Ellis Yarboi

Sameer Rau

CSCI 4511 Project Report

Introduction and Description

Connect 4 is a two-player strongly solved connection game (a strategy game where players attempt to complete a connection with pieces) in which the players first choose a color and then take turns dropping colored discs from the top into a seven-column, six-row vertically suspended grid with over 4.5 trillion possible positions (see figure 1). The pieces fall straight down, occupying the first available space within the column from the bottom. The objective of the game is to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally before the opponent does. The game comes with perfect information, so that a player on any given turn has access to all the previous moves made by them and their opponent. The game can also be classified as zero-sum, as a move that a player makes impacts the outcome of the next move (a player's advantage is an opponent's disadvantage). This can be seen from the very first move, as the first player can always win by playing the right moves such as beginning by dropping a piece in the middle column. In this regard, the game can resemble that of tic-tac-toe, in which the object is to connect three of one's piece vertically, horizontally, or diagonally as well as the fact that it comes with perfect information and is a zero-sum game.

The first solve of the game using any form of artificial intelligence was John Tromp's 8-ply database in 1995, which is considered to be a fairly brute method based on the more sophisticated techniques used to construct the AI. Figure 1 below shows his findings (so far) for

the number of possible positions in each cell of the 6x7 grid. Today, most artificial intelligence algorithms are able to solve Connect 4 using minimax and/or negamax (a type of minimax algorithm that uses the aforementioned zero-sum property of a two player game), with alpha-beta pruning providing a more efficient way of finding the best possible move to obtain a player's maximum gain. Alpha-beta pruning is the algorithm typically chosen to write most Connect 4 AI algorithms, due to the fact that it is much faster at finding potentially good moves in a shorter time period. It usually runs ten moves ahead of the current state to determine best possible moves for the future to put itself in a position to win. However, it is not the most efficient way in playing a competitive match against a human opponent; that would be the Monte Carlo tree search (became more widely used in 2006), which caters specifically to games such as Connect 4 because many times the board gets close to full capacity where one player is virtually forced to concede as they have to grant the other player the win. The Monte Carlo tree search, which is also a heuristic search algorithm, can be applied to other games as well, such as video games and nondeterministic games such as poker (in which a player cannot see what other players are holding at any point - it is partially observable).

Our project aims to replace a human player of Connect 4 with a rational AI program.

height							
8	9 13343	8424616	1104642469				
7	8 4587	1417322	135385909	14171315454			
6	7 1571	235781	15835683	1044334437	69173028785	4531985219092	
5	6 537	38310	1706255	69763700	2818972642	112829665923	
4	5 179	6000	161029	3945711	94910577	2265792710	54233186631
3	4 58	869	12031	158911	2087325	27441956	362940958
2	3 18	116	741	4688	29737	189648	1216721
1	2 5	13	35	96	267	750	2118

1 2 3 4 5 6 7 8 width

Figure 1: Number of possible positions in a Connect 4 grid, as calculated by Tromp.

PEAS description and task environment

When generating a rational agent to play against, certain properties of the program need to be examined. As with any agent discussed throughout the course, we must evaluate what it considers as its environment, how it observes the environment (via sensors), how it acts on it (via actuators) and what variables determine an agent's success, and how it can build on these variables to continue to improve itself (known as the performance measure). Additionally, we must consider the how it comes to the conclusion that a move is the most efficient and correct one (in conjunction with its performance measure).

While playing Connect 4, the game grid is deemed the major component of the environment. It is the stage upon which all actions occur between players. Similar to that of a tennis court in which a match is played, with the tennis racket and ball acting on the surface as actuators, in Connect 4 the court is represented by the grid with the pieces acting upon it (filling in the grid). By definition, this makes the game pieces represented by two different colors as actuators, as they are the ones performing the action in this scenario in order to reach a final result (win, lose, or draw). In a player to player competition, the sensors are represented by human vision (eyes) to guide where it might be a smart move to place a piece in a certain column, and where it decreases the chance of a win. The performance measure is updated based on how close to the goal of reaching four in a row, column, or diagonal has been reached. In the case that the opponent has reach closed to this milestone, the performance measure will be

shifted towards preventing the other player from winning the game. Of course, this evaluation gets updated at the completion of every turn as the choice of moves change as well.

A task environment evaluation describes several components regarding the relationship between the agent and how it reacts with its surroundings. First, it checks whether an agent in a given outcome is observable or not by all players involved. Next, it sees the number of players involved in the event, and classifies it as a single-agent or multi-agent affair. Then, it determines whether the game is deterministic, stochastic, or strategic based on whether there is a final outcome or not. The following check looks to see whether the progression of events within the game is episodic or sequential. After that, whether the environment is static or dynamic, that is, whether changes are being made to the environment during the course of the game. Finally, the task environment evaluation look at whether the surroundings are discrete or continuous based on the number of possible places to move around in the environment. This number could be infinite (continuous) or be restricted to a certain value (discrete).

As it relates to Connect 4, the game is fully observable, as after a move has been made by a player, the move can be seen for the remainder of the game up to its completion. It is a multi-agent game consisting of two players who are each trying to maximize their gain at each point but also minimize their losses at the same time. The game is deterministic, as one move by a player impacts the move for the other player the following turn. In the case of the opponent having three consecutive pieces in any of the aforementioned combinations (row, column, diagonal) the player cannot put their piece in any random place. This also results in the game being classified as sequential rather than episodic, as one move is dependent on the next rather

than them being independent entities (although this is not totally clear at the early stages of the game). Connect 4 is considered to be a static because the environment itself is not changing; during a person's turn the other person cannot always plan ahead to place a piece in a certain location before the previous turn is over as the landscape of potential moves could change at the conclusion of the other player's turn. Finally, the game of Connect 4 can be classified as a discrete task environment, as defined by the constraint satisfaction problem there are only 42 possible variables in the game (representing each of the cells in the 6x7 grid). Therefore, there are only a certain number of places a piece can be placed before the board is completely full. At any given turn, there are at most seven possible moves (depending if any columns have been completely filled). Thus, Connect 4 can be summarized as a fully observable, multi-agent, deterministic, sequential, static, and discrete task environment.

Domains and constraints

The game of Connect 4 can be best described in the form of a constraint satisfaction problem. The number of variables represented in Connect 4 is simply the dimensions of the grid where a piece by either player can be placed. Since the grid consists of seven columns and six rows, there are 42 possible variables. The domains, or the possible outcomes of each variable by the game's completion, can have three possibilities: either a game piece representing the first player, a game piece representing the second player, or a blank where no player has placed a piece. The constraints lie within staying within the bounds of the grid (this is more of a programming concern than an actual real life version of the game). Entering a row or column that does not fall between the grid boundaries would result in an out of bounds error, as a variable that does not exist beyond the currently existing 42 is trying to be accessed. Moreover, a

piece cannot lay floating in any column; once a piece has been dropped into a column it must go to the bottom-most possible row such that all the spaces below the piece (if any) have a player piece. Likewise, once a column is full, the player must look for a different column to put a piece in. These two conditions are both shown below in figure 2. By using these two constraints, it prevents a player from attempting to put multiple pieces in a cell that already contains a piece. A tie results if all 42 variables have a non-blank status, that is, every space has a player piece. This is seen below in figure 3.

2 (before)	**2 (after)**	**Not allowed**
0	2	1
2	2	2
1	1	2
2	2	0
1	1	1
1	1	0

Figure 2: An almost filled column 2 to a fully filled column 2, as well as an invalid combination, where 0 denotes a blank space, 1 represents Player A's move, and 2 represents Player B's move.

0	**1**	**2**	**3**	**4**	**5**	**6**
2	2	1	2	2	2	1
1	1	1	2	1	1	2
2	1	1	1	2	2	2
1	2	2	1	1	1	2
1	2	1	1	2	2	1

2	1	1	2	2	2	1

Figure 3: A completely filled 6x7 Connect 4 grid without a winner, resulting in a tie. 1 represents Player A's move, and 2 represents Player B's move.

Pseudocode

As mentioned above, a win by a player is determined by checking the rows, columns, and diagonals. Below is the pseudocode we wrote at the preliminary stages of the project to obtain whether a player has won or not for rows and columns, respectively (before AI implementation). For the rows, the check for consecutive pieces starts looking to the left of the last placed game piece, where match is by default set to 1 representing the currently tracked number of consecutive pieces found. If this piece matches the last placed piece the match count is incremented by one and the search continues to look left, with this process continuing until a different piece is found, a blank cell has been found, or four matches have been found. If the match count reaches 4, a player is crowned the winner. Once a different piece has been found, the program returns to the last placed game piece, and the checks begin to the right of the game piece in a similar process. If the match count never reaches 4 after these checks, it has been determined that the player has not got four pieces in a row. The column check is slightly easier, as the last placed piece would represent the topmost piece for four consecutive pieces found in a column. Therefore, the check only needs to go in one direction; that is, checking for the pieces below the last placed piece. In a similar fashion to the row check, the match count starts at 1 by default representing the last placed piece. If the piece residing in the same column but lower row matches the previous placed piece, the number of matches increases by 1. Otherwise, another player piece means four consecutive pieces in a column have not been found and a winner has not yet been found (assuming the board is not already full).

Row check:

int match = 1; //where match is the number of consecutive pieces in a row. At any point it will be at least one, which is the last piece placed.

The matrix[row][column] represents location of last placed piece.

```
placedlocation = matrix[row][column];

left_column = column-1;

for(int i=0; i<matrix.length; i++) {
        while((matrix[row][left_column] == playerpiece) && (column >=0)) {
                match++;
                new_column = column-1;
        }

        if(matrix[row][column-1] != playerpiece) { //where playerpiece = player's turn
                        //reset to original location here
                        while((matrix[row][column+1] == playerpiece) && (column<=6)) {
                                match++;
                        }
                }
        }

        if(match>=4) {
                System.out.println("Congrats player" + playerpiece);
        }
}
```

Column check:

```
for(int i = 0; i < 4; i++)
{
        if( (matrix[row-i][column] == playerpiece) && (row - i >= 0) )
                match++;
}

if(match >= 4)
        gameOver = true;
```

Minimax Algorithm and Game Trees

Adding the AI implantation into the project, we used the minimax search heuristic to plot the AI's move based on where the player has placed a piece on the board. At the beginning of the game, the moves by the computer are relatively random, as there is little information to work with, given that there are very few pieces and not much of an opportunity for the AI to improve its performance measure given the player's moves. The branching factor, by default, starts at seven columns. If the column gets filled up, as seen in figure 2, the branching factor is decreased by one. Of course, a check for a win does not begin until the seventh move (combined between the player and computer), as a player cannot win until they have played a minimum of four game pieces. From a non-technical standpoint, the AI chooses where to place a game piece based on where it has chances to win, except for when the human player is about to win. Looking at it in the form of a more technical game tree, each possible path is given a relative weight that determines how "beneficial" that move is to the computer. A positive weight indicates that the move helps out the AI in getting four consecutive pieces in a row, column, or diagonal, a 0 represents a neutral move that does not optimize the computer's chances of winning, and a negative value represents a move where the player is about to win the game. The negative weight takes precedence, of course, over a path that helps out the AI in order to prolong the game and keep the human from winning. This in turn improves the computer's performance measure as well over the course of the game. As mentioned earlier, the branching factor represents the number of potential moves that a player or computer can make, depending on the number of filled up columns. Therefore, a single path will be more efficient in playing the game at certain point than traveling down others. Looking four levels ahead, each path has a scaled down weight of 1/level. Depth first search is used to travel down each path to the lowest level. When the recursion travels back up the game tree, the parent node has its value updated based on whether it

is a max/min, as in a typical minimax algorithm. The result, when the recursion is complete and reaches the root node helps the computer determine what move to make and where to place a piece on the grid.

Possible future work

There is enormous potential for future work. Not only can the AI get more sophisticated and smarter using alpha-beta pruning within this specific project (to cut down on the number of paths that it has to check) and in general apply these techniques to other games such as tic-tac-toe, backgammon, and various card games (such as poker), the design of the computer from this project can be applied to the premise of robotics and other applications that revolve around the subject of "how to get an artificial intelligent system to think along the lines of a human". IBM's Watson did precisely this on a trial run on *Jeopardy*, effectively constructing a database of knowledge and weighting how often an answer comes up in a search result and how useful the key word is. The current artificial intelligence agent that we have constructed can be placed in other settings in the future with tweaks to the program (of course), such as making tough decisions that we as humans have to make by looking at the possible outcomes of decisions, looking ahead to possible consequences afterwards, and assigning each individual path a weight not to different from the system used to place a piece on the game board. The agent is practically a utility-based agent that weighs its options before making an optimal decision in reaching a certain goal. Thus, in the future, a potential plan would be to build on the existing capabilities of the artificial intelligent system, and get the AI to handle more complex activities with the overall goal in mind to bridge the gap between the abilities of a human and the abilities of a computer by getting it to think more like we do in our everyday lives.

Bibliography and references

Tromp, John. “John’s Connect Four Playground.” Github, 2008. Web. 10 May 2016.

Allis, Victor. “A Knowledge-based Approach of Connect-Four The Game Is Solved: White Wins.” *Department of Mathematics and Computer Science Vrije Universiteit* Amsterdam, Netherlands October 1988.