Chen Zhang
CSC 242 Artificial Intelligence
Project 1
Due Date: 02/15/16
No partners

Regular Tic Tac Toe and Advanced Tic Tac Toe Games

Introduction

The purpose of this project was to implement an artificial intelligence algorithm to the well-known Tic Tac Toe (TTT) game and the advanced version of the game. The rules of advanced TTT was explained in the project1.pdf, and this version has much more complex state spaces than the regular one. The A.I should be able to play against human player with a good strategy. The good strategy in regular TTT means it should NEVER lose to human player no matter it plays as "X" or "O"; and it should win over human player at most of times in advanced TTT. The actual implementations of those two games in my project are different, details were explained in the later of this paper.

Basic Concepts

Both two versions of TTT has competitive environment, and it is turn-taking, zero-sum games of perfect information. The environments are deterministic and fully observable. To form the regular TTT state space problem, we need to define several concepts in the following:

Initial State: the empty board of the game, total position is 9;

Action: placing X or O in the board;

Applicability: whether the previous position is occupied;

Result: after placing the X or O in the board, the state of the board;

Goal Test: either the board is full or one player achieves three in a column, row or diagonal of pieces;

Path Cost: each action costs 1 in this case (we do not care about path cost in this implementation)

It is clear that the number of regular TTT states is 9!= 362,880, it is possible for the A.I to do a depth first search based on MiniMax algorithm. A quicker improved algorithm called Alpha Beta pruning was actually used in my regular TTT project. The benefit of pruning is some branches of the search tree can be eliminated so that the search time can be limited to the more promising tree. The reason why I preferred Alpha Beta pruning was I intended to write a general method for regular TTT and advanced TTT. (Laziness is the best motivation to improve performance of computers.)

The advanced TTT has a total state $\sim 10^{120}$, which has more possibilities than the total number of atoms in the universe $\sim 10^{80}$. In this case, heuristic function was implemented to cut off the search at a certain depth. The state space problem was formed as following:

Initial State: the empty board of the game, total position is 81;

Action: placing X or O in the board;

Applicability: whether the previous position is occupied; and the current board number should be the previous position number; if the current board number is full, any board number is fine

Result: after placing the X or O in the board, the state of the board;

Goal Test: either the big board is full or one player achieves three in a column, row or diagonal of pieces in a single small board;

Path Cost: each action costs 1 in this case (we do not care about path cost in this implementation)

Algorithms for TTT

Player enum class has Player.X and Player.O, and a method nextPlayer returns the opposite player of the current player. I used a 3*3 Player array to represent the actual board in the Board class. Inside the Board class, getWinner() method checks a winner case in horizontal/vertical/diagonal/anti-diagonal scenarios, and isFull() check if the board is entirely occupied. Those two methods together is a representation of goal test. One of the most significant methods in this class called copyBoard(), which might need some clarification here. Recall that, in the Alpha-Beta pruning, it generates successors of possible states after each move; however, those states are pseudo-states, the actual/original state must be saved. So copyBoard() is used to duplicate the current Board. Another important method called getPossibleMoves(), this method is return all unoccupied positions from the board. The state class basically has similar methods as the board; additionally, it also keeps track of whose turn for the current state. The Move class takes parameters: Player(X or O) and a position number. Board, State, Move, Player classes are basic game designs; so far no actual A.I is involved.

In the Games class, getHumanMove() method is to take a position number(integer) from the user. Since my board is represented in 3*3 2D array, the input number (1D) has to be converted into 2D array representations. The equation can be found in Figure 1.

$$x = \frac{input\ number - 1}{3}$$

$$y = (input\ number - 1)\%3$$

Figure 1:

In Java, Integer division returns the true result of division rounded down to the nearest integer. For example: 2/3=0 in java instead of 1.

This 1D to 2D projection is straightforward for the 3*3 board, but it becomes more complex when dealing with 9*9 board in advanced version of TTT. In my pruning() method, it takes four parameters: state, integer(alpha), integer(beta), and integer(depth). Like I mentioned before, this pruning()method is generic method that can be applied both in TTT and advanced TTT projects. The recursive calls stop after a terminal state is reached on the duplicate board (that's why copyBoard() matters); the computer assigns a utility of +1 to all computer wins, -1 to all human wins, and 0 to all draws. At this point, the A.I unwinds the recursion and backtracks through the call stack, and the best move for computer is chosen. During each turns, the move that benefits the current agent the most is chosen. The best move is chosen by the utility value. The Player.X is assumed to play for the lowest utility, and Player.O for the highest. Then, getAIMove() is the corresponding move from A.I, where initializes Alpha-Beta Pruning.

Algorithms for Advanced TTT

I initially thought the advanced TTT is basically changing the 3*3 board to 9*9 board, plus minor revision of detecting the winner, and adding heuristic function. Apparently, things were not that smooth as expected. I got stuck for 2 days, so I changed my original implementation from 2D array to 1D array. Before going to the details about my new implementation, I would like to list the challenges I met for the 9*9 2D array implementation:

- Projection from board number and position number to 9*9 is very complicated, and confusing: those equations basically took me more than 3 hrs to figure it out by myself

$$a = \frac{(positionNumber - 1)}{3}$$

$$b = (positionNumber - 1)\%3$$

$$c = \frac{(boardNumber - 1)}{3}$$

$$d = (boardNumber - 1)\%3$$

$$x = a + c \times 3$$

$$y = b + d * 3$$

*Figure 2: Detailed equation about how to project user input into 9*9 board 2D Array positionNumber and boardNumber are acquired from user; and x/y are the coordinates in the 2D array*

- Since the winner case has to be detected in 3*3 small boards, I have to divide the 9*9 board into 9 3*3 boards mathematically (I have not figured this out before I gave up)
- Alpha-beta pruning has to be applied to 9*9 board state instead of 3*3 board state; otherwise the A.I assumes it can apply its move to anywhere into an empty 3*3 board
- The most challenging part is: debug; once something is not working, I have to be extremely clear about math projection and keep track of previously board number and position number

Due to all the challenges above, I decided to start from scratch again. This time 1D array is created for Board class. The board class is initialized for size at 10, leaving 0 as null. In other

words, if the user inputs 5, the board[5] will be occupied. A BigBoard class is created as 1D array as well, this BigBoard is essentially the game board, which includes 9 small board 1D array. To be clear, if a user wants to apply move to board number 5 and position number 9, it simply calls BigBoard[5].board[9]. This simplifies all the projection/mathematical problems. The checkWinner() method is inherited from the regular TTT, but it iterates all the 9 boards (from BigBoard[1] to BigBoard[9]).

The State, Player classes remain similar to the previous implementation. A BigState class was created for the purpose that it can keep track of current board number and position number.

In Games class, several vital designs should be addressed here:

1. In the gameLoop() method, the boardNumber is set to the positionNumber after the current move is applied to the BigState. So the user does not necessarily to input the new boardNumber unless the previous the new board is full. This avoids user's invalid input boardNumber.
2. The pruning method is almost same as TTT one even the implementation was changed from 2D array into 1D array. The only change here is once the searching depth reaches 10, the recursion terminates and starts to unwind. Recall from regular TTT, the recursion will not stop until it reaches the end of depth (9). The concept here is I do not intend to let the users wait too long for A.I to calculate the best move. The time usually increases exponentially if the search depth level raises.
3. Last but not least: heuristic function. The heuristic function is called once the recursion reaches terminal, which is similar to the regular TTT. But the value returned by heuristic function is not perfect and is only estimated. Basically, my heuristic function checks whether there is two consecutive same players in a

row/column/diagonal/anti-diagnol in the small board and of course this iteration goes through 9 small boards. For example: if one small board has two Player.X and an empty position in a row/column, the score will increase by 300 points. If the empty position is replaced by the opposite player, the score will only increase by 100 points. Vice Vera.

Result:

For TTT, my A.I never lose to human player. It can easily beat the player if the human player does not play accordingly or smart. In theory, that's because alpha beta pruning can search down the entire tree (totally 9 depth).

For advanced TTT, I tested 10 times, the A.I outplays 10 times against me. Since my heuristic function is basically checking if there is some consecutive player in a row/col/diagonal, so it still might give me some bad heuristic function in some rare cases. But it's hard to test it out those cases without tons of times.

In Figure 3, one can easily find out the elapsed time for finding best move from A.I increases

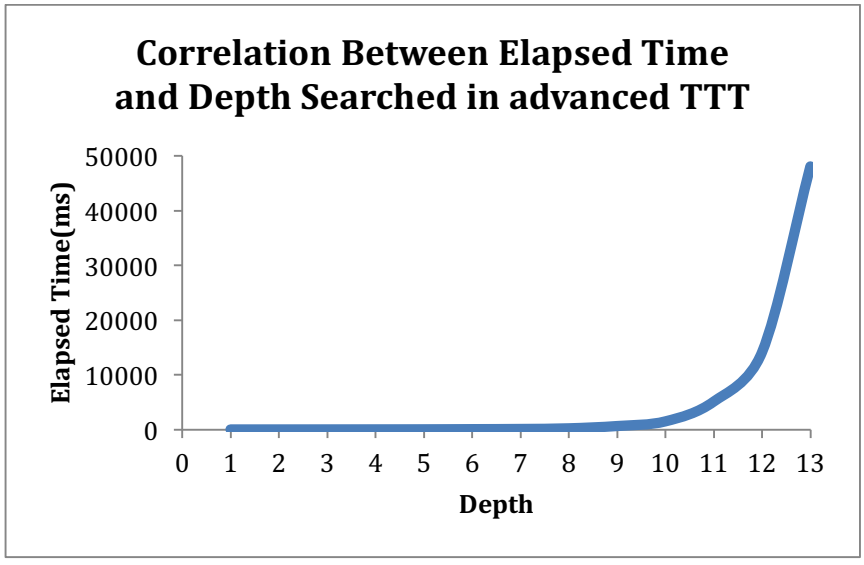


Figure 3

exponentially if the depth level raises. Both the time and space complexity here is O(n!).

Conclusion and Future Possible Improvement:

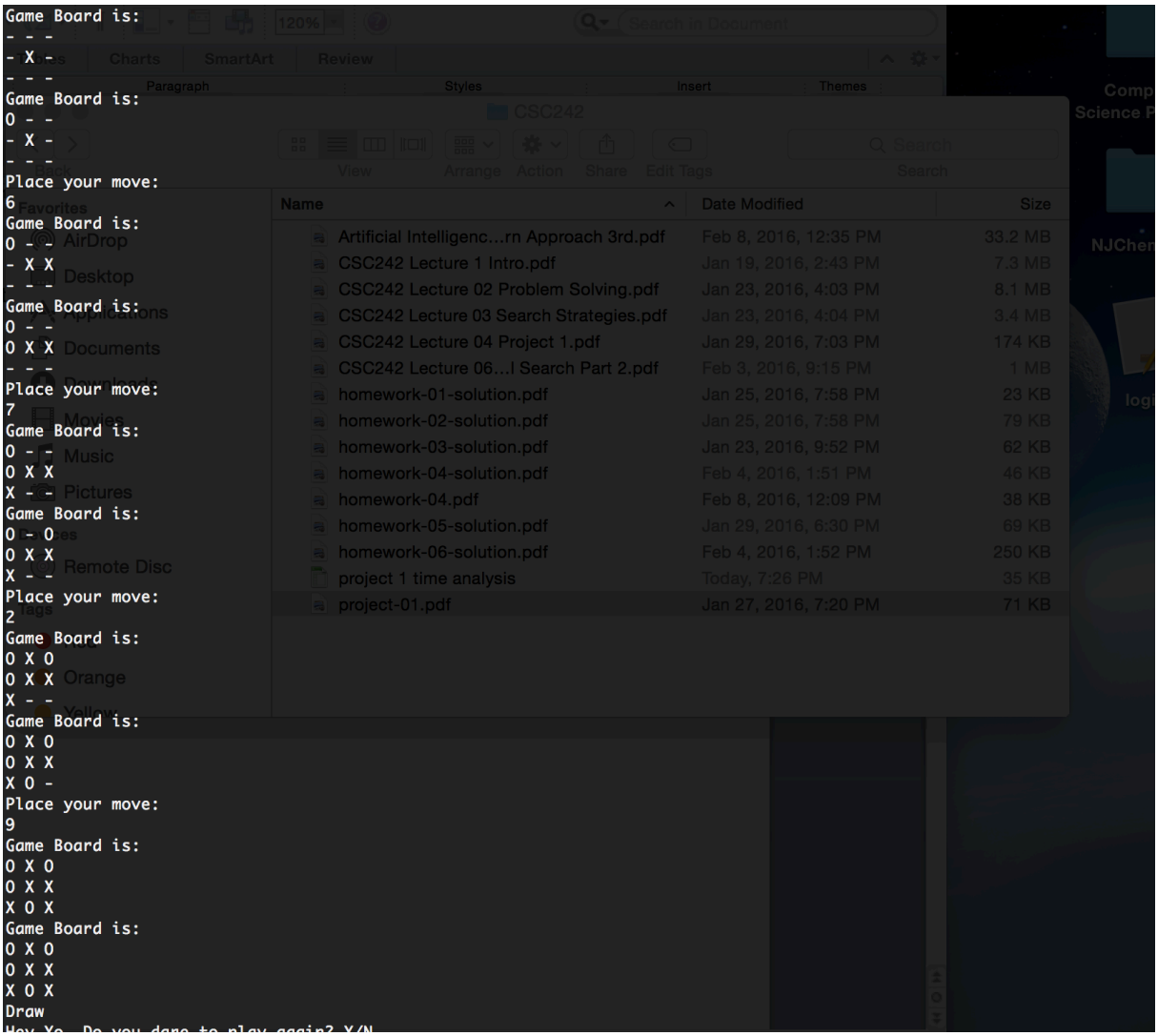
The exponential complexity of the advanced TTT problem created a number of computational problems when it tries to reach the end of recursion. In order to have better winning rate, the tree depth should be as large as possible. But in reality, it cannot be done; so we have to trade performance for time.

Another problem is heuristic function; it's nearly impossible to write a perfect function to evaluate scores for the current states. Although the A.I was able beat human player in all my tests, there is a need for more rigorous testing of the program versus human player.

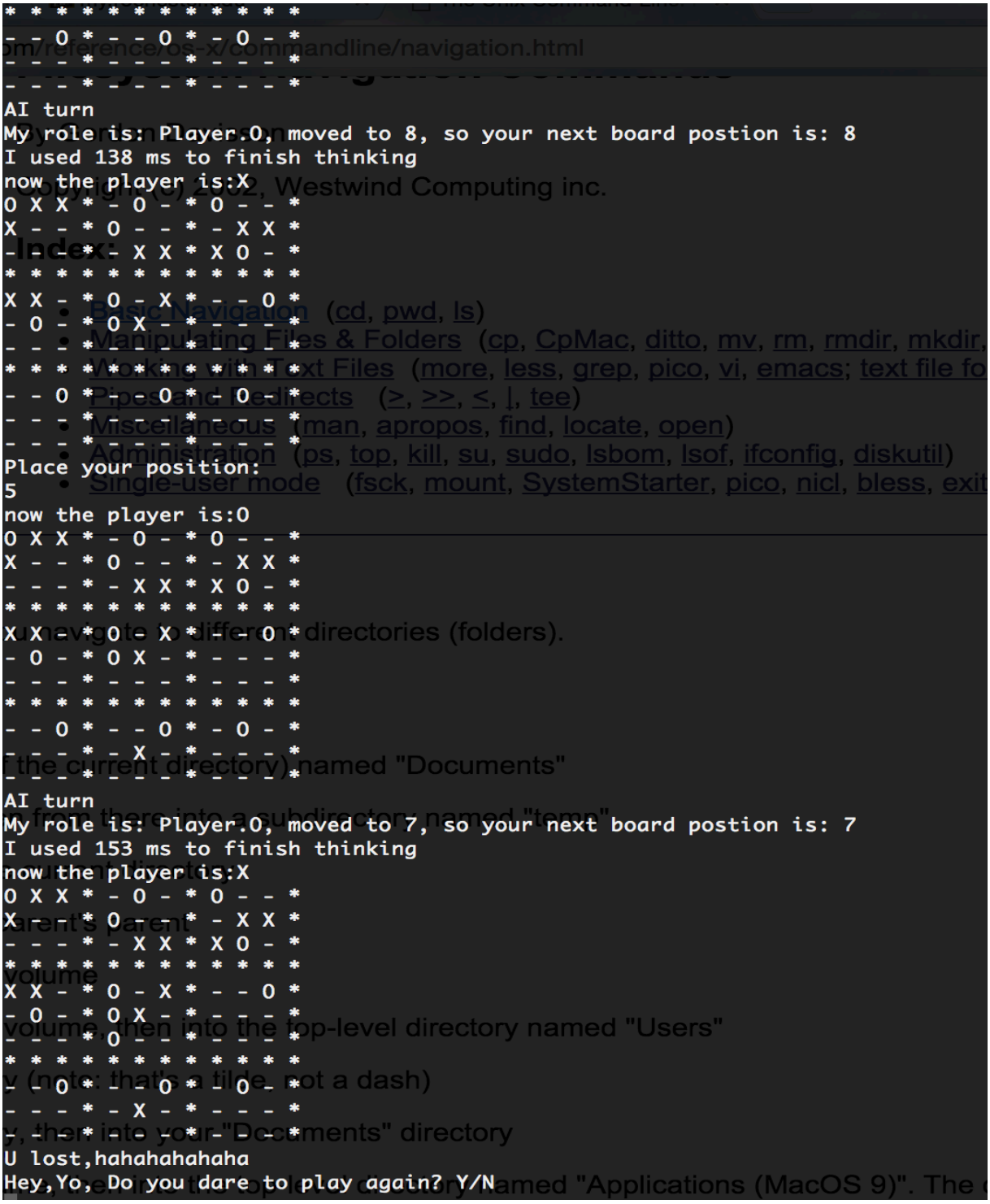
During my advanced TTT tests, the A.I sometimes takes additional move/moves to win the game. I think the problem is because the difference between a solution and optimal solution, where minimizes the path cost. In alpha-beta pruning and heuristic function, I did not consider the path cost. In future, an optimal solution algorithm might be developed for better performance.

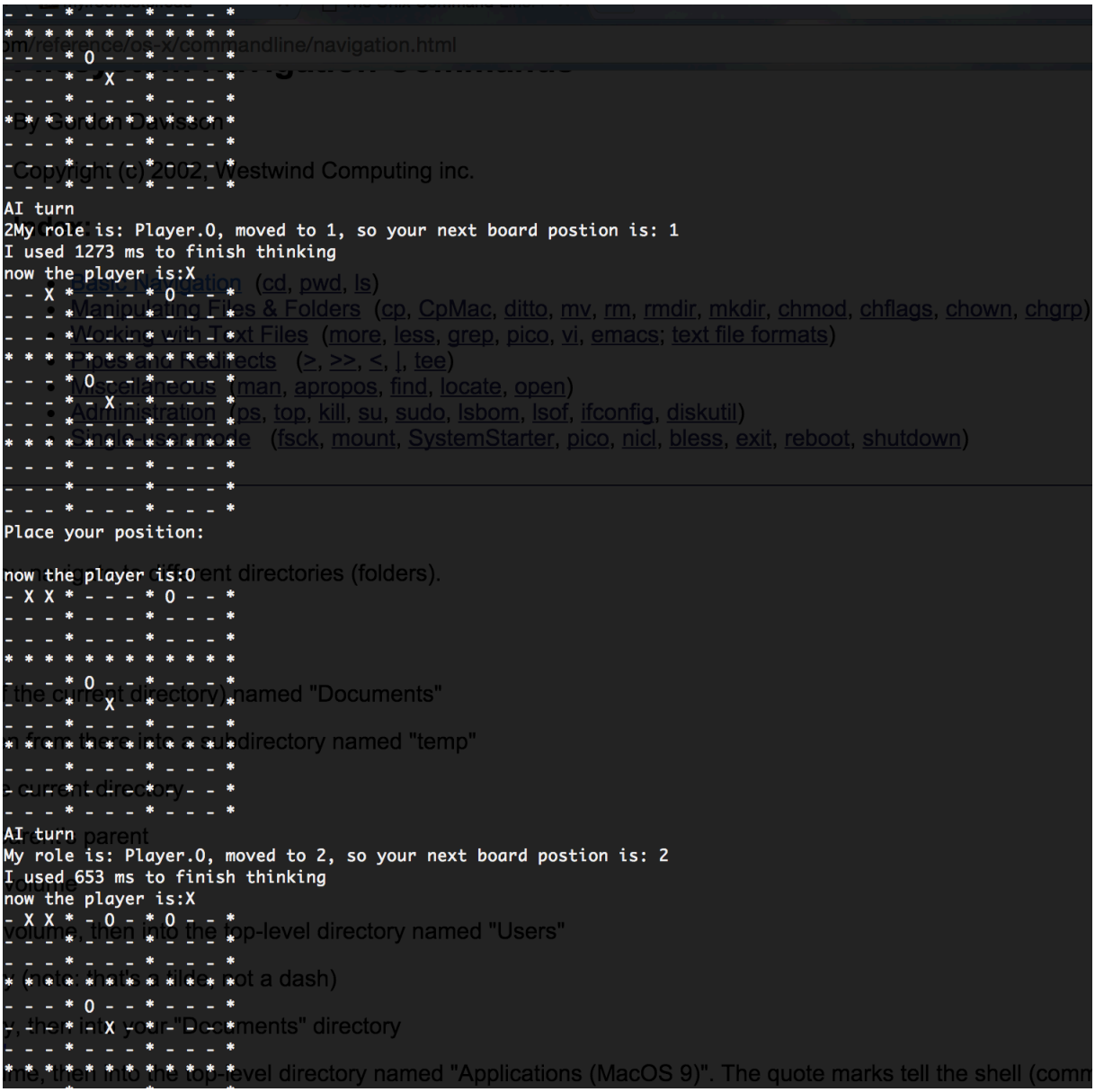
Attachments:

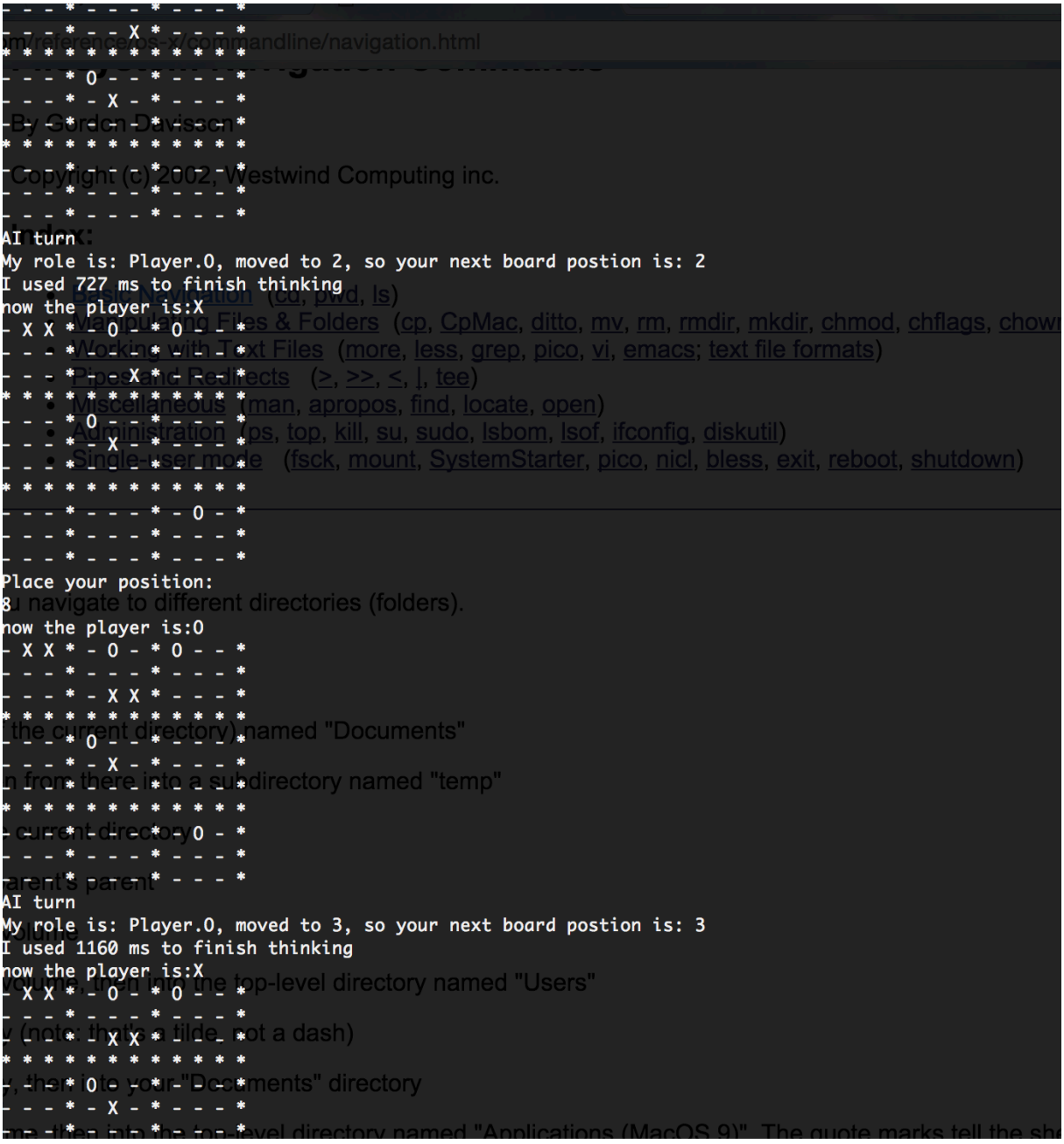
Example of TTT procedure:

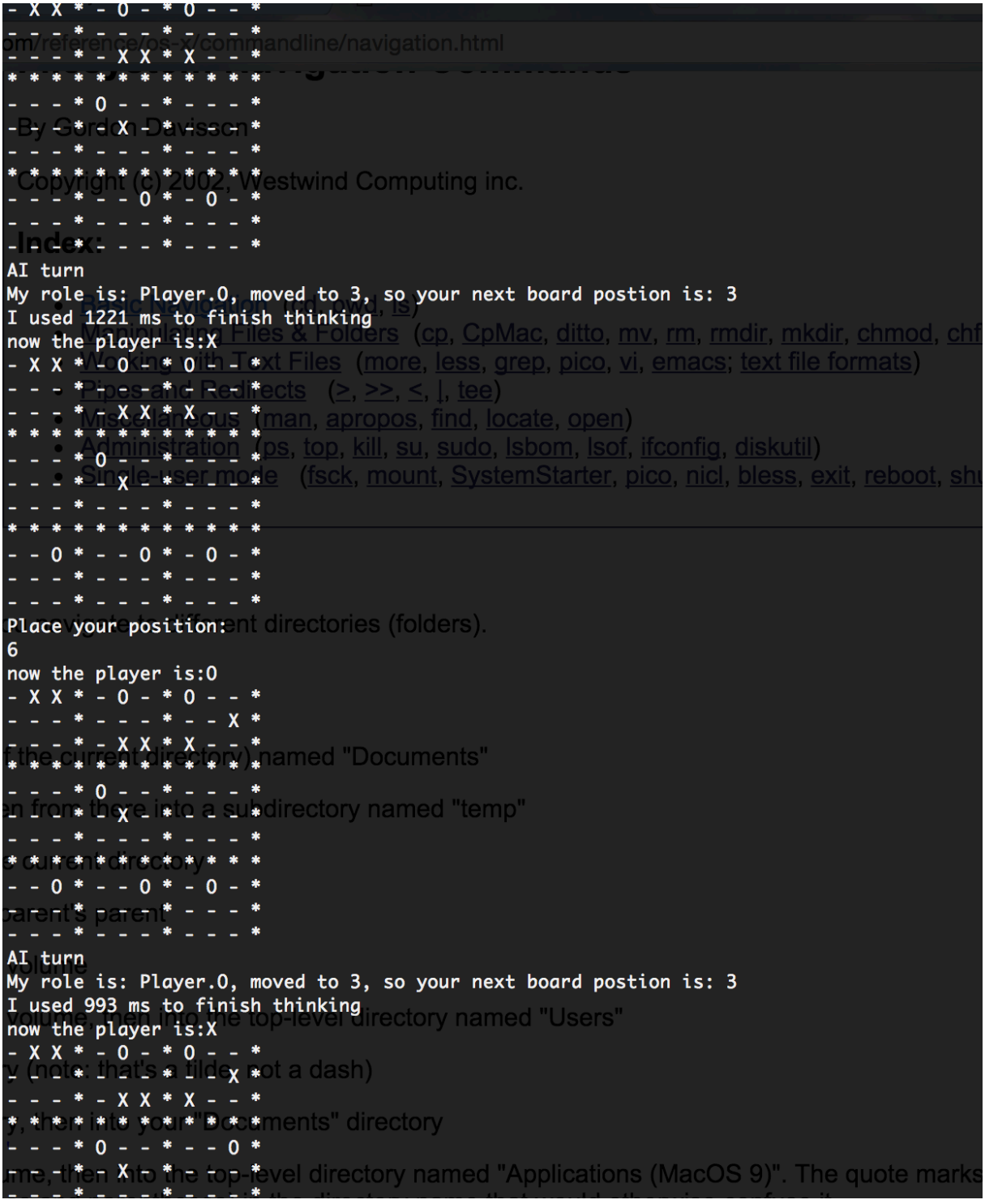


Example of 9X9 Advanced TTT:

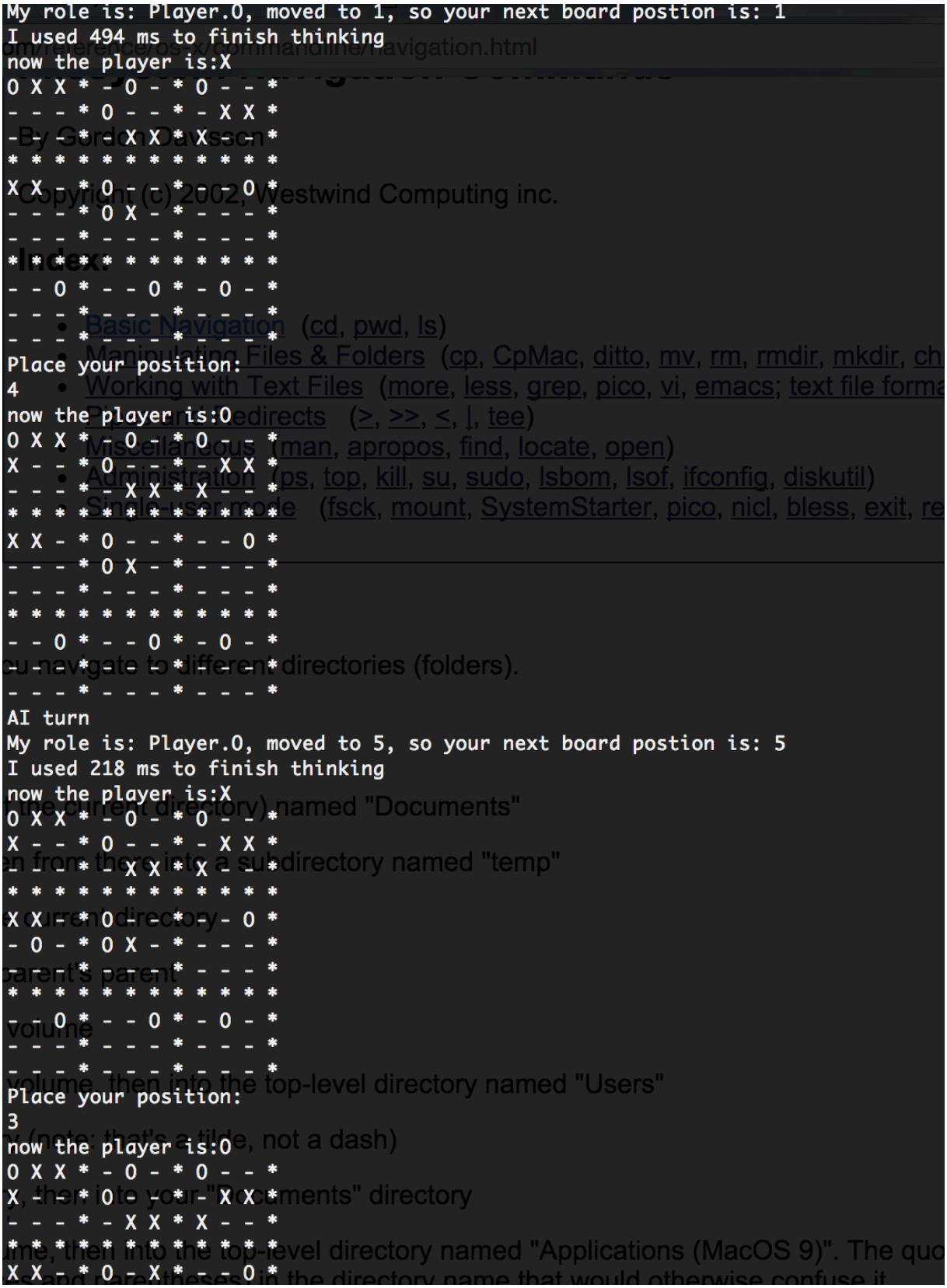


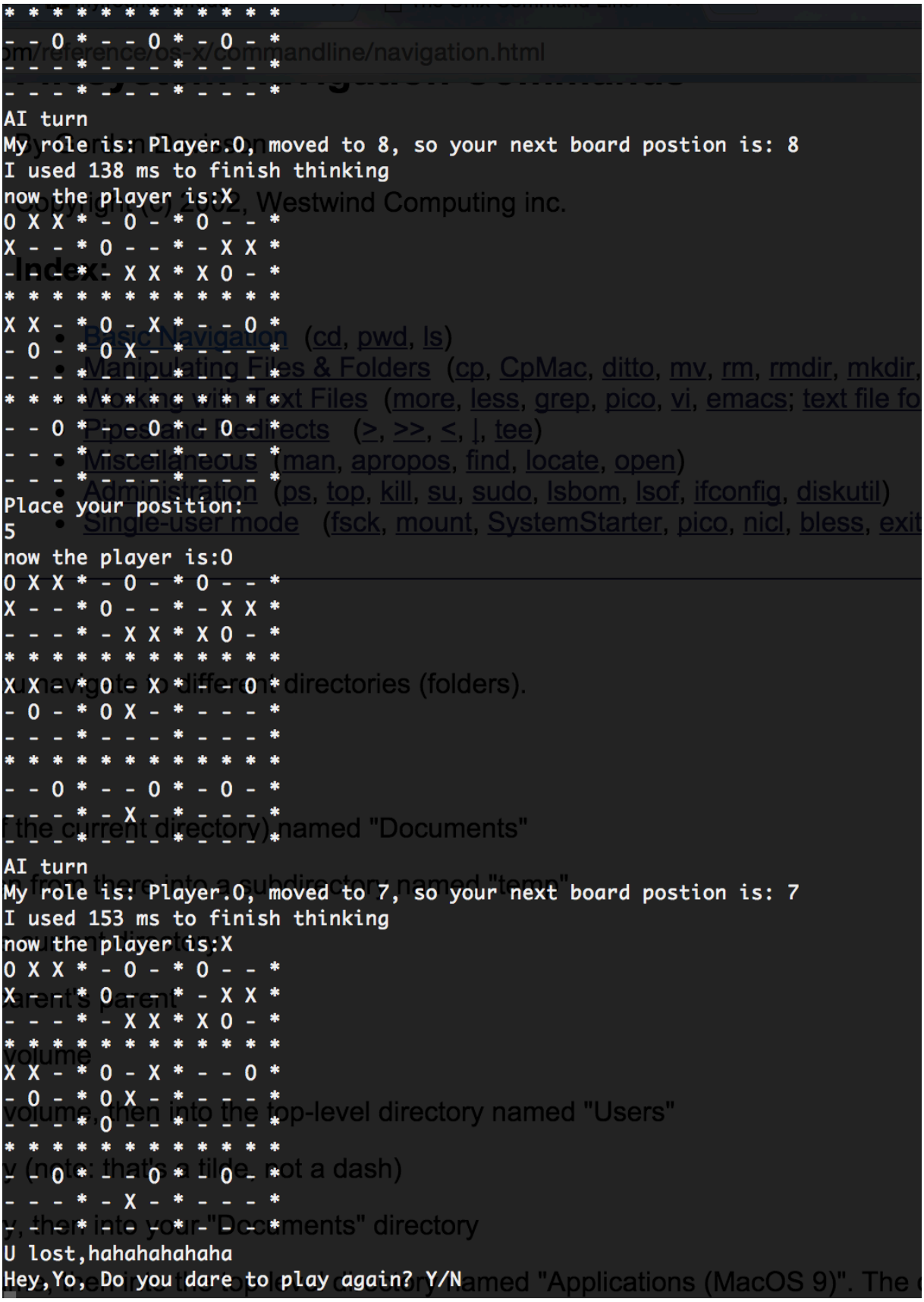

- - - * - - - * - - - *
* * * * * * * * * * * *
- - - * 0 - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
By Gordon Davisson
- - - * - - - * - - - *
- - - * - - - * - - - *
Copyright (c) 2002, Westwind Computing inc.
- - - * - - - * - - - *
AI turn
2My role is: Player.O, moved to 1, so your next board postion is: 1
I used 1273 ms to finish thinking
now the player is:X
- - X * - - - * 0 - - *
- - - * - - - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - - * 0 - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - - * - - - * - - - *
- - - * - - - * - - - *
- - - * - - - * - - - *
Place your position:
now the player is:O
- X X * - - - * 0 - - *
- - - * - - - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - - * 0 - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - - * - - - * - - - *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.O, moved to 2, so your next board postion is: 2
I used 653 ms to finish thinking
now the player is:X
- X X * - O - * 0 - - *
- - - * - - - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - - * 0 - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *


- - - * - - - * - - - *
- - - * - - X * - - - *
* * * * * * * * * * *
- - - * O - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.O, moved to 2, so your next board postion is: 2
I used 727 ms to finish thinking
now the player is:X
- X X * - O - * O - - *
- - - * - - - * - - - *
- - - * - - X * - - - *
* * * * * * * * * * *
- - - * O - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * *
- - - * - - - * - O - *
- - - * - - - * - - - *
- - - * - - - * - - - *
Place your position:
8
now the player is:O
- X X * - O - * O - - *
- - - * - - - * - - - *
- - - * - X X * - - - *
* * * * * * * * * * *
- - - * O - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * *
- - - * - - - * O - *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.O, moved to 3, so your next board postion is: 3
I used 1160 ms to finish thinking
now the player is:X
- X X * - O - * O - - *
- - - * - - - * - - - *
- - - * - X X * - - - *
* * * * * * * * * * *
- - - * O - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * *
- - - * - - - * O - *
- - - * - - - * - - - *
- - - * - - - * - - - *


- X X * - 0 - * 0 - - *
- - - * - - - * - - - *
- - - * - X X * X - - *
* * * * * * * * * * * *
- - - * 0 - - * - - - *
- - - * - X - * - - - *
* * * * * * * * * * * *
- - - * - - 0 * - 0 - *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.0, moved to 3, so your next board postion is: 3
I used 1221 ms to finish thinking
now the player is:X
- X X * - 0 - * 0 - - *
- - - * - - - * - - - *
- - - * - X X * X - - *
* * * * * * * * * * * *
- - - * 0 - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - - - * - - - *
- - - * - - - * - - - *
Place your position:
6
now the player is:0
- X X * - 0 - * 0 - - *
- - - * - - - * - - X *
- - - * - X X * X - - *
* * * * * * * * * * * *
- - - * 0 - - * - - - *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.0, moved to 3, so your next board postion is: 3
I used 993 ms to finish thinking
now the player is:X
- X X * - 0 - * 0 - - *
- - - * - - - * - - X *
- - - * - X X * X - - *
* * * * * * * * * * * *
- - - * 0 - - * - - 0 *
- - - * - X - * - - - *
- - - * - - - * - - - *

```
- X X * - 0 - * 0 - - *
- - - * - - - * - X X *
- - - * - X X * X - - *
* * * * * * * * * * *
- - - * 0 - - * - - 0 *
- - - * - X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.O, moved to 4, so your next board postion is: 4
I used 958 ms to finish thinking
now the player is:X
- X X * - 0 - * 0 - - *
- - - * - - - * - X X *
- - - * - X X * X - - *
* * * * * * * * * * *
- - - * 0 - - * - - 0 *
- - - * 0 X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - - - * - - - *
- - - * - - - * - - - *
Place your position:
2
now the player is:O
- X X * - 0 - * 0 - - *
- - - * - - - * - X X *
- - - * - X X * X - - *
* * * * * * * * * * *
- X - * 0 - - * - - 0 *
- - - * 0 X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.O, moved to 4, so your next board postion is: 4
I used 402 ms to finish thinking
now the player is:X
- X X * - 0 - * 0 - - *
- - - * 0 - - * - X X *
- - - * - X X * X - - *
* * * * * * * * * * *
- X - * 0 - - * - - 0 *
- - - * 0 X - * - - - *
- - - * - - - * - - - *
```


My role is: Player.O, moved to 1, so your next board postion is: 1
I used 494 ms to finish thinking
now the player is:X
0 X X * - O - * O - - *
- - - * O - - * - X X *
- - - * - X X * X - - *
* * * * * * * * * * * *
X X - * O - - * - - O *
- - - * O X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - O * - - O * - O - *
- - - * - - - * - - - *
- - - * - - - * - - - *
Place your position:
4
now the player is:O
0 X X * - O - * O - - *
X - - * O - - * - X X *
- - - * - X X * X - - *
* * * * * * * * * * * *
X X - * O - - * - - O *
- - - * O X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - O * - - O * - O - *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.O, moved to 5, so your next board postion is: 5
I used 218 ms to finish thinking
now the player is:X
0 X X * - O - * O - - *
X - - * O - - * - X X *
- - - * - X X * X - - *
* * * * * * * * * * * *
X X - * O - - * - - O *
- O - * O X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - O * - - O * - O - *
- - - * - - - * - - - *
- - - * - - - * - - - *
Place your position:
3
now the player is:O
0 X X * - O - * O - - *
X - - * O - - * - X X *
- - - * - X X * X - - *
* * * * * * * * * * * *
X X - * O - X * - - O *


* * * * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - - - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.0, moved to 8, so your next board postion is: 8
I used 138 ms to finish thinking
now the player is:X
0 X X * - 0 - * 0 - - *
X - - * 0 - - * - X X *
- - - * - X X * X 0 - *
* * * * * * * * * * * *
X X - * 0 - X * - - 0 *
- 0 - * 0 X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - - - * - - - *
- - - * - - - * - - - *
Place your position:
5
now the player is:0
0 X X * - 0 - * 0 - - *
X - - * 0 - - * - X X *
- - - * - X X * X 0 - *
* * * * * * * * * * * *
X X - * 0 - X * - - 0 *
- 0 - * 0 X - * - - - *
- - - * - - - * - - - *
* * * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - X - * - - - *
- - - * - - - * - - - *
AI turn
My role is: Player.0, moved to 7, so your next board postion is: 7
I used 153 ms to finish thinking
now the player is:X
0 X X * - 0 - * 0 - - *
X - - * 0 - - * - X X *
- - - * - X X * X 0 - *
* * * * * * * * * * * *
X X - * 0 - X * - - 0 *
- 0 - * 0 X - * - - - *
- - - * 0 - - * - - - *
* * * * * * * * * * * *
- - 0 * - - 0 * - 0 - *
- - - * - X - * - - - *
- - - * - - - * - - - *
U lost,hahahahahaha
Hey,Yo, Do you dare to play again? Y/N