We expect incorporating more domain specific knowledge and strategies from current published research on 2D connect four would be very beneficial in making further incremental improvements to our engine.

The code for our 3D Connect Four agent is available on our GitHub page.
https://github.com/parkedwin/FourPlay

Sources

1.) http://thescipub.com/PDF/jcssp.2009.283.289.pdf
2.) https://www.mathsisfun.com/games/connect3d.html
3.) http://cython.org/
4.) http://roadtolarissa.com/connect-4-ai-how-it-works/
5.) http://www.computer.org/csdl/proceedings/sbrn/2002/1709/00/17090236.pdf
6.) http://www.researchgate.net/publication/228931327_Evolving_Connect-Four_Playing_Neural_Networks_Using_Cultural_Learning
7.) http://www.ce.unipr.it/~gbe/velena.html