Spring 2012

Teaching Computers to Think: Analysis of Artificial Intelligence and Connect Four

Kirk Baly
Valparaiso University

Andrew Freeman
Valparaiso University

Andrew Jarratt
Valparaiso University

Kyle Kling
Valparaiso University

Owen Prough
Valparaiso University

Kirk Baly, Andrew Freeman, Andrew Jarratt, Kyle Kling, Owen Prough
Prof. Greg Hume
Research in Computer Science (CS 492)
Spring, 2012

Teaching Computers to Think: Automated Analysis of Connect Four

Abstract
Connect Four is a classic two person, zero-sum game in which players utilize their wits and gravity to connect four of their own pieces in a horizontal, vertical or diagonal row while blocking their opponent's attempt to do the same. We have constructed a simulation of this game, which we have used as a base for the implementation and testing of varying Artificial Intelligence (AI) systems. Early strategies worked according to simple strategic methods, while more advanced heuristics employed a Min-Max Tree in tandem with methods to determine how advantageous a certain board would be. This Min-Max Tree goes beyond a simple strategy, as it allows for the computer to look many moves ahead, thus picking the move that optimizes its chances of winning. The collection of statistics for the various strategies has allowed for the analysis and improvement of the AI structures.

Introduction

We developed a game based on Milton Bradley's published game of Connect Four. Although Connect Four has existed in essence for quite some time, it was popularized under this name upon publication by Milton Bradley in 1974. A two-player game, the objective of Connect Four is to connect four of your tokens horizontally, vertically, or diagonally before your opponent does the same. Players take turns dropping one of their tokens into a column, at which point it falls to the bottommost available row in that column. The standard board is 6 rows by 7 columns.

We intended to create a model of the board such that the initial height, width, and number of connected tokens to win could be specified by the user at runtime. This would make analysis of winning (or losing) strategies possible given a variety of game parameters. Our goals included creating a program to play Connect Four, and to develop gameplay strategies in order to allow the game to be played by the computer against either human or computer controlled opponents.

Methods

We began by creating an abstract model of the game in Java which allowed for two players to play a game of Connect Four. During development, we followed the model-view-controller pattern to make our code as modular and easy to maintain as possible. We also hosted our code in an SVN repository, to allow concurrent development and ensure that everyone was always using the latest version of the code. Once we had these set up, we turned our attention to the design and implementation of artificial intelligence strategies.

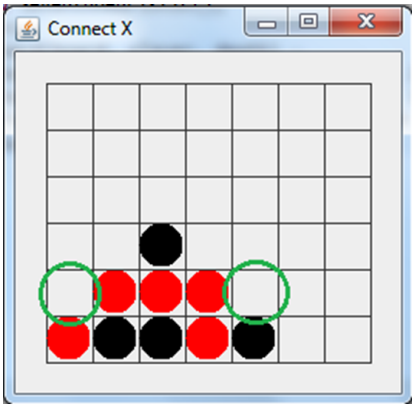
The first strategy we implemented was a player who selects a column at random to move in. This was trivial to implement, but provided a jumping-off point for more advanced strategies. To improve this strategy, we added the following set of rules:

1. Make a winning move if one exists

2. If your opponent will win on their next turn, move so as to prevent this loss.
3. Do not make a move which allows your opponent to win on their next turn.

Further improvements were seen by adding the rule “If possible, move in a way which gives you two possibilities to win so that your opponent cannot prevent your victory.” A possible scenario involving this fourth rule is seen in Figure 1.

Figure 1.



We then branched off into using MinMax trees. A MinMax tree is used in AI systems to take a board and determine which move will be most advantageous. It uses a heuristic to assign a value to a move based on the principle of minimizing the potential to lose and maximizing the potential to win. Each tree has a depth that represents the number of moves in advance that it uses to evaluate the current move. We designed our MinMax tree implementation in a way which allowed us to simply plug in new heuristic evaluation functions as we developed them. We developed and implemented the following heuristics:

- First: Ranks a board based on whether or not the board is a win, loss, or an immediate set-up for an opponent victory.
- Second: First Heuristic + takes into account the depth of the board within the MinMax tree
- Third: Second Heuristic + takes into account the number of series of three consecutive tokens
- Fourth: Third Heuristic + considers the depths of the series of three tokens
- Fifth: Fourth Heuristic + gives value to the number of wins a board has
- Sixth: Fifth Heuristic + looks for a situation in which a win is guaranteed, where a threat has two adjacent, win-inducing available slots. A threat is a series of three consecutive tokens which could potentially result in a win either immediately or later in the game
- Seventh: Sixth Heuristic + looks for a situation in which two consecutive tokens have both adjacent slots open, in order to possibly set up for an automatic victory
- Eighth: Seventh Heuristic + subtracts the value given for the opponent with heuristics 4, 5, 6, and 7 in order to judge the other player’s performance with a board

Realizing that we were developing new ways to rank boards without having a good idea of what criteria were the most important, we decided to design and implement a genetic algorithm to “evolve” the proper weight values for us. A genetic algorithm is an optimization technique that uses a fitness function to attempt to find the best value for a variable over many iterations, in a manner that mimics natural selection. We used a genetic algorithm to perfect the weights assigned to each of the different patterns evaluated in the eighth heuristics.

The algorithm used a special heuristic which we called the 'genetic heuristic' in which a weight value, an integer between 0 and 20, could be assigned to each of the patterns we were interested in. Initially, we generated a large number of these genetic heuristics, each with its own randomly assigned value for each of the weights. Then, we competed each of the heuristics against the enhanced random strategy and assigned each heuristic a value depending on how well it performed. The best performing heuristics were then used to form a new collection of genetic heuristics. We also added heuristics with random weight values into the collection, in order to widen the scope of possible values. In repeating this process, we were able to find a set of weight values that approached the optimal weight value for each of the patterns.

Conclusions

We successfully created a program which allows for game parameters to be set at runtime, including selection from a variety of strategies for computer-controlled players. Using this program, we were able to analyze each strategy's performance. Based on the strategies and testing methods we used, we found that, given a single strategy played against itself, the player to move first won between 3 and 7% more games than the player who moved second.

Figure 2 shows the optimal set of weights our genetic algorithm produced. The genetic heuristic, using these weight values, outperformed eighth heuristic, which looked for the same patterns but with unadjusted weight values. This may indicate which patterns are better to look for and which ones are not necessarily important. Ergo, the low weight value we extracted for the pattern present in the fourth heuristic indicates that simply looking for the number of series of three consecutive tokens is not substantially advantageous. The higher weight values given to the patterns in the sixth and seventh heuristic shows us that it is more beneficial to look for two or three consecutive tokens with both adjacent sides open and ready for an immediate token placement. The highest weight value we extracted was for the pattern present in the fifth heuristic, which indicates that looking for boards with multiple wins is the most advantageous method among the four patterns we examined.

Figure 2.

Pattern Introduced by Heuristic	Weight Value
Fourth	4
Fifth	19
Sixth	13
Seventh	12

Future Work

One improvement that could be made to decrease run-time and memory usage in order to more

quickly and efficiently obtain results is to improve the overall design of our MinMax trees. A logical method for achieving this would be utilizing alpha-beta pruning, which involves “trimming” parts of the tree that are known to contain non-optimal solutions. We have already put forth some effort to avoid reconstructing the entire tree each time a new move is needed, so that we only need to generate one layer of children each time.

As part of our data analysis on each improving heuristic, data was kept on the time elapsed for a set number of game iterations. One potential area for future investigation is the Big-O notation for each heuristic. A genetic algorithm was used to determine weights for each heuristic, however using Big-O notation would be a standard for assessing the speed of each heuristic in the MinMax tree.

Once we have a more efficient and faster way to generate trees it will become easier to work with our genetic algorithm, which at the moment is computationally expensive. From this we will be able to obtain more ideal values for our weights in a faster time and also be able to analyze any additional patterns we may be interested in.

Also, it may be interesting to analyze how strategies perform when the win condition is to connect any number other than four tokens, or having a board of different dimensions than the conventional 6x7.

References

Allis, Victor**.** *A Knowledge-based Approach to Connect-Four: The Game is Solved: White Wins*. October 1988: Vrije Universiteit. Amsterdam, The Netherlands. Web.