

CSC 242 Artificial Intelligence

Tic Tac Toe project

Write-up

Zhizhou Zhang

University of Rochester

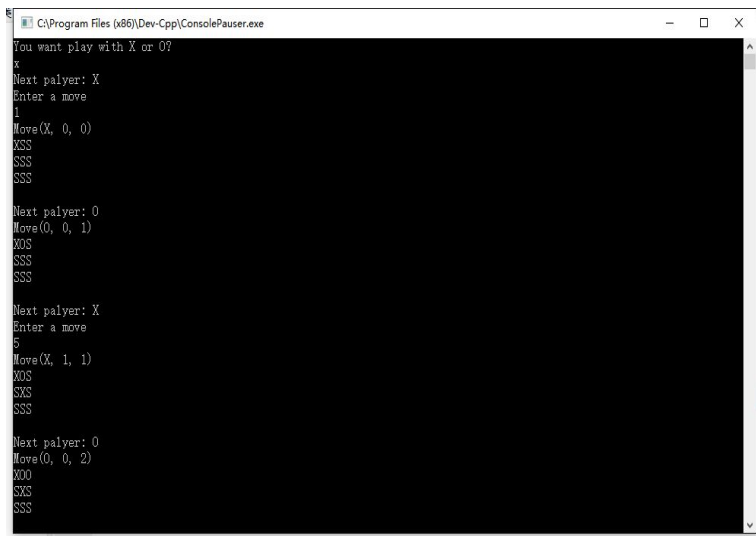
1. Overview

There is two part in this project, first part is to implement a simple Tic-Tac-Toe, and the second-part is to build a 9-board TTT with special constraint that the player have to follow the last position number to place the piece. E.g. we use two number to represent a move (a, b), a as the number of board, b like the number inside of a board. If player1 put (1, 2), the player2 then have to choose some available place from board 2.

2. Part 1, simple TTT

Actually there is not a lot to say in this part. Just follow the suggestions Professor Ferguson gave during the lecture, divide the game into different class, like player(using enumerating type that simplify the evaluation process), move, board, state and game. Also since he said we don't need to focus on this part a lot(his won words, theoretically, you can do part 2 and skip part 1), I skipped several part, like check the move is legal or not. Also, I have to say I use no algorithm in computer decision part to get the next move. It just return the next empty place, so pretty easy to beat. But still, I covered the basic requirement. I will use some

screen-shots to show.

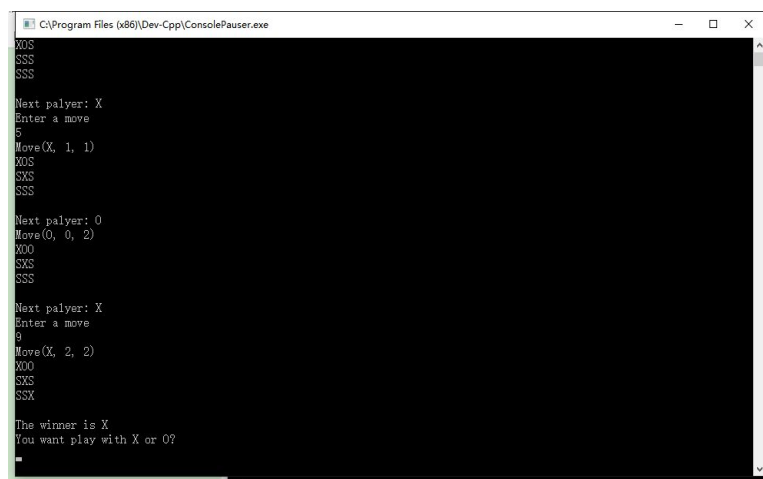


```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
You want play with X or O?
x
Next palyer: X
Enter a move
1
Move(X, 0, 0)
XSS
SSS
SSS

Next palyer: O
Move(O, 0, 1)
XOS
SSS
SSS

Next palyer: X
Enter a move
5
Move(X, 1, 1)
XOS
SXS
SSS

Next palyer: O
Move(O, 0, 2)
XOO
SXS
SSS
```



```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
XOS
SSS
SSS

Next palyer: X
Enter a move
5
Move(X, 1, 1)
XOS
SXS
SSS

Next palyer: O
Move(O, 0, 2)
XOO
SXS
SSS

Next palyer: X
Enter a move
2
Move(X, 2, 2)
XOO
SXS
SSX

The winner is X
You want play with X or O?

```

3. Part 2, advanced TTT

In this part, I use most part of the program directly from part1, however, I cover all the requirements that I did not finish in the part one. For example, the program check the if the input is valid(all invalid type take into consideration and give different feedback).

Ant to the core part, the decision-making, I summarize it as: pre-calculation, minimax and alphabet-pruning.

Pre-calculation:

First, I use some extra memory to store some triple-pair like (1, 2, 3). By

enumerating all 72 combination(3 in a row, 8 in each board, 9 board in total), I can check if there is a winner in the game by traversal. Also, I use triple-pair to calculate the utility function. Just consider one board:

1 2 3

4 5 6

7 8 9

The potential winning combination is different. Like if I put 1, I can win by (1, 2, 3), (1, 4, 7) and (1, 5, 9). But 5 have 4 possible combination. By enumerating that(actually first 8 from former part), it helps the utility and minimax procedure.

Utility:

First, I declare a parameter named standFor, which is to differ the player-first or computer first(since player's piece is different however the value of the piece remains the same). The utility function I just sum up the possible triple-number(X as -1 and O as 1, 0 leaves to blank), then switch the sum of that three slot with different value.

A) $\text{sum} = 3$ or (-3)

It's like XXX(or OOO, both in either a row, column or diagonally, and the following condition is the same), we know there is already a winner under this situation, so it is deterministic or dead game, so assign a maximum under this situation, the positive or negative depends on the standFor.

B) $\text{sum} = 2$ (or -2)

It's like XXS(S for blank), it is a threaten move, cause we know if next time the same player can play this board again, it will cause a win(assume each player choose the best move). So give this situation a large weight(but less than maximum, since it is not a win).

C) $\text{sum} = 1$ (or -1)

It can happen with two conditions: XOX and XSS.

For XOX, we know, since TTT is a zero-sum game, so block a potential winning is pretty smart move, if you can't get a win immediately. So still I give it a reasonable high score. For XSS, it's just a move, not very good or very bad. So all we need to do is determine it is former or latter, and we can finish it by bit-manipulation(former only happen when the 3 slot all are occupied)

D) $\text{sum} = 0$

It's like XOS or similar thing. It's just a normal move, so it deserve some points.

Also if the current move is in the center, we give it higher score than corner, since it introduces more combination to win, and corner generates more than edges(4, 3 and 2 respectively).

To sum up this part:

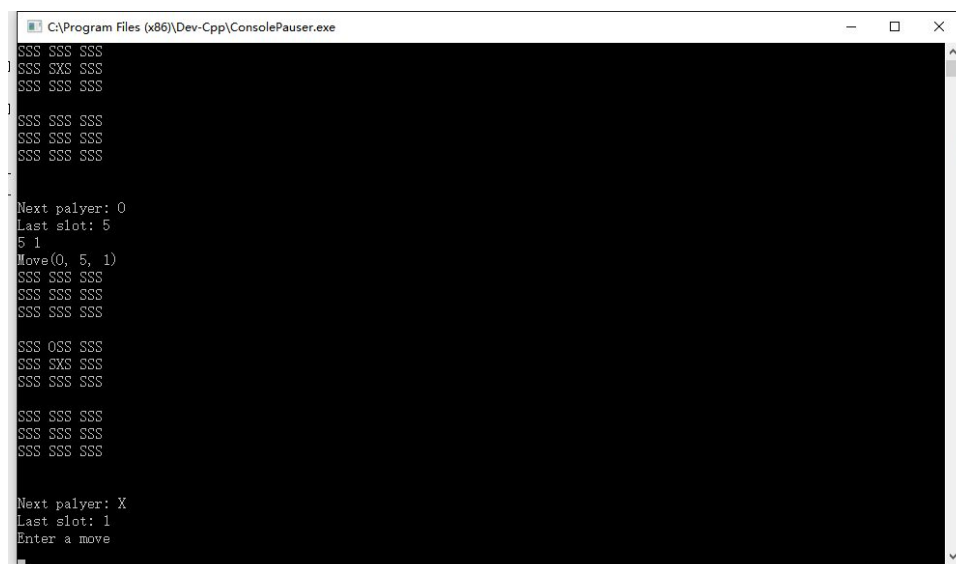
3-in a row(win) > 2-in a row(easy to win) > block the other(prevent opponent win) >>> other conditions.

Minimax and alphabeta-pruning

Since it's a game, we have to show respect to our enemy and regard each player will choose the best move, that's why I use minimax algorithm. At the same time, if I use the alpha-beta pruning introduced in the AIMA 5.3, I can cut down several bad branches. Thus it's really a huge improvement in the efficiency. And generally speaking, the goal of the computer is to maximize its potential score, as we use $-1(X)$ which is a negative number and its move O is 1. If we go second, I just use the parameter `standFor` to change it from 1 into -1, thus, we can still use the same utility and alpha-beta pruning with just slight modification.

To sum up this part, using minimax algorithm with the help of alpha-beta pruning to search from all possible next moves. Then choose the move with highest score.

Screen-shots and performance



```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
SSS SSS SSS
SSS SXS SSS
SSS SSS SSS

SSS SSS SSS
SSS SSS SSS
SSS SSS SSS

Next palyer: O
Last slot: 5
5 1
Move(0, 5, 1)
SSS SSS SSS
SSS SSS SSS
SSS SSS SSS

SSS OSS SSS
SSS SXS SSS
SSS SSS SSS

SSS SSS SSS
SSS SSS SSS
SSS SSS SSS

Next palyer: X
Last slot: 1
Enter a move
```

Sadly, the program works pretty well. As a human and the guy who coded this, I can't beta it, if I set the depth of search to something large than 2! Set to 7, I spent whole afternoon to get a win,

but I failed..... Based on my desktop, depth 7 is best, since 8 will cause very obvious delay time.

So personally I'm happy with it, since I can't beat it(unless I cheat like change depth into 0)

Some bugs and other defects

Well, in the advance TTT part, I broke several rules in OOP(object-oriented-programming) like encapsulation. Sometimes I have to change the private or protected members into public, to access to it and modify it. If I have extra time later, I will try to re-write in good encapsulation style with a GUI.

Reference:

Artificial Intelligence: A modern approach(especially chapter 5)

Slides from Professor Ferguson

<https://en.wikipedia.org/wiki/Tic-tac-toe>