



YILDIZ TECHNICAL UNIVERSITY

BLM1031 PROJECT

Skippity Game and MiniMax

Authors:

Habil Coban

habil.coban@std.yildiz.edu.tr

2023/2024 – 2nd Semester

Contents

1	Introduction	2
2	Skippity Game And Rules	2
3	Game Implementation in C	2
4	Game Bot	4
4.1	Pruning	5
4.1.1	Example Prunings	6
4.2	Example Games	7
4.3	Pc Scoring System	8
5	Conclusion	9

1 Introduction

In the course BLM1031, we were assigned a project to code the game Skippity in C. In addition to the game itself, we were required to implement a mode to play against the computer. In this report, I will explain how I implemented the game and the minimax algorithm I used for the bot as thoroughly as possible.

2 Skippity Game And Rules

Skippity game is a 2-4 player board game. The objective of the game is to capture the most sets of skippers to win. A complete set includes one of each color skipper (There are 5 colors). To setup the Board, place skippers randomly on the table except 4 squares in the center. After that you can start to play.

On your turn, use a skipper that is already on the board to jump over and capture another skipper. The skippers do not belong to any player until they are captured, so on each turn you can use any skipper that is on the board as your jumping piece.

All jumps are in a straight line rather than diagonal. This means you can jump and capture skippers that are vertically or horizontally adjacent to your jumping piece. To jump: choose a skipper, lift it over the skipper you are jumping and put it down on an empty square that is adjacent to the skipper you just jumped.

You are allowed to use a skipper to make multiple jumps (and captures) if it is possible to do so. A skipper can change direction as many times as necessary to make multiple jumps. You capture each skipper you jump over. You do not have to make a multiple jump if you do not want to.

You can read more from this [link](#).

3 Game Implementation in C

The table struct contains the pieces and score of player 1 (p1), the pieces and score of player 2 (p2), information about whose turn it is, the size of the game board (NxN), and a pointer to a 2D array where the game pieces are stored. The colored game pieces are represented by the numbers 0, 1, 2, 3, and 4 in the array, and they are displayed on the screen as A, B, C, D, and E, respectively.

```
typedef struct P
{
    unsigned short score;
    signed char piece[5];
} Player;

typedef struct T
{
    int pcScore; // Used for BOT
    unsigned short turn;
```

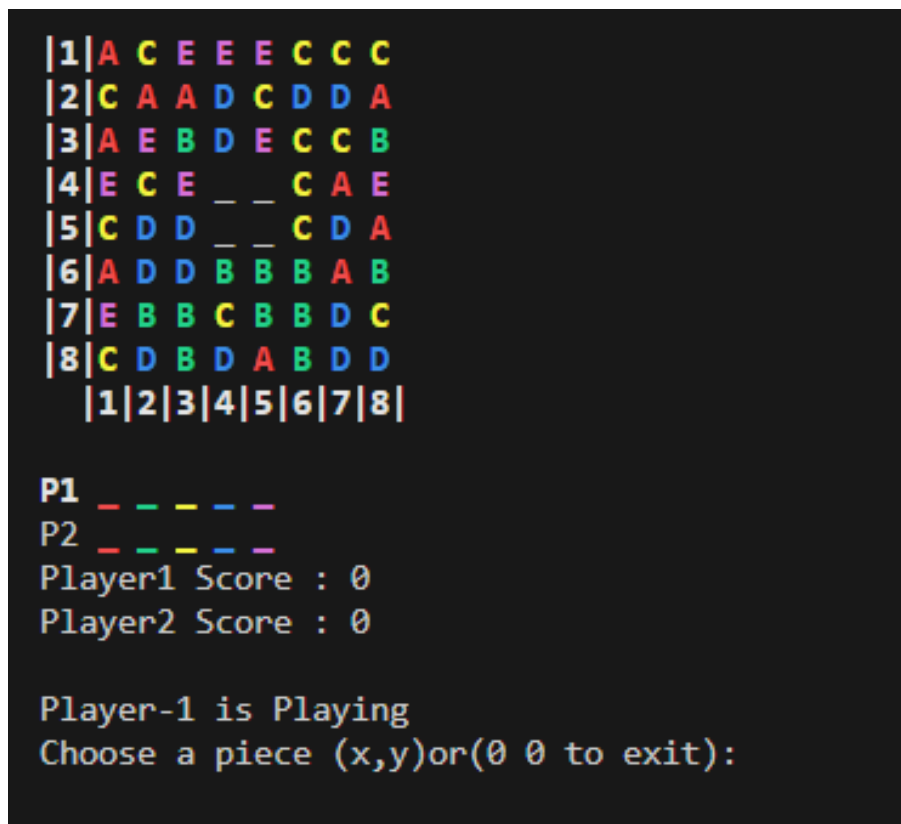
```

unsigned char gameType;
unsigned char N;
signed char** data; //signed char is 4 byte enough for our use

    Player p[2];
} Table;

```

The game has two modes: two-player and against the computer. When it's your turn to play, you need to enter the coordinates of the piece you want to move and then the direction you want to move it (W, A, S, D). As long as you can make a move, the game will keep asking you for the direction you want to move. If you don't want to continue, you can enter '1' to pass your turn.



```

|1|A C E E E C C C
|2|C A A D C D D A
|3|A E B D E C C B
|4|E C E _ _ C A E
|5|C D D _ _ C D A
|6|A D D B B B A B
|7|E B B C B B D C
|8|C D B D A B D D
  |1|2|3|4|5|6|7|8|

P1  _ _ _ _ _
P2  _ _ _ _ _
Player1 Score : 0
Player2 Score : 0

Player-1 is Playing
Choose a piece (x,y)or(0 0 to exit):

```

4 Game Bot

After some research I decided to use MiniMax Algorithm for the bot. So what is MiniMax Algorithm? The minimax algorithm is a decision-making tool used in artificial intelligence, particularly in game theory and computer chess, to determine the optimal move for a player assuming that the opponent also plays optimally.

It operates by simulating all possible moves in a game along with their subsequent outcomes, building a game tree where nodes represent game states and edges represent moves. The algorithm recursively evaluates these game states by assigning a value to each leaf node, indicating the desirability of that outcome for the maximizing player. The maximizer aims to choose the move with the highest value, while the minimizer (the opponent) aims to choose the move with the lowest value. The minimax algorithm traverses this tree, propagating values back to determine the optimal path. Alpha-beta pruning can enhance the algorithm by eliminating branches that won't affect the final decision, thus improving efficiency. This systematic approach ensures that the chosen move leads to the best possible outcome against a rational opponent.

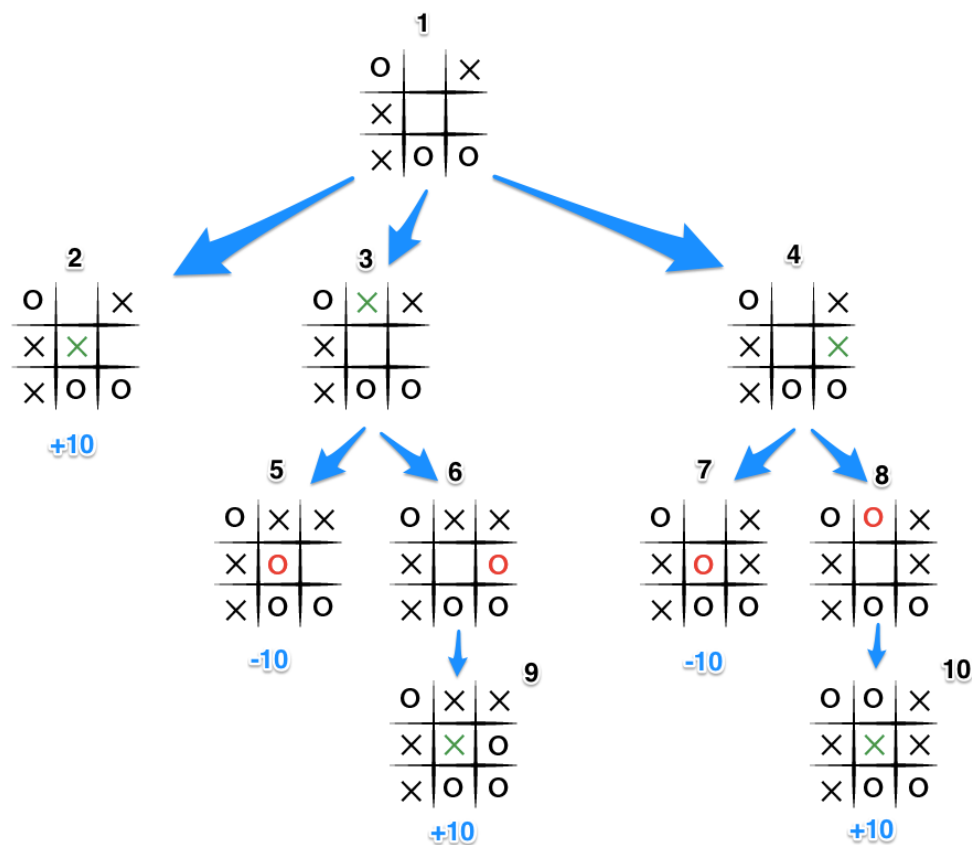


Figure 1: MiniMax for TicTacToe

4.1 Pruning

Alpha-Beta Pruning works by keeping track of two values: **alpha**, the best already explored option along the path to the root for the **maximizer**, and **beta**, the best already explored option along the path to the root for the **minimizer**. As the algorithm recursively evaluates nodes, it updates these alpha and beta values.

When evaluating a move, if the algorithm finds that the current node's value suggests a worse outcome than previously examined nodes (i.e., the value is less than **alpha** for the maximizer or greater than **beta** for the minimizer), **it prunes the subtree rooted at this node**, skipping further evaluation because it won't influence the final decision. This pruning drastically reduces the number of game states that need to be considered, thus speeding up the decision-making process without affecting the accuracy of the minimax result. In essence, alpha-beta pruning makes the minimax algorithm more practical for complex games by **eliminating unnecessary calculations** and focusing only on the most promising moves.

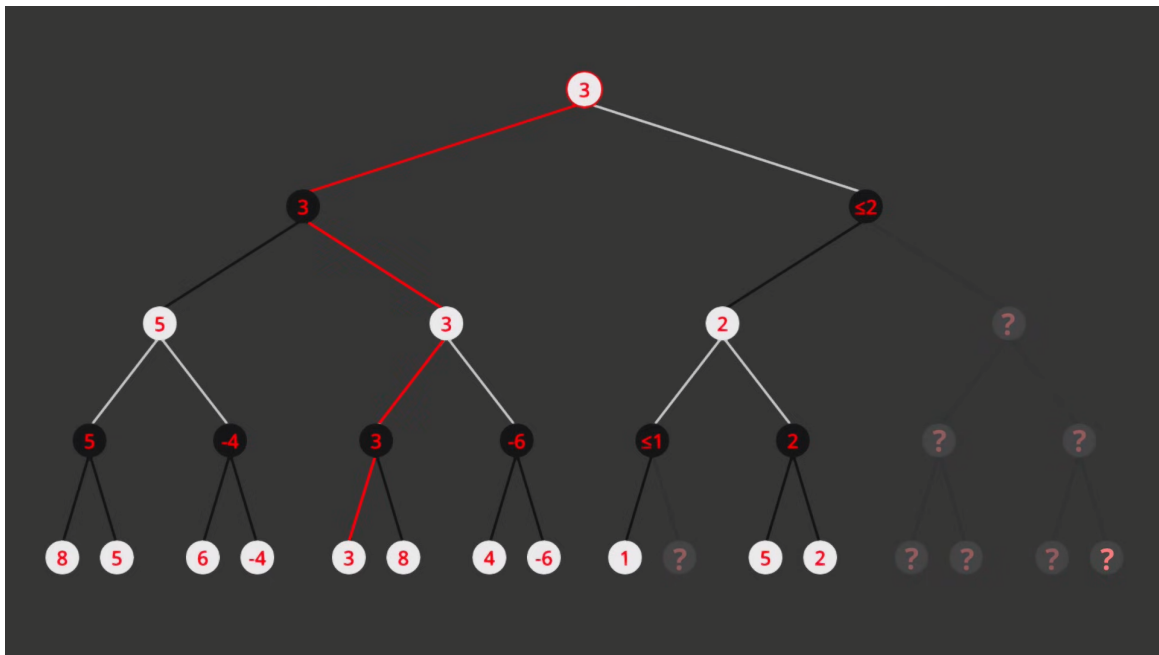


Figure 2: Alpha-Beta Pruning

4.1.1 Example Prunings

The examples are calculated on a 6x6 board with a depth of 4 moves.

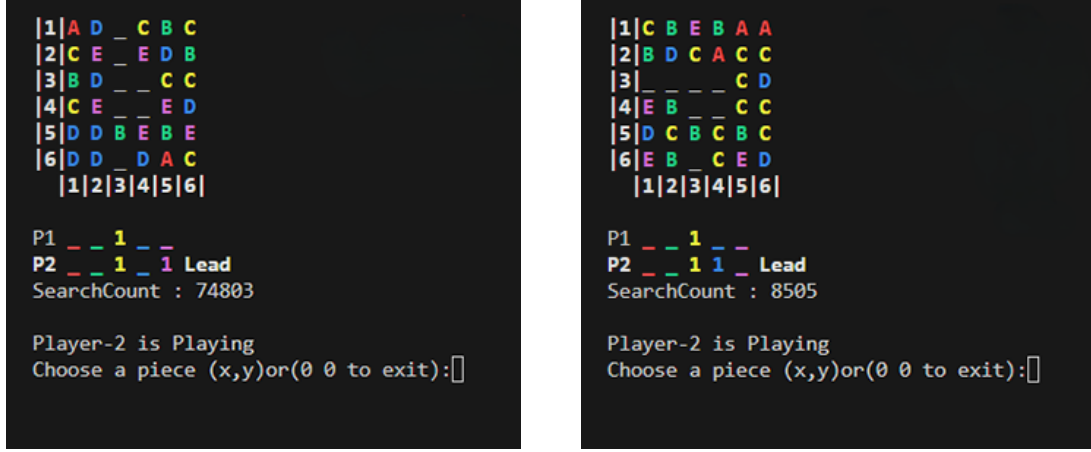


Figure 3: First Move

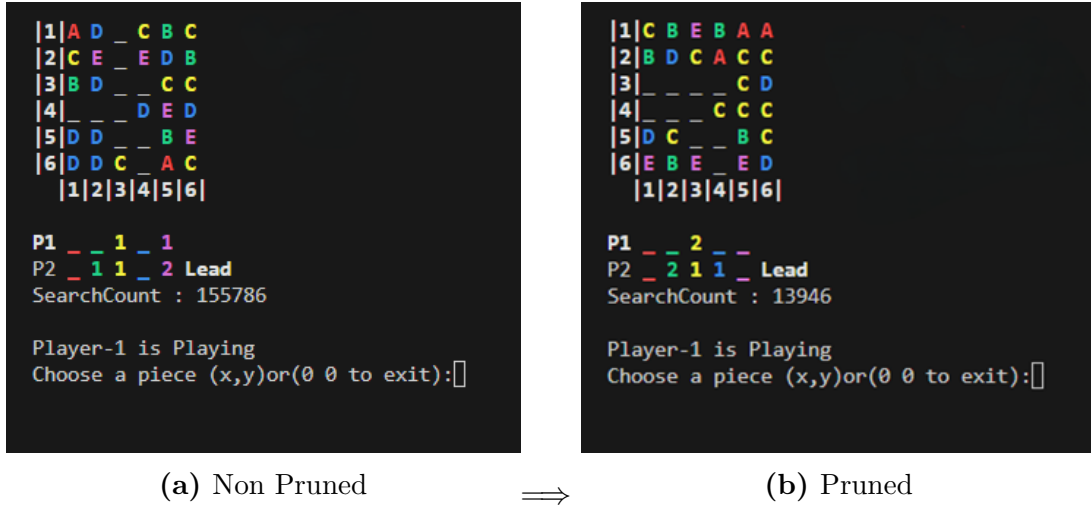


Figure 4: Second Move

As we can see from the examples, there is a significant decrease in the number of searched tables and we have made improvements in terms of performance.

4.2 Example Games

Below is a table showing that the computer has chosen a good move. The computer's moves are highlighted in white.

(a) PC Move
(4,5)→(2,5)

(b) Human Move
(1,3)→(3,3)

Figure 5: First Moves

(a) PC Move
(2,5)→(2,3)→(4,3)→(4,5)

(b) Human Move
(2,1)→(2,3)

Figure 6: Second Moves

As we can see, the computer chose another move instead of completing its own set in the first move. If its first move had been from (1,3) to (3,3)(Figure 5), there would have been an opportunity for the Player to complete his set. The computer chose the move that prevented this and managed to win the rest of the game.

Extra: Results of 100 games played by two computers against each other. Since their codes are the same, their winning distribution is approximately 50-50.

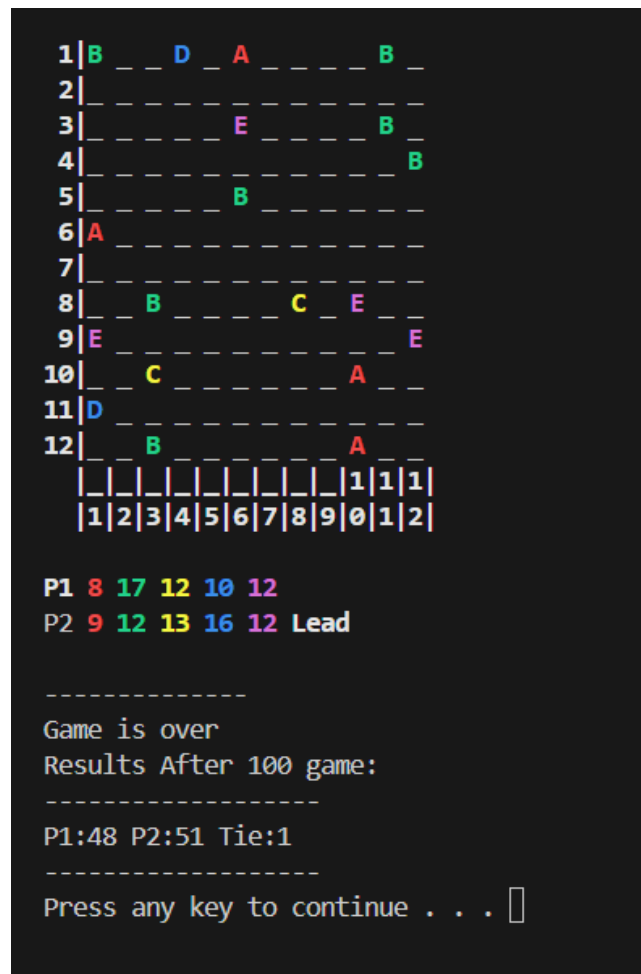


Figure 7: PC vs PC

4.3 Pc Scoring System

The computer uses a different scoring system than the normal game when evaluating itself while taking the pieces, because in the game, the one who takes the most sets wins, not the one who takes the most pieces. The image you see below shows two different situations, although there are more pieces on the left, the situation on the right is more optimal for the computer because it is easier to complete the set later in the game.

So how does it calculate these scores? We can summarize the situation as follows:

- 5 Different stones: 40 points
- 4 Different stones: 19 points
- 3 Different stones: 9 points
- 2 Different stones: 4 points



Figure 8: Pc Evaluation

1 Different stones: 1 points

Let's explain why the computer calculates the scores this way with an example. For example, in the **first case, there is a move to get 3 red pieces**, and in the **second case, there is a move to get 1 red and 1 green piece**. The second move is chosen among these situations because it will be easier to complete the set later in the game. Therefore, the computer will prefer the second situation

5 Conclusion

I think the project was a good introduction to artificial intelligence. Although it took a lot of time to implement the game and bot, and to write the code properly and efficiently, I learned many new things. It was an enjoyable project. If you want to take a look at the project code, the GitHub [link](#)

References

Algorithms explained – minimax and alpha-beta pruning. (n.d.). [\[Video\]](#). YouTube.
Minimax Algorithm for Tic Tac Toe (Coding Challenge 154). (n.d.). [\[Video\]](#). YouTube.
Simple Explanation of the Minimax Algorithm with Tic-Tac-Toe. (n.d.). [\[Video\]](#). YouTube.
Alpha-Beta Pruning. (2024, June 3). In Wikipedia. <https://en.wikipedia.org/wiki/Alpha>