AI Project: Connect Four

We tried to do the same moves in order to have comparable results. With task 5b, the order of the regarded moves in the min-max-algorithm changed, however. This resulted in the AI playing other moves, so we had to change our first ten moves, otherwise we would have lost before the 10th move.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5a | 5b |
| Move 1: | 1426 | 4680 | 1208 | 1665 | 4680 | 1228 |
| Move 2: | 925 | 4680 | 771 | 1544 | 4680 | 1419 |
| Move 3: | 654 | 4680 | 574 | 1535 | 4680 | 1502 |
| Move 4: | 809 | 4623 | 745 | 1506 | 4623 | 1563 |
| Move 5: | 1221 | 4622 | 1158 | 2208 | 4622 | 1386 |
| Move 6: | 2459 | 4056 | 2007 | 2367 | 4056 | 1692 |
| Move 7: | 909 | 4544 | 854 | 1572 | 4544 | 1658 |
| Move 8: | 760 | 4256 | 736 | 1645 | 4256 | 1391 |
| Move 9: | 901 | 4240 | 852 | 1936 | 4240 | 929 |
| Move 10: | 724 | 3972 | 572 | 1577 | 3972 | 1189 |

1. Original Version  
2. Without shallow pruning  
3. Shallow pruning with <= check  
4. Deep pruning  
5a. Reordering without pruning  
5b. Reordering with deep pruning

**2. Deactivate shallow alpha-beta-pruning**

**Code**The highlighted code was commented out by us. The same if statement has also been commented out in expandMinNode.

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Explain why the outcome has to be exactly this number**

When commenting the pruning mechanism out, the min-max-algorithm will consider every solution of the given depth. With the standard depth of 4 and 8 columns for the playing board, that is a total of 8 moves for the first depth, then 8\*8 moves for the second depth, 8\*8\*8 for the third depth and 8\*8\*8\*8 moves for the fourth depth. This adds up to 4680 moves when analyzing 4 moves ahead on an 8-column playing board. This was also measured when running the application.

So, the formula is 8^1 + 8^2 + 8^3 … + 8^n. The generic formula is (8^(n+1) – 8) / 7

This is a graphical representation of the search tree:

Ein Bild, das Text, Himmel enthält.

Automatisch generierte Beschreibung

**3. Fix condition of pruning**

**Code**

The highlighted condition has been adjusted. In the expandMinNode method, the condition has been adjusted to (strength <= parentMaximum).

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**Explanation**

This change improves the efficiency, because if you find a solution that is as good as a previous solution, then you can stop the search. The reason is that this solution should not turn out better in a deeper analysis than the previous solution (given a perfect heuristic). So, you might as well take the first solution that you already have analyzed and that has an heuristic value that is just as good.

**4. Deep alpha-beta-pruning**

**Code**

*expandMaxNode (shallow pruning)*

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

*expandMaxNode (deep pruning)*

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

*expandMinNode*

**Ein Bild, das Text enthält.

Automatisch generierte Beschreibung**

**Explanation**

We changed the two Methods according to the pseudocode mentioned in the lecture. The parameters alpha and beta have been added. The original shallow pruning parameter parentMinimum has been deleted.

**5. Reordering of moves**

**Code**

The for-loop in expandMaxNode and expandMinNode have been adjusted. Now, the moves are ordered from moves in the middle to moves on the ousides.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

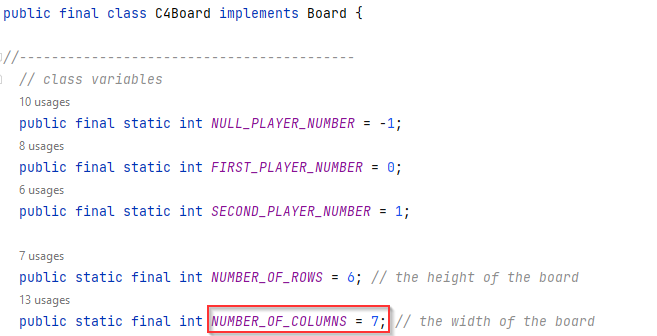
**Explanations**

Moves in the middle are generally considered to be better, as there are more possible ways to connect four. Starting the search in the middle therefore reduces the average number of visited solutions with min-max-pruning.

For task 5.a) the pruning has been deactivated similar to task 2. As seen in the table, the number of visited solutions then increases to the same as in task 2. This happens because the order of the considered moves does not matter when pruning is disabled. Without pruning, all possible solutions are considered anyways, so there is no benefit.

**6. Reduce number of columns to seven and explain the magic number**

As described in the specification, we changed the constant from 8 to 7 and confirmed that the game is still playable. The right-most column is still visible, but not playable.



**Explanation of the magic number**

Regarding the magic number of 84, this is the code where it is used:





It is important to note, that the class C4Row does not represent a row in the playing board but actually a chain of 4 slots in any direction. In other words, this vector of C4Rows contains all the possible ways of winning the gaming by connecting four slots.

That being said, the number 84 is the count of different ways to win a game of connect 4 on the given 8 \* 6 board.

Finding a formula for this number proved to be rather difficult, as this formula must consider all the chains of 4 in a horizontal, vertical and diagonal direction. For example, the horizontal chains of 4 only in the first row are the following:

Ein Bild, das Küchenutensilien, Raspel, Schreibmaschine enthält.

Automatisch generierte Beschreibung

This can be generalized to the number of columns-3. So, the horizontal chains of 4 are that number times the number of rows. This means that the formula for the number of horizontal chains of 4 is

(R-3)\*C

Where R is the number of rows and C is the number of columns. The number of vertical chains is the same formula, but the parameters are switched:

(C-3)\*R

The number of diagonal chains is more difficult to understand. We approached this by looking at a board of 4x4 and see how the total of chains changes when added another row or column. In the end we came up with this formula:

(R-3)\*(C-3)\*2

This means that the total number of chains of 4 can be calculated with this formula:

(R-3)\*C + (C-3)\*R + (R-3)\*(C-3)\*2

Of course this formula would be problematic with boards where the number of columns or rows is below 3, as there could be a negative number of chains. This can be fixed with the following formula:

= MAX((R-3),0)\* C + (MAX((C-3),0)\* R +(MAX((C-3),0)\*MAX((R-3),0)\*2))

**7. Explain the heuristic Function**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**a) What quantities does it compute? How does it weigh them?**

The first two if-guards handle the cases, in which one player has already won the game by connecting 4 slots. In this case the strength is either the highest min value or the highest high value.

If the game is still undecided, three variables as used with a certain weight. Also, the following constants are used:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

A bit shift by 5 means that a value is multiplied by 2^5. This means that the variables have the following weight:

p1\_1InARow -> 1  
p1\_2InARow -> 4  
p1\_3InARow -> 32

The variable p1\_1InARow is the count of C4Rows that have one stone from player 1 in them and no stones from the other player. Likewise, p1\_2InARow and p1\_3InARow store the number of rows where there are 2 and 3 stones from player 1 and no stone from player 2 in the row. If a row has stones from both players in it, then this row does not count to any of the three variables per player.

**How is it computed? Explain how the different objects involved communicate to produce the result?**

Each C4Row has an EventListener on each of its four slots. Each C4Row also remembers the number of stones from player 1 and player 2. When a new stone is added to the row, it adjusts the values of the six variables holding the number of uncontested stones per player. These counts are stored in an Object called C4Stats, where all C4Rows update the value.

During the min-max-algorithm, moves are played for both players when analyzing the positions in a certain depth. Afterwards, the moves are taken back by firing an Event that player NULL has made a move instead of player 1 or 2. This signals the affected C4Rows that a move was taken back and to adjust the values accordingly.

**8. Sometimes the computer can win in one move, but does not do so. Can you explain why?**

Consider this position where it is the turn of player red:

Ein Bild, das Screenshot, Kreis, Rechteck, Design enthält.

Automatisch generierte Beschreibung

First, all possible move orders with a depth of 4 are played. Those board states at the deepest level will then be compared using the heuristic function (for simplicity, pruning is ignored here).

Of course, the red player could win with only more here, but let’s assume the player decides against it and plays this:

Ein Bild, das Screenshot, Kreis, Rechteck, Farbigkeit enthält.

Automatisch generierte Beschreibung

Now it’s the move of the black player. The black player cannot really do much, but he might try to at least close one chain of 3 from the red player:

Ein Bild, das Screenshot, Kreis, Farbigkeit, Rechteck enthält.

Automatisch generierte Beschreibung

Now, it’s the turn of the red player again. By closing that other chain of 3, the red player can still win instantly. With this example, one can see that the last move of the red player did not really matter anyway, because the game can also be won in this move.

As the heuristic function does not care at which time a player wins, but only that the player wins eventually within the search depth, it will see a move that wins in one turn and a move that wins in two turns as equal.

The reason that the first column was chosen for the first move of the red player is simply that this was the first move that was considered in the first depth. Any other of the 7 possible columns has no chance to be chosen afterwards, because it cannot be better than the move on the left-most column, as this move definitely wins in the next move.