The Design and Analysis of an AI for the Game *Conniption*

Project by: Amir Khadivi, Harshkumar Patel, Kifah Issa

Introduction and Rules:

In Conniption the goal is to get 4 chips of the same type in a row vertically, horizontally, or diagonally. What makes conniption unique from Connect Four is that winning in Conniption can also be accomplished through flipping the board. Each player is allowed 4 flips of the board, and with this additional option of flipping the board before and/or after a move, the maximum possible number of moves from a given state becomes 29. It is important to note that if a player ends in a flip the next player cannot start the move with a flip. Furthermore, it is not known how many moves are typically done on average to end a match in Conniption, but assuming each player does, for example, 3 moves, there are 29^6 or 594,823,321 possible states in this game tree. Therefore, to design a sufficiently challenging AI that can play this game without frustrating the user because of its slow search speed, we need to cut down the possible number of states available in our search space at our reasonable depth limit (d=6). To make this happen we turn to the classical *minimax* algorithm with an implementation of *alpha-beta pruning* to find a solution without needing to traverse the entire 6-ply search space.

AI and Experimental Design:

We used simple data structures for most of the major parts of our AI. For example, we used linked nodes with references to the parent nodes as the data structure to create relationships between the nodes (to form a tree). The total state of a node is characterized by a series of descriptions: an array representing the board, an action (represented by a String) that was used to create the current board from the previous one (such as "chip", "chip/flip", "flip/chip/flip"…etc.), a reference to the parent node, and if the board was brought to its new state by the insertion of an 'X' Chip or 'O' Chip. With this representation of a node we utilized a *minimax* algorithm with *alpha-beta pruning* to provide the best possible move for the AI.

To utilize a *minimax* algorithm with alpha-beta pruning it was necessary to create a utility function, and several evaluation functions to estimate the final value of the game (produced from a utility function) from a given state. Below the various evaluation functions are described:

EVAL1:

This evaluation function utilizes a map of values representing the number of possible ways to win from a given position on the board. For every "O" on the board the function sums the values from the map that correspond to the position of the "O"s. This is done for "X" as well and is subtracted from the total sum for "O"s. This difference plus an offset factor (138) to prevent negative numbers is what is returned by the function. The map of the values is shown below:

3	4	5	7	5	4	3
4	6	8	10	8	6	4
5	8	11	13	11	8	5
5	8	11	13	11	8	5
4	6	8	10	8	6	4
3	4	5	7	5	4	3

Figure 1: Map of Values Used for EVAL1

HYBRID1:

HYBRID1 uses the evaluation function EVAL1 plus a combination of two other evaluation functions to produce a final value. The two other heuristics utilized in the HYBRID1 evaluation function is the heuristic (GOAL function) that scores a state with 4 in a row for "O" and/or "X". This heuristic gives a high positive value for a state with four "O"s in a row and a very negative value for a state with four "X"s in a row. It provides a value of 0 for a tie. Another function utilized (IS3INROW function) is one that measures the number of "O"s and "X"s in a row with consecutive blank spaces open allowing for a win. For example, it is important to measure the number of "O"s and "X"s that are 3 in a row with an open space right after. We hypothesized that it is also important to count the number of "X"s and "O"s that are two in a row with 2 consecutive blank spaces after, as well. We decided to return a value for this function using the equation (considering AI is using O and Human is using X).

(eqn. 1) IS3INROW = NUM3O – (NUM3X+NUM2X).

The following combination of functions ultimately produced a reasonably challenging AI.

(eqn. 2) HYBRID1 = EVAL1 + GOAL + 4*IS3INROW;

HYBRID2:

HYBRID2 uses the evaluation function HYBRID1 plus a function (topstack function) that prioritizes putting chips at the top of each column on the board. We hypothesized that this function would help prevent any of the chip/flip or flip/chip combinations that make the AI lose repetitively. It is important to note that HYBRID2 uses different coefficients for the functions that it is composed of in comparison to HYBRID1. We tried to optimize the weights of the functions by trial and error (see eqn. 3).

(eqn. 3) HYBRID2 = EVAL1 + GOAL + 2*IS3INROW + TOPSTACK

HYBRID3:

HYBRID3 is almost same as HYBRID2, with little modifications. It includes **Flip Penalty**. The purpose of using Flip Penalty is the observation that we made, at some point of the game AI was wrongly using flip, where it should just put chip in the board. Considering maximum number of flips during the game, using flip is obviously expensive. While using flip penalty, we consider that flip penalty just prevents AI from using flip where another move (not using flip) with the same utility value is available. So, AI will use flip where it is appropriate to use it. In HYBRID3, we also changed the weights of IS3INROW, (considering AI is using O and Human is using X)

(eqn. 4) IS3INROW = ((3 * NUM3O) + NUM2O) – ((3 * NUM3X) + (2 * NUM2X)

After changing weights so many times we realize that this was perfect equation.

(eqn. 5) HYBRID3 = EVAL1 + GOAL + IS3INROW + TOPSTACK – FLIP_PENALTY

NOTE: We changed the weights while doing different experiments, so the weights might not be the same.

Data and Results:

To test the AI's various evaluation functions we designed experiments where the AI would face another AI with the same or different evaluation function, depending on the experiment type. See below for brief description of experiment and corresponding results.

Experiment 1: AI1 ("X") Vs. AI2 ("O") Percentage Wins

AI1\AI2	EVAL1	HYBRID1	HYBRID2	HYBRID3
EVAL1	54(X):45(O)	38(X):62(O)	44(X):54(O)	39(X):61(O)
HYBRID1	63(X):37(O)	51(X):47(O)	40(X):(58)(O)	41(X):59(O)
HYBRID2	43(X):57(O)	29(X):71(O)	32(X):68(O)	42(X):57(O)
HYBRID3	40(X):60(O)	37(X):63(O)	34(X):66(O)	37(X):63(O)

Figure 2:
Note: The (r, c) position indicates the function of row (r) plays first and function of column (c) plays second. For example at r = 2, and c =3, we have the game HYBRID1 vs. HYBRID2 with HYBRID1 playing first as "X". In the simulation of *AIvsAI "X" always plays first and "O" plays second*. If the percentages do not add to 100% then it is due to tied matches. *Number of games played per matchup, N = 100.*

Experiment 2: Move Ordering and Number of Nodes Examined in *Alpha-Beta Pruning*

MOVE ORDERING LIST 1:
actionc(n);
actioncf(n);
actionfc(n);
actionfcf(n);
actionf(n);
# of nodes explored: 472,605

MOVE ORDERING LIST 2:
actionf(n);
actionc(n);
actioncf(n);
actionfc(n);
actionfcf(n);
# of nodes explored: 375,754

As seen from the two move ordering lists and the number of nodes explored by the *minimax* algorithm with *alpha-beta pruning*, the order in which children nodes are created has an effect on the number of nodes explored be the algorithm.

Discussion:

The data from experiment 1 shows that the hybrid evaluation functions performed better than the non-hybrid evaluation function, EVAL1. When EVAL1 faced EVAL1 the first player had a slightly higher chance of winning the match. Furthermore when EVAL1 would compete against HYBRID1, it would lose most of the time regardless of whether it would play first or second. Against HYBRID2, EVAL1 would win 57 out of 100 games if HYBRID2 went first, but would only win 44 games if HYBRID2 played second. This result was rather confusing since playing against HYBRID2 as a human is more challenging than playing against EVAL1 by a long shot. We hypothesize that this weakness in HYBRID2 may be due to the random picking of the best possible moves returned from the *minimax* algorithm. For example, when playing the AI, the first, best possible move is returned by the *minimax* algorithm, not a random, best move from a list of equally valued moves. Although we do not have any data for Human vs. AI games, we believe the HYBRID2 evaluation function produces a better evaluation function from our experience playing the game. This is the limitation of playing AIs with different evaluation functions against each other. The simulations do not provide much insight in how the AI fares against a human. However, from our Experiment 1 data and experience playing the AIs we believe that the hybrid evaluation functions do perform better than the non-hybrid EVAL1 function.

The data from experiment 2 confirmed the notion that move ordering[1] is critical when determining how many nodes are pruned from a given search tree. For example, with move ordering list 1 (see page 3) 472,605 nodes are explored while with move ordering list 2 375,754 nodes are explored. More nodes are pruned with move ordering list 2 than move ordering list 1. At a depth of 6 (6 ply depth search), a maximum of 29^6 (594,823,321) possible nodes exist; therefore, *alpha-beta pruning* is effective in decreasing the search space.

[1] By move ordering I mean the order in which child nodes are created from a parent node.

Individual Involvement

Amir Khadivi:

I worked on coding the “minimaxalg” class, which is essentially the AI. This class is what utilizes minimax with alpha-beta pruning. I also worked on modifying various evaluation functions such as HYBRID1 and HYBRID2. However, I did not create HYBRID1 or EVAL1. I was the sole designer of HYBRID2 with the new heuristic function “topstack” which gives priority to placing chips on top of each column of the board. Harsh developed the AIvsAI method, and I developed the original HumanvsAI method. Although Harsh created the AIvsAI method I implemented the experiments to get the data presented in this paper. Harsh also implemented the original Humanvs.Human method. I also am responsible for writing most of this report and presentation. I believe Harsh and I worked very well together on this project even though there were times where we could not understand each other. He is a good teammate.

Harsh Patel:

I started working on project by first creating the basic game (Human vs. Human version) including the rules for the games and also command line user interface. Amir was much interested in minmax and alpha beta pruning. So we decided he would work on Human vs. AI. Using the basic Human vs. Human version of the game, Amir worked on Human vs. AI version (minmax and alpha-beta pruning and maintaining rules of the game for AI). Meanwhile, I was working on evaluation functions. I created HYBRID1 evaluation function that’s actually a combination of three different evaluation functions. I also coded HYBRID3 evaluation function that is slight variation of HYBRID2 and includes Flip Penalty. At the last, for experimental purposes I coded AI vs. AI version of the game in that I made some changes to Amir’s minmax code so that it can work for AI vs. AI. I also wrote some part of the report and suggested what to include in presentation to Amir. I can say I and Amir has worked very well with each other. Just for this repost purpose, we are dividing the work otherwise I would say we both know each other’s work very well and we helped and motivated each other throughout the project. I would say it was great that Amir was my teammate.

Kifah Issa:

For my part in the project I created the GUI. The GUI only displays the conniption board. It updates every time the board status is changed. I just drew red, black, and white circles to implement it and put an orange square behind each one to create a background.

GUI:

GUI was developed very late, so we didn’t have enough time to implement it in our code. Below is the snapshot of the GUI that was developed for very old version of our code.