An Implementation of the Minimax Algorithm with Connect Four in Kotlin

Ajay J. Patnaik
Department of Computer Science
University of Dayton
Dayton, Ohio USA
patnaika2@udayton.edu

ABSTRACT

This paper presents a system that uses artificial intelligence to play a game of *Connect Four*. With more than four and a half trillion possible board combinations, the amount of system memory needed would be intractable. Using a depth constraint, we can reduce the amount of computations and memory needed. The artificial intelligence algorithm selected for this system was the minimax algorithm, using an alpha-beta heuristic. The algorithm looks through all of the possible board states, up to the constraint, and returns the next best move. With this implementation of the minimax algorithm, you can play a game of Connect Four against the artificial intelligence with varying levels of difficulty.

KEYWORDS

Alpha-Beta; minimax algorithm; artificial intelligence.

ACM Reference Format:
Ajay J. Patnaik. 2019. An Implementation of the Minimax Algorithm with Connect Four in Kotlin. In *Proceedings of CPS 452: Emerging Programming Languages (CPS 452).* ACM, New York, NY, USA, 2 pages.

1 INTRODUCTION

Artificial Intelligence is a field that is rapidly growing. In a society where new technology is releasing everyday, it seems that artificial intelligence is on the forefront. The system presented in this paper implements the minimax algorithm with an alpha-beta heuristic. The minimax algorithm is an effective method for achieving computer players in turn-based games. The algorithm goes back and forth on simulating the computers turn and the human turn to maximize the score of the board for the computer and minimizing the score of the board for the human. [1]

If you were to start a new game of Connect Four, where would you place your piece and why? You want to make the most out of your turn while trying to cut off the opponent. It is impossible to know where you want to play every piece before you start the game. Therefore, you recalculate what moves you want to do every time it is your turn. If you think like this, you are thinking like the minimax algorithm.

CPS 452, Fall 2019, University of Dayton, Dayton, Ohio 45469—2160

Table 1: Cell values and their representations

Cell Value	Representation
0	Unplayed position
1	User played position
2	Computer played position

2 IMPLEMENTATION

The Connect Four application uses the command line interface to get input and show output. It then starts a game loop and allows the user to play their first piece. The computer will then calculate its best move and play the piece, allowing for the user to continue the cycle. The system checks for a win after every move. Once a player wins, the game will stop.

2.1 Game Loop

The game loop is the starting point of the application. The loop continuously iterates while the game has not yet been won. For every iteration, it starts out by asking the user which column they would like to play their piece. If the column is valid, the system will place the piece in column and the turn will change to the computer. The next iteration will start and the computer will calculate the best move and play the piece. If a win occurs in any turn, the game will end.

2.2 Play Piece

At each turn, you will be presented with the current board state. Each position on the board will be either a 0, 1 or 2. The representations of these values are listed in Table 1. When you select a column to play a piece in, the system calls the `checkPiece` and the `playPiece` methods. The `checkPiece` method takes in the column and checks if the highest place in the column is filled. If the position is not filled, the system will call the `playPiece` method. The `checkPiece` method takes in the column and the player who is placing a piece. It loops from the lowest position in the column and checks if the position already has a piece in it. If it does, the next iteration in the loop will check the next lowest position in the column. The following sample code is of the `playPiece` method.

```
fun playPiece(player: Int, col: Int) {
   for(row in rows downTo 1) {
       if(state[row-1][col-1] == 0) {
           state[row-1][col-1] = player
           break
       }
   }
}
```

2.3 Computer's Decision

The computer's decision starts by finding the maximum board score the the turn. It will then find the minimum board score for when it is the human turn. This cycle continues until the depth constraint is hit. The computer will then take the most optimal route and play the piece that starts it.

2.3.1 Board Scoring. The board score is a sum of all the types of ways that the computer can place a piece. It starts with the score method. In this method, there are four loops, one for each way a game can be won. A win can come with four vertical, horizontal, diagonal with a positive slope, or diagonal with a negative slope, pieces in a row. Each way has a nested loop that iterates through each board position. In the nested for loop, the `scorePosition` method is called. This method takes in the row, column, and the orientation in which the board is currently being tested. The `scorePosition` method loops through an example of a four-piece-in-a row win. If there is computer piece in that position, it will add one score to that position. At the end of this `scorePosition` loop, it will return the value back to the nested for loop. It will then add that score to an accumulation of every score for that type of win. This method repeats for the other three ways to win. At the end of this, the system adds the four sums together and gets the board score.

2.3.2 Minimax Algorithm. Minimax is an algorithm that selects the best move by examining all possible outcomes a certain number of steps ahead. This evaluator selects a move that maximizes their chance to win or minimizes their chance to lose [2]. Although this method will find the best move, it will be computationally heavy and costs too much time. To combat this, we can use an alpha-beta approach [3]. This means that the max node will be the alpha value and the min node will be the beta value. This is illustrated in Figure 1.

The algorithm is started by calling the `maximize` method. This method loops through each column of the board. In each iteration, it creates a copy of the current board state and places a piece in column. It then calls a `minimize` method, passing in the new board. The minimize method also loops through each board iteration. Once again, a copy of the board is made and the next piece is placed in the column. Each time that `maximize` calls `minimize` and `minimize` calls `maximize`, the depth it decreased by one. This happens until the depth is zero. After this, the max move is gotten and the min move is gotten. Depending on which method you are in, it will either return the minimum or the maximum. Using these two methods together, the system can simulate the maximizing of the computers moves while minimizing the humans moves. Once the best move is gotten from the algorithm, the piece is placed for the computer.

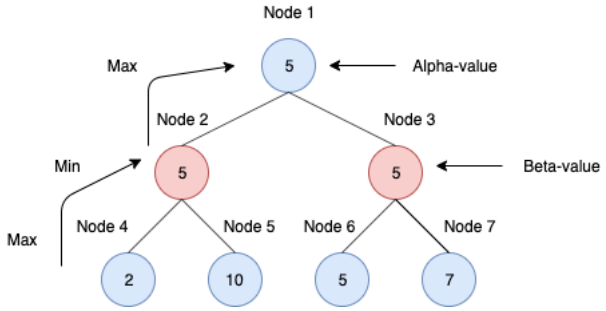

Figure 1: An example of alpha-beta pruning.

2.4 Check Win

The `checkWin` method has four parts. One for each type to win: vertically, horizontally, diagonally with a positive slope, and diagonally with a negative slope. Each way is tested by using a nested for loop. In that for loop, the four pieces necessary to win for the player are tested. For example, if the system was testing a vertical win, the system would see if row 7, column 1-4 were played by player 1. If this is the case, the game is over. This method occurs after every piece is placed. The following sample code is of the check for a vertical win.

```
for (r in state.size-1 downTo 3) {
   for (c in state[0].size-1 downTo 0) {
         if(state[r][c] != 0 &&
            state[r][c] == state[r-1][c] &&
            state[r][c] == state[r-2][c] &&
            state[r][c] == state[r-3][c]) {
            return state[r][c]
         }
   }
}
```

3 CONCLUSION

We presented a Connect Four game with an artificial intelligence player, developed in Kotlin, using the minimax algorithm with alpha-beta heuristics. The human player is able to play a computer at various difficulties. We explored the minimax algorithm and a way that we can make it more efficient. Using alpha-beta heuristics, we can more optimally get the best next more in a game Connect Four.

REFERENCES

[1] M. C. du Plessis. 2009. A Hybrid Neural Network and Minimax Algorithm for Zero-sum Games. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT '09)*. ACM, New York, NY, USA, 54–59. https://doi.org/10.1145/1632149.1632158

[2] E. R. Escandon and J. Campion. 2018. Minimax Checkers Playing GUI: A Foundation for AI Applications. In *2018 IEEE XXV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*. 1–4. https://doi.org/10.1109/INTERCON.2018.8526375

[3] S. Mozaffari, B. Azizian, and M. H. Shadmehr. 2015. Highly efficient alpha-beta pruning minimax based Loop Trax Solver on FPGA. In *2015 18th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*. 1–4. https://doi.org/10.1109/CADS.2015.7377789