Connect Four

Jacob Frericks

Abstract

Games have always been interesting when it comes to artificial intelligence, but Connect Four is specifically interesting. This game has the opportunity for many different algorithms to lead to a solution (such as the random method or using other artificial intelligence algorithms) but the game is simple enough to easily understand each algorithm. In this study, it was decided to use the MiniMax algorithm to determine what the best move for the agent would be. The advantage of this algorithm over, say, the random method is it's effectiveness at choosing a decent move, while keeping a reasonable time and space efficiency.

One of the major challenges discussed in this paper is how much the MiniMax algorithm is dependent on your heuristic. Generally, a bad heuristic will cause a bad move by your agent. Before research was done, it was believed that the algorithm choice was the most important part of the agent. However, it has been concluded that the heuristic is just as important as the algorithm, due to the above reasons.

1 Introduction

This section will discuss the rules of the game Connect Four as well as the goal of this study.

1.1 Rules

The game Connect Four is a simple two player game. Its based on a game board that is 7 columns across and 6 down. There are two colors (red and black) of pieces that indicate which player owns which piece (similar to Tic Tac Toe's X and O).

Each player will rotate in turns. This means that no player can go twice in a row, and because of this it is required to decide who will go first. In standard Connect Four games, red goes first, so Player 1 is always assigned to red, while Player 2 (or the AI, if single player mode is chosen) is assigned to black. In this study, "r" stands for "red" and "b" stands for "black".

A standard turn consists of a player (either Player 1 or Player 2) putting their respective piece into a column. Gravity will then take that piece as far down in the game board as possible. Once a player has done this, both players check to see if there are any four pieces of the same color in a row/column/diagonal. If so, then the game is over, and that player has won. If not, then the next player takes their turn. If all slots are full and there are no four in a row lines, then it is a draw.

1.2 Goal

The goal of this study was to give the developer an easy and fun way to learn about the different AI algorithms. This was done by the creation of an AI agent that could play the game Connect Four against a human player. While the rules were taken from the physical games directions, the source code for this agent was created from scratch.

Since an agent is being created to play this game, it is necessary to define the PEAS for this agent. Please see Table 1 for this description.

Type	Description
Performance	See heuristic
Environment	Game board and player's pieces (Red and Black)
Actuators	Placement of the pieces
Sensors	N/A

Table 1

2 Algorithms

This section will describe two main algorithms (Random and MiniMax) and describe their advantages and disadvantages.

2.1 Random

The random algorithm is chosen if the player decides to play on the "Super Easy" level. This algorithm randomly

chooses a number from 1-7 and places it's piece in that slot.

This algorithm has several advantages. First is it is the easiest to create. The agent will simply randomly choose a slot to put a piece in, regardless of the situation. The second advantage is the time complexity. Since the algorithm is simply randomly picking a column to put the piece in, this is done in nearly constant (O(1)) time.

While the advantages of this approach make this algorithm pretty attractive, there are some major disadvantages to this approach. The main one is that, as mentioned earlier, this approach doesn't consider the situation of the board. If Player 1 has 3 in a row and has potential to win next turn, the agent will have the probability 1/7 of blocking that win. This is a pretty terrible heuristic, and this algorithm is usually disregarded entirely. In this study, it was decided to keep it in the agent, as a player may want to play against an easy opponent.

2.2 MiniMax

The MiniMax algorithm is chosen if the player decides to play on the "Easy" level. This approach assumes that Player 1 will do what is best for him, and so the agent can anticipate and react to that move. A tree is created of the possible moves, and the agent will go through the ones that would best suite the player, and chooses what would be the best move to react to that. The agent goes through a certain depth of the tree, which was implemented to allow for further difficulty levels at another date. For instance, if the difficulty "Hard" was to be implemented, we could just increase the depth of the tree that the algorithm is to be going through.

The MiniMax approach has several advantages. The main advantage is that this algorithm is situation (heuristic) based. This means that (given the same example as with the Random algorithm) if Player 1 has three in a row, then this algorithm will be smart enough to know that the best move for Player 1 is to place his piece into the column that would give him four in a row. So the algorithm will know to place it's piece in that column instead. See section 2.2.1 for more details.

This approach has several disadvantages, however. The main one is time complexity. The normal time complexity for MiniMax is O(b^d) where b is the branch factor and d is the depth. In the case of Connect Four, this is simply O(7^d), since there are seven columns, and therefore 7 options to choose from (which means the branching factor is also seven). This means that it is a huge time complexity (exponential), and according to Ron Adams, Erik Ibsen, Chen Zhang in their paper "Smart Connect Four" wasn't even feasible in 2003:

> This algorithm was initially going to use the minimax algorithm to search the game tree for the optimal move. However, the tree proved to be too massive to search in this way, due to computational limitations.

Obviously the advances in computers have expanded the computational limits by large amounts in the last 9-10 years, but another thing that allows this algorithm to be used is the fact that I don't search through the entire tree, but only up to a certain depth.

This study was originally creating the MiniMax tree iteratively. However, as mentioned in an article from an unknown author on Purdue University's website, the MiniMax algorithm could easily be written as a recursive method. This is the method that was eventually used.

2.2.1 Heuristic

A section of the MiniMax that is not considered in the Random algorithm is the heuristic. The heuristic is how the algorithm is able to determine what is a good move versus what is a bad move. The heuristic that was chosen to be implemented is strictly defensive. This heuristic will run on each possible move, and is determined by counting how many of the opponent's "2 in a row" exists on the board (once again, going horizontally, vertically and diagonally) and multiplying that number by a certain fixed number. It does the same with how many of the opponent's "3 in a row" exists and multiplies that number by another (larger) fixed number. The program will then add those two numbers together, and that sum is the heuristic. In this case, the larger the heuristic, the more likely it is that the agent will chose that move.

3 Future Improvements

This section talks about the several interesting expansions to this study that could be implemented in the future.

3.1 Heuristic Improvements

Heuristic improvements is probably the biggest thing that could be done to improve the agent's progress in the game. While this was out of the scope of this study, there are several ways to implement a better heuristic. The heuristic that was implemented in this study was strictly defensive.

You can implement a "strictly offensive" heuristic, where the agent would attempt to get his four in a row as soon as possible, regardless if the player has the possibility of getting their four in a row next turn.

Another heuristic (and probably the best) is a combination of offensive and defensive. Perhaps it implements the defensive move if the player has three in a row, otherwise it implements the offensive.

3.2 GUI

Another interesting expansion on this study could be to create a graphical user interface (GUI). Since this study was focused on algorithms, the GUI didn't play a big factor. However, this would be a very welcome improvement, as not many people like to play games via command line anymore.

3.3 Alpha-Beta pruning

The downfall of the MiniMax algorithm is the time complexity. If it got to the point where it took noticeably long for this algorithm to complete, the Alpha-Beta pruning algorithm could cut down on several branches of the tree. This algorithm is when you know that you have the best answer from a certain section of the tree, so you skip over the rest of the tree and continue elsewhere (therefore "pruning" the tree).

4 Results

This section describes the results of the Random algorithm and the MiniMax algorithm

4.1 Random's Results

As you may guess, Random didn't fair so well against the human players. In face, it lost nearly 100% of the time. This was expected, since the Random algorithm simply chooses a random number from 1-7 and puts it's piece there. This algorithm doesn't look at the board to block a four in a row, and can't tell if it could win by placing a piece in column x. All moves have 1/7 chance of being chosen.

When playing against itself, however, it did a little better (although not by much). It turns out that the both agents won about 20%, but about 60% of the time it was a draw. This isn't too surprising, since there is no heuristic in this algorithm.

4.2 MiniMax's Results

Since the heuristic that was implemented was strictly defensive, it was thought that this algorithm will most never win, but the human player won't win either. However, the way the defensive algorithm was implemented didn't account for the "trap moves" that a player can make. A trap move is when you have two different "three in a rows" so if the other player blocks one, you can still connect four pieces in the second "three in a row", and therefore win the game. It turns out, however, that these trap moves are hard to implement correctly against another player, but the heuristic could be fixed to account for this. Overall, however, the player won 35% of the time, the agent won 2% of the time, but the rest was a draw.

It was also thought that if you put two agents with a defensive heuristic against each other, that the majority of the games would end in a draw. It turns out this is correct. While different agents won in some games, it ended up to be about equal (~5% wins each)

4 Conclusion

The main goal of this study was to give the developer a simple and fun way to study and research different artificial intelligence algorithms. This goal was most effectively accomplished. It was learned that although MiniMax is the most effective algorithm of those studied, it also has the largest time complexity, and is heavily dependent on the heuristic you develop along with it. It was also learned that, while the Random algorithm has the best time complexity (since it runs in constant time) it plays absolutely terrible when playing against a human.

5 References

Adams, Ron, Erik Ibsen, and Chen Zhang. A Connect Four Playing AI Agent: Algorithm and Creation Process. N.p., 15 Dec. 2003. Web. 4 Dec. 2012.

"Artificial Intelligence: A Modern Approach (3rd Edition) [Hardcover]." Artificial Intelligence: A Modern Approach (3rd Edition): Stuart Russell, Peter Norvig: 9780136042594: Amazon.com: Books. N.p., n.d. Web. 04 Dec. 2012. <http://www.amazon.com/Artificial-Intelligence-Modern-Approach-Edition/dp/0136042597>.

"Department of Computer Science, Purdue University." Department of Computer Science, Purdue University. N.p., n.d. Web. 04 Dec. 2012. <http://www.cs.purdue.edu/>.