# Intro to A.I. Topics

Connect Four

March 12, 2009

## Introduction

Connect Four is a tic-tac-toe like game in which two players drop discs into a 7x6 board. The first player to get four in a row (either vertically, horizontally, or diagonally) wins.

The game was first known as "The Captain's Mistress", but was released in its current form by Milton Bradley in 1974. In 1988 Victor Allis solved the game, showing that with perfect play by both players, the first player can always win if he plays the middle column first, and if he chooses another column first the second player can always force a draw.

Today we will explore the different strategies involved in playing connect four, how a computer could emulate these strategies, and how these techniques relate to other artificial intelligence topics involved in solving games with large search spaces.

For convenience, we will call the first player white (W) and the second player black (B).

Note that we initially get somewhat detailed about game situations, but do not get bogged down in the details. The important part is how we will *use* the fact that these details exist to make a game-winning strategy.

## Solvability

When looking for a strong solution to a game (recall from last time that a strong solution means knowing the outcome of the game from any given board position) one strategy to try would be storing all possible game positions in a database, exploring the tree of game play from each position, and determining the winner in each case. We will see that this strategy, at least at this time, is not really feasible for Connect Four (and even much less so for more complex games like GO and chess...).

First we look for an upper bound on the number of possible Connect Four board positions. Each grid square can be in one of 3 states: black, white, or empty. Since there are 7x6 = 42 squares, this gives a very crude upper bound of $3^{42} \geq 10^{20}$. A not so much closer look reveals that we can get a much tighter upper bound by noticing that many positions we counted before were illegal. For instance, if one square is empty in a column, then all the squares above it must also be empty. So we throw these possible positions out. Removing these configurations gives a better upper bound of $7.1 * 10^{13}$ possible positions.

There are other types of illegal positions that are harder to detect. For instance, if we are assuming that white moves first, then some game configurations, such as a stack in one column from bottom up of BWBWBW is impossible, since black would have had to move first. It turns out that no one has been able to weed out all of these positions from databases, but the best lower bound on the number of possible positions has been calculated by a computer program to be around $1.6 * 10^{13}$. So we would need *at least* that many positions stored to do a brute force search of the game. That would take an estimated 4 Terabytes of memory. Not so practical...

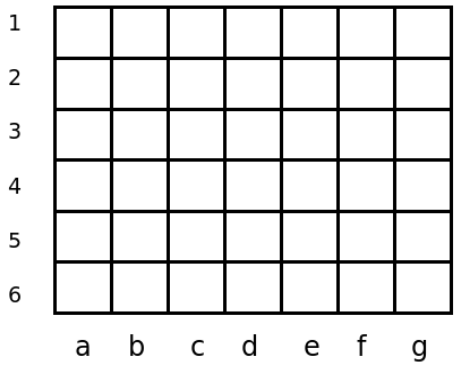
As we saw last time with the tic-tac-toe example, this strategy won't always work. In general, we must classify these sorts of rules into two classes:

- Rules that guarantee a certain results (and require proof that they do so)
- Heuristic rules that are generally advantageous but are not without downfall (like the strategy given above)

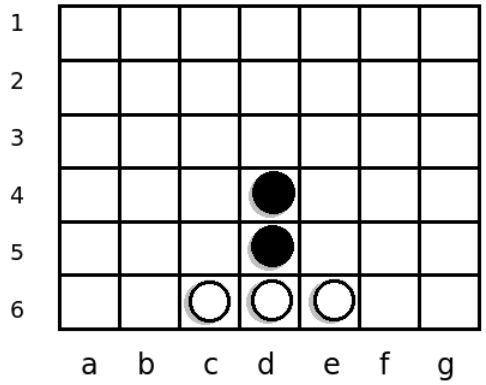
We'll explore possible strategies to follow for Connect Four below. After describing a set of general rules to follow, we will see a new type of search, conspiracy number search, related to the pn-search technique from last week.

First we'll learn some terminology associated with describing the strategy.

Nomenclature We will number the 7 x 6 board with columns a to g left to right and numbers 1 to 6 from bottom to top. So the coveted middle bottom square is d1.



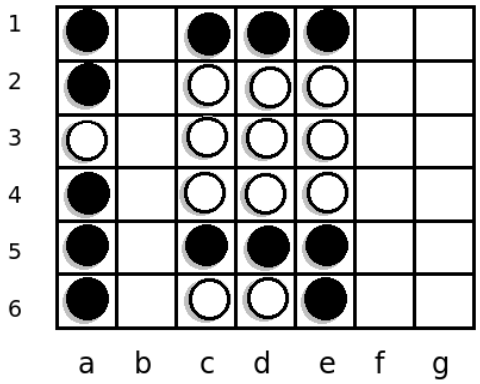

Threat A threat is a square that if taken by opponent forms a game-winning group of four. For example, in the game board below White has threats at b1 and f1.




Useless Threat A useless threat is a threat that will never be able to be carried out

by the opponent. Note that a threat can only be carried out if the opponent is forced to play the square below the threat.

The picture below illustrates the concept of a useless threat. In this game, it is White's turn to move. It appears that White has threats at b2, b3, b4, b5, b6, f2, f3, f4, f5, and f6. But Black has threats at b2, b6, f2, and f6. Clearly the lower numbered squares will be filled in first, and so since both White and Black have threats at b2 and f2, no other threats matter, since they are all above threats shared between players. So all squares but b2 and f2 are useless threats.




Odd and Even Threats

It is clear that a threat can only be carried out if the opponent is forced to play the square below (or he allows you to play below the threat). In analyzing threats, certain patterns show up in how the squares are dividing among players.

The odd/evenness of a threat is determined by the row number. So d1 is an odd threat, etc. If we were to just fill up the board, for the most part, White will get the odd squares and Black will get the evens. Clearly, White starts with an odd square (1). Say we have filled up the entire board except the last column. If play continues until the board is filled up, it must be true that White will get the remaining odd squares and Black the remaining evens: we must end with Black, since play alternates between players and there are an even number of squares, and so black must get the top (6), white must get 5, etc. This even/odd pattern continues throughout the game in general (of course it is possible for White to get *some* even squares and Black to get some

odds). So if we get to the last column and White has an even threat and Black has an odd threat, the game will end in a draw.

If White has an odd threat and Black has an even threat in the same column, the lower threat will win. If the threats are in different columns, White's is stronger. In general, we see the following patterns:

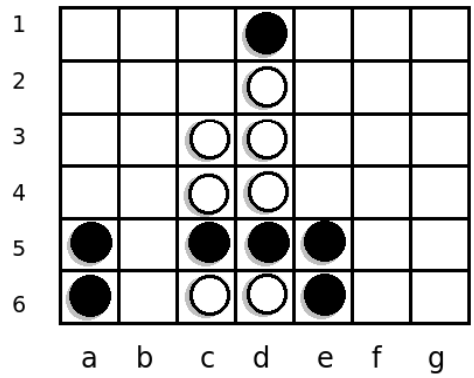
- White has an odd threat, Black even: White wins
- White and Black both have even threats: there is no column where an odd number of squares can be played, so both players will get their normal squares (as defined above), and Black will be able to refute White's threat and win.
- White has an even threat, Black an odd threat: draw.
- White and Black both have odd threats: usually neither of these threats end up working and depend on other threats.

In a careful analysis of threats it is important to make sure that taking care of one threat does not allow another threat to be created.

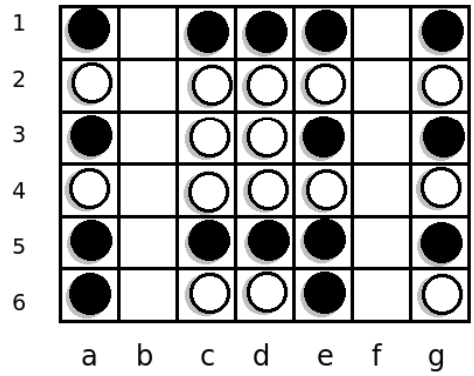
Zugzwang The formal definition of this strange German word: a situation where a player is forced to make a move when he would rather make no move at all.

In connect four, a player is able to "control the zugzwang" if the player is able to guide the way odd and even squares are divided up among players.

As an example, we look at the following game situation (Allis 26), where White is about to move:




Note that all columns contain an even number of pieces, so White will never fill up a column since it must take only odd squares. So Black can just play "follow-up" and mimic White's every move. This will result in the following position:




Now White must play either b1 or f1, which Black will follow, and win the game with a group of four on the second row.

So in conclusion, Zugzwang involves being able to divide up how even and odd squares are distributed to the two players. Black wanted only even squares because eventually it would be able to fulfill its threat at either b2 or f2. But if it had wanted odd squares, it could have just stopped playing follow up and played in a different column.

## Rules

As an example of using a knowledge based approach to teach a computer to play a game, the following rules were used in programming VICTOR to

win connect four. Each rule classifies threats and gives solutions to some of them. Each rule is valid for the player that controls the Zugzwang, which is assumed to be black in the following examples. Each of these "rules" is a possible winning connection for the player.

Claimeven Controller of zugzwang can get all empty even squares which are not directly playable by letting the opponent play all empty odd squares.

Required: Two squares, directly above each other. Both squares are empty, the upper square must be empty.

Solutions: All groups which contain the upper square.

Baseinverse Based on the fact that a player cannot play two directly playable squares in one turn.

Required: Two directly playable squares

Solutions: All groups which contain both squares.

Vertical Based on the face that a player cannot play two men in the same column in one turn, while by playing one man in the column, the square directly above becomes immediately playable.

Required: two squares directly above each other. Both squares empty, upper square must be odd.

Solutions: all groups which contain both squares

Aftereven Side-effect of one or more claimevens. If a player in control of zugzwang can complete a group using squares from claimeven, he will eventually be able to finish the group.

Required: a group which can be completed by the controller of the zugzwang, using only squares of a set of claimevens.

Solutions: all groups which have at least one square in all aftereven column,s above the empty aftereven group in that column. Also, all groups which are solved by the claimevens.

Lowinverse Based on the face that the sum of two odd numbers is even.

Required: two different columns, each with 2 squares lying directly above each other. All must be empty and the upper square must be odd.

Solution: All groups which contain both upper squares, all groups which are solved by verticals.

Highinverse Based on the same principle as lowinverse:

Required: Two columns which 3 empty squares each, upper square is even.

Solutions: all groups which contain the two upper squares, groups which contain the two middle squares, all vertical groups which contain the two highest squares of one of the highinverse columns

If the lower square of the first columns is directly playable: all groups which contain both he lower square of the first column and the upper square of the second.

If the lower square of the second column is directly playable: all groups which contain both the lower square of the second column and the upper square of the first column.

Baseclaim Combination of two basinverses and a claimeven.

Required:Three directly playable squares and the square above the second playable square. The non-playable square must be even.

Solutions:Solutions: All groups which contain the first playable square and the square above the second playable square. All groups which contain the second and third playable square.

Before Based on a combination of claimevens and verticals

Required: A group without men of the opponent, which is called the Before group. All empty squares of the Before group should not lie in

the upper row of the board.

Solutions:All groups which contain all squares which are successors of empty squares in the Before group. All groups which are solved by the Verticals which are part of the Before. All groups which are solved by the Claimevens which are part of the Before

Specialbefore A version of the before

Required: A group without men of the opponent, which is called the Specialbefore group. A directly playable square in another column. All empty squares of the Specialbefore group should not lie in the upper row of the board. One empty square of the Before group must be playable.

Solutions: All groups which contain all successors of empty squares of the Specialbefore group and the extra playable square. All groups which contain the two playable squares. All groups which are solved by one of the Claimevens. All groups which are solved by one of the Verticals.

## Computer Solution Implementation

Victor Allis's program VICTOR developed a method of finding an optimal strategy based on the 9 rules given above. The position evaluator (white or black) is given a description of the board and comes up with an optimal next move.

First all possible instances of the nine rules above are found and checked against all 69 possibilities to connect winning groups. The rule applications that solve at least one problem are stored in a list of solutions with a list of the groups solved by the solution and a list of other solutions that can be used to solve the problem.

The next step is finding which solutions can work together. First all so-

lutions are assembled as nodes into an undirected graph, where two nodes are connected if and only if they can't be used simultaneously. These connections are stored in an adjacency matrix. Then, the problems (threats) are added as nodes, and solutions are connected with problems if they solve the problem (no problems are connected).

Note that two rules might not necessarily be able to be used at the same time. The following table describes the relationships between rules (taken from Allis, section 7.4):

CL = Claimeven, BI = Baseinverse, VE = Vertical, AE = Aftereven, LI = Lowinverse, HI = Highinverse, BC = Baseclaim, BE = Before, SB = Specialbefore.

| | CL | BI | VE | AE | LI | HI | BC | BE | SB |
|---|---|---|---|---|---|---|---|---|---|
| CL | 1 | | | | | | | | |
| BI | 1 | 1 | | | | | | | |
| VE | 1 | 1 | 1 | | | | | | |
| AE | 1 | 1 | 1 | 3 | | | | | |
| LI | 2 | 1 | 1 | 1&2 | 4 | | | | |
| HI | 2 | 1 | 1 | 1&2 | 4 | 4 | | | |
| BC | 1 | 1 | 1 | 1 | 1&2 | 1&2 | 1 | | |
| BE | 1 | 1 | 1 | 3 | 2&3 | 1&2 | 1 | 3 | |
| SB | 1 | 1 | 1 | 3 | 2&3 | 1&2 | 1 | 3 | 3 |

1. Combination allowed if the sets of squares are disjoint.
2. Combination allowed if no Claimeven is below the inverse.
3. Combination allowed if sets of squares are column wise disjoint or equal.
4. Combination allowed if the sets of squares are disjoint, and the set of columns used by the inverses are disjoint or equal.

Then we solve the following problem:

Given: Two sets of nodes, $S$(olutions) and $P$(roblems). Try to find an independent subset $C$ of $S$ with the property that $P$ is contained in the set of all neighbors of $C$, $B(C)$. (Note this is a potentially NP-complete problem)

The following recursive algorithm was used by Allis:

```
FindChosenSet(P, S)
{
```

```
    if (P == EmptySet) {
           Eureka() /* We have found a subset C which meets all constraints. */
    } else {
           MostDifficultNode = NodeWithLeastNumberOfNeighbours(P);
           for (all neighbours of MostDifficultNode) {
                  FindChosenSet(P - { MostDifficultNode },
                                S - AllNeighboursOf(ChosenNeighbour));
           }
    }
}
```

## Conspiracy Number Search

The actual solution to connect four was arrived at using the above methods combined with search tables storing the values of different game positions as well as using traditional game search methods. In particular, Allis used **conspiracy-number search**, which is very closely related to the proof number search that we talked about last week. In the instance of connect four, consider three types of nodes: -1 black can at least draw the game, 1 the game is a win for white, or 0 the game is as yet undecided. Now any node that has as a child a node with a value of 1 can be colored 1. Any node that has all nodes colored -1 can be colored -1. Note it is much easier to change a node to a 1 than a -1.

The conspiracy number of a node is a tuple of counts for the number of children needed to "conspire" to change the value of the node to each of the possible values. Let $(x, y)$ be the conspiracy number of a node, with $x$ the number of nodes that need to conspire to change the value to 1, and $y$ to -1. We know $x$ will always be 1, since we only need one child of value 1 to change our value to 1. But $y$ is the number of ones of the node yet to be evaluated if all those evaluated so far have been -1. If sons have already been evaluated to 1, then $y$ is $\infty$.

The purpose of conspiracy number search, like pn-search is to evaluate as few nodes as possible to find the result of the game tree. Therefore, we try to avoid evaluating nodes with large conspiracy numbers. If we want to evaluate a node at the top of the tree, we choose the neighbors with the lowest

conspiracy numbers to evaluate until we are sure of their value. We move up the tree in this way, whenever possible avoiding evaluating nodes with large conspiracy numbers.