# Comparing Four Connect-Four AI Agents

Cameron Knopp
Department of Computer Science
University of Massachusets Lowell
Lowell, MA, 01852, USA
Cameron_Knopp@student.uml.edu

Linik Yonn
Department of Computer Science
University of Massachusets Lowell
Lowell, MA, 01852, USA
Linik_Yonn@student.uml.edu

***Abstract*—We implemented a Particle Swarm Optimization AI agent for Connect Four and put it into a tournament-style competition along with three other AI agents. The other AI agents were Minimax with alpha-beta pruning, Expectimax, and a random AI. Ultimately, we found that Minimax with alpha-beta pruning is the most effective of the four algorithms for this game.**

***Keywords—AI, Connect Four, Particle Swarm Optimization***

## I. INTRODUCTION

Connect Four is a two-player board game in which each player first chooses a color and then the players take turns dropping their colored pieces into the columns of a 7x6 grid. When a piece is dropped into a column on the grid, it falls straight down and occupies the next available open space in that column. In order to win this game, a player must get four pieces in a row, either horizontally, vertically, or diagonally.

In this paper, we implement four different AI to play Connect Four and we report the outcomes of having them compete against one another. We implement Particle Swarm Optimization (PSO), Minimax with alpha-beta pruning, Expectimax, and a random AI. The following subsections give a brief overview of these four algorithms.

### *A. Minimax w/ Alpha-Beta Pruning*

Minimax is a recursive decision-making algorithm which chooses an optimal move for the player by following the assumption that the other player is also acting optimally. In following this strategy, the Minimax algorithm minimizes the maximum possible loss (i.e., it minimizes the worst-case scenario by considering all the best responses that the opponent would make). Alpha-beta pruning increases the efficiency and speed of this algorithm by reducing the number of nodes evaluated.

### *B. Expectimax*

The Expectimax algorithm is very similar to the Minimax algorithm, except that it does not follow the assumption that the opponent is playing optimally. Instead, the Expectimax algorithm takes the average of the utility of the opponents possible next moves.

### *C. Particle Swarm Optimization*

Particle Swarm Optimization is an evolutionary algorithm which is based upon the behavior of animal species living in large colonies, such as birds. It randomly initializes a set number of particles, called a swarm, into the search space. It then evaluates the current position of each particle according to a fitness function, and then updates each particle's position according to a mathematical formula [6]. The particle's next position is influenced by the best position which it has achieved thus far, as well as the best position achieved thus far by the entire swarm. Ultimately, the goal of this algorithm is to converge towards a global best position, which is what the agent chooses as its next move. The algorithm is not guaranteed to choose the optimal solution, however, as the mathematical formula it uses contains two random variables.

### *D. Random*

This AI simply picks its next move randomly from the list of next possible moves. We include this algorithm mainly to test that the other algorithms are truly playing intelligently, since a worthless algorithm will lose a high number of times to this fully-random AI.

## II. LITERATURE REVIEW

There has been a variety of different AI agents that have been implemented for Connect Four over the years, including Minimax w/ alpha-beta pruning [1], Expectimax, Negamax [2], Deep Q-Learning [3], Monte Carlo Search Tree [3], and AlphaZero [4]. These AIs have all proven to be effective at playing Connect Four, but it does not appear that there have been any published attempts at implementing a Particle Swarm Optimization agent for this game. For this reason, we believe it is a worthy task to implement this algorithm and compare it with the Expectimax and Minimax algorithms, which have already been shown to be effective for this game.

In addition, we increase the size of the board from 7x6 to 8x8, as it is known that dropping a chip in the middle column in 7x6 Connect Four gives the starting player a very sizable advantage. Researchers have discovered that the first player can win every time if they place a chip in the middle column and then proceed to play optimally from that point forward, regardless of what moves their opponent makes [5]. By eliminating this middle column, and instead having two nearly-central columns (the 4th an 5th columns on an 8x8 board), we are attempting to even-out the playing field by eliminating the possibility of the first player gaining an immediate outstanding advantage. We believe that this will give more interesting outcomes when running our tournament, as the first player will not simply be able to win every game by placing a chip in the middle column. We further attempt to even-out the playing field by having each player's first move be chosen randomly. This additionally makes sure that the AIs can handle a wide variety of situations, as they will not have the opportunity to make the exact same first move each game.

## III. Methodology

The following subsections give a high-level overview of our algorithm implementations and their respective heuristics, followed by an overview of how we run the tournament.

### *A. Minimax w/ Alpha-Beta Pruning*

We implement this algorithm with a depth of 6. Without alpha-beta pruning, we are only able to reach a depth of four before the agent takes an unreasonable amount of time to make a decision (>10 seconds). The alpha-beta pruning optimization allows us to reach a depth of six within a reasonable time-frame.

### *B. Expectimax*

We implement this algorithm with a depth of 4. Above this depth and we find that the algorithm takes an unreasonable amount of time to make a decision (>10 seconds).

### *C. Heuristic*

Our Minimax and Expectimax AIs both utilize the same heuristic function. We do this so that there is no advantage given to one or the other by the means of a better heuristic function. Instead, the algorithms are simply evaluated by their individual merit, i.e., how they choose to utilize the information given to them by the heuristic function. Also, as a side note, it was important that the heuristic function be simple as it is called many times by each AI and therefore needs to be computationally efficient. This heuristic function evaluates the quality of a given board state for a player in a few steps. First, the heuristic function gets a list of all the next possible locations where the player can place a chip. For each of these open positions, the heuristic function determines the number of possible 4-in-a-rows that can still be made from each of these positions (i.e., a possible 4-in-a-row must not have any enemy players in the way), and returns the sum of these numbers.

### *D. Particle Swarm Optimization AI*

The Particle Swarm Optimization AI uses a population of size 50 and a max generation of 250. We came to these numbers through experimentation by playing this AI against our Minimax AI until the games were not predictably won by one AI or the other. A population size of 50 is reasonable for this given search space, since the particles are randomly initialized to different positions on the board, which has a total of 64 spaces (since it is an 8x8 grid). The c1 and c2 values for this algorithm are both initialized to 2, as these are their standard values in a basic PSO implementation [6]. The r1 and r2 values are calculated anew, using NumPy's random function, each time the particle update equation is carried out.

### *E. Particle Fitness Function*

As described earlier, the Particle Swarm Optimization AI must determine the fitness of each of the particles that it has dispatched onto the board. The fitness function works nearly the same as the previously-described heuristic function. For a given particle, we first add up all of the possible 4-in-a-rows that can potentially occur, horizontally, vertically, and diagonally, from that position. However, where this fitness function differs is that it also takes into account whether the given position will win the game, whether the given position will prevent the opponent from winning, and whether the given position will set the opponent up for a win. It places top precedence to a position that will win the game for the player, followed by a position that will prevent the opponent from winning, and then finally avoids positions that will set the opponent up for win.

### *F. Random AI*

The random AI simply randomly chooses one of the next open valid locations.

### *G. Tournament Setup*

- Game played on an 8x8 grid, rather than a 7x6 grid, for reasons stated earlier.
- Each player plays against the other 100 times total.
- The player making the first move alternates for each game, so therefore in 100 games, each player will make the first move 50 times.
- First move for each player chosen randomly (i.e., first two moves of each game are random) for reasons explained earlier.
- We conclude that one AI is more effective than the other if it achieves more wins than the other.
- We deem the most optimal AI of the four to be the one with the greatest number of total wins.

### *H. Hypothesis*

Based upon previous research by others and also our experiences playing against each of the four AI, we hypothesize that the Minimax AI agent will ultimately be the most effective of the four AI agents. Following this agent will be the Particle Swarm Optimization (specifically, trailing by 30 or less total wins). We predict that the Expectimax algorithm will come in third place, and that the random AI will come in last. This will be due to the fully-deterministic nature of the Minimax algorithm, as opposed to the PSO and random agents. Furthermore, the Minimax agent runs under the assumption that its opponent is playing optimally, unlike the Expectimax algorithm. We predict that Expectimax's assumption about its opponent and its lower depth will prove to be a disadvantage in this game.

## IV. Results

The results of having the AIs play against one another, in the manner described in the previous section, are shown in the tables below.

### A. *Results of Tournament*

TABLE I. RESULTS OF MINIMAX VS RANDOM

| ***Minimax Wins*** | ***Random Wins*** | ***Ties*** |
|---|---|---|
| 99 | 1 | 0 |

TABLE II. RESULTS OF PSO VS RANDOM

| ***PSO Wins*** | ***Random Wins*** | ***Ties*** |
|---|---|---|
| 99 | 1 | 0 |

TABLE III. RESULTS OF EXPECTIMAX VS RANDOM

| ***Expectimax Wins*** | ***Random Wins*** | ***Ties*** |
|---|---|---|
| 97 | 3 | 0 |

TABLE IV. RESULTS OF EXPECTIMAX VS MINIMAX

| ***Expectimax Wins*** | ***Minimax Wins*** | ***Ties*** |
|---|---|---|
| 7 | 93 | 0 |

TABLE V. RESULTS OF EXPECTIMAX VS PSO

| ***Expectimax Wins*** | ***PSO Wins*** | ***Ties*** |
|---|---|---|
| 43 | 57 | 0 |

TABLE VI. RESULTS OF MINIMAX VS PSO

| ***Minimax Wins*** | ***PSO Wins*** | ***Ties*** |
|---|---|---|
| 64 | 36 | 0 |

TABLE VII. TOTAL WINS

| ***AI Agent*** | ***Total Wins*** |
|---|---|
| Minimax | 256 |
| PSO | 192 |
| Expectimax | 147 |
| Random | 5 |

### B. *Discussion of Results*

As can be seen in “Table VII.”, the Minimax with alpha beta pruning algorithm achieved the highest number of wins, and we therefore deem it the most effective of the four algorithms that we implemented, followed by PSO, Expectimax, and finally the random AI. It is not exactly known why the random AI was able to win five times total, but we suspect that this is likely due to a fault in the heuristic function, since the Minimax algorithm is deterministic and with a depth of 6 should be able to beat the random AI every time. We can think of two main reasons as to why Minimax was able to beat Expectimax and PSO:

*1) Fully deterministic:* The Particle Swarm Optimization algorithm contains elements of stochasticity, while the Minimax algorithm is fully deterministic. That is, if placed in the exact same scenario one million times, the minimax algorithm will make the exact same choices 100% of the time. However, this is not the case for PSO, as its slight stochasticity will sometimes cause it to make non-optimal moves.

*2) Different depths:* The Expectimax AI was set to a depth of four, while the Minimax AI was set to a depth of six. These were the maximum depths with which, given our computing power, the algorithms could make decisions within a time frame of less than 10 seconds. In most cases, these algorithms will make better decisions given a higher depth, and our results corroborate this.

*3) Assumptions about opponent:* As explained earlier, the Minimax runs under the assumption that its opponent is playing optimally and makes decisions accordingly. Expectimax, however, does not run under this assumption, and instead runs on the assumption that there is an element of chance in its opponent’s decisions. In this particular game, there is no element of chance (other than the forced random first moves), so therefore the Expectimax algorithm is at a disadvantage in its decision-making.

## V. CONCLUSION

To summarize, we implemented four different AIs and subjected them to direct comparison by means of a tournament-like Connect Four competition on an 8x8 board. We implemented Minimax with alpha-beta pruning with a depth of six, Expectimax with a depth of four, Particle Swarm Optimization, and a random AI. We used the same heuristic function for Minimax and Expectimax and the particle fitness function used by the Particle Swarm Optimization AI was nearly the same as this heuristic function.

We had each AI play 100 games against the other three AIs for a grand total of 600 games played throughout the entirety of the tournament. Ultimately, we found that Minimax w/ alpha-beta pruning won the highest number of games (256 wins), with PSO coming in second place (192 wins), Expectimax in third (147 wins), and random in last (5 wins). Our original hypothesis predicted these final rankings, though we originally predicted that the Minimax and PSO win totals would be closer. We mainly attribute this outcome to the deterministic nature of the Minimax algorithm as well as the fact that in this implementation has a higher depth than our Expectimax implementation. Nonetheless, the PSO agent did fairly well, and proved itself to have legitimate potential for use in games such as this one. Further work can be done in tweaking its parameters to achieve a more optimal outcome.

## VI. FUTURE WORK

We have several ideas for what we would work on and experiment with if given more time and computing power. The following are some of our ideas:

*1) Tweak PSO Parameters:* Further experiment with the c1 and c2 values in the PSO particle update formula in an attempt to improve the effectiveness of this AI in Connect Four.

*2) Increase Expectimax Depth:* Increase the depth of the Expectimax agent to six to see how it performs against the Minimax agent of the same depth. Furthermore, experiment with depths higher than 6, or with Expectimax at a higher depth than Minimax.

*2) Implement more complex algorithms:* Expand the tournament to include implementations of highly complex AI algorithms, such as Deep Q-Learning or AlphaZero, which have both been successfully created for Connect Four.

*3) Run more games at each stage of the tournament:* Run the tournament with a much larger number of games at each stage, such as ten thousand. This would give an even clearer idea of how much these AI differ in their effectiveness.

*4) Further increase the size of the board:* Increase the size of the board even further (since we already increased it from 7x6 to 8x8) up to a size such as 16x16 to see what effect this has on the tournament results.

*5) Gather more data for deeper analysis of AI:* Keep track of which types of wins the given AI most commonly achieves (e.g., horizontally), which columns it most often picks, etc.. This data could be used to analyze the different playing styles of various algorithms.

## References

[1] G. Vandewiele, "Creating the (nearly) perfect connect-four bot with limited move time and file size," *Medium*, 28-Nov-2017. [Online]. Available: https://towardsdatascience.com/creating-the-perfect-connect-four-ai-bot-c165115557b0. [Accessed: 06-May-2020].

[2] "Connect Four," *Wikipedia*, 15-Apr-2020. [Online]. Available: https://en.wikipedia.org/wiki/Connect_Four. [Accessed: 06-May-2020].

[3] G. Wisney, "Deep Reinforcement Learning and Monte Carlo Tree Search With Connect 4," *Medium*, 25-Apr-2019. [Online]. Available: https://towardsdatascience.com/deep-reinforcement-learning-and-monte-carlo-tree-search-with-connect-4-ba22a4713e7a. [Accessed: 06-May-2020].

[4] A. Young, "Deploying an AlphaZero-powered Connect Four AI with GraphPipe," *Medium*, 04-Sep-2018. [Online]. Available: https://medium.com/@sleepsonthefloor/deploying-an-alphazero-powered-connect-four-ai-with-graphpipe-d2b088619562. [Accessed: 06-May-2020].

[5] V. Allis, "A Knowledge-Based Approach of Connect-Four," *ICGA Journal*, vol. 11, no. 4, pp. 165–165, Oct. 1988.

[6] S. Sengupta, S. Basak, and R. Peters, "Particle Swarm Optimization: A Survey of Historical and Recent Developments with Hybridization Perspectives," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 157–191, Oct. 2018.