CS 640 Programming Assignment 3 Report

AI Game —— 4×4×4 Tic-Tac-Toe

November 25, 2019

Teamwork

Team members	Kerberos	BU ID	Works
Ziqi Tan	ziqi1756	U 88387934	Design data structure and algorithm. Coding.
Kaijia You	caydenyo	U 44518396	Coding and software testing. Write documentations.
Tian Ding	dingtian	U 90706530	Coding. Adjusting parameters.

Contents

1 Assignment Requirements 3

2 Insight into 4×4×4 Tic-Tac-Toe Game 3

3 Skeleton Code Quick Review 3

4 Methodology and Implementation 4

4.1 Evaluation Function 4

4.2 Change Data Structure 4

4.3 Occupy the strongest points as fast as possible 5

4.4 Winning move and force move 5

4.5 Minimax 5

4.6 Alpha-Beta Pruning 6

4.7 Progressive Deepening 6

4.8 Heuristic Pruning 6

5 Algorithm Design 7

6 Results and Discussion 9

6.1 Battle with randomness AI 9

6.2 AI vs. AI with the same settings 9

6.3 AI vs. AI with different settings 10

6.4 AI behavior analysis 10

7 Conclusions 10

References 10

Appendix 11

The rest of this report is organized as follows. First, we review the assignment requirements. Second, we provide insight into the 4×4×4 Tic-Tac-Toe Game mostly based on [1]. Third, we go through the skeleton code and discuss our methodology. Finally, we discuss our test result. Our strategy can easily defeat the algorithm with simple defend and attack strategy and perfectly defeat random algorithm. The second player of our AI performs better than the first player.

1 Assignment Requirements

In this assignment, we are required to implement an AI 4x4x4 cubic tic-tac-toe game by using minimax and alpha-beta pruning method which drive our AI make decisions as beneficial as possible. We should try our best to modify our algorithm and beat AI implemented from other teams for extra credits.

2 Insight into 4×4×4 Tic-Tac-Toe Game

According to [1], the first player in 4×4×4 Tic-Tac-Toe Game can always force a win. Additionally, no draw exists.

3 Skeleton Code Quick Review

runTicTacToe.java serves as the game engine. The **run()** method is a critical part. It should be recognized that the decision algorithm (myAIAlgorithm(board, player)) will be called many times in the game.

```
public void run()
{

    Random rand = new Random();
    int turn = rand.nextInt(2)+1; //1 = player1's turn, 2 = player2's turn, who go first is randomized

    while((result = isEnded())==0) //game loop
    {
        if(turn==1)
        {
            positionTicTacToe player1NextMove = ai1.myAIAlgorithm(board,1); //1 stands for player 1
            if(makeMove(player1NextMove,1,board))
                turn = 2;
        }
        else if(turn==2)
        {
            positionTicTacToe player2NextMove = ai2.myAIAlgorithm(board,2); //2 stands for player 2
            if(makeMove(player2NextMove,2,board))
                turn = 1;
        }
        else
        {
            //exception occurs, stop
            System.out.println("Error!");
        }
    }
```

Data structure of the Tic Tac Toe board

```
public class positionTicTacToe {

    int x;
    int y;
    int z;
    int state;
    // mark by player 1 or player 2 or null ('X', 'O', '_')
    // { 0 : unmarked, -1 : No meaning, 1 : 'X', 2: 'O' }
```

4 Methodology and Implementation

4.1 Evaluation Function

In every board configuration, the player should evaluate his/her current situation, which is a crucial part of a heuristic process for an Artificial Intelligence. We adapt the following strategy [2].

The evaluator has a list of how many 1-in-a-rows, 2-in-a-rows, 3-in-a-rows and 4-in-a-rows each player has. Where the evaluators differ is in what they do with this information, as described below.

```
private int[] playerSequenceNum = new int[4];
private int[] opponentSequenceNum = new int[4];
```

It assigns a positive value to every n-in-a-row the player has, and a negative value to every n-in-a-row the opponent has. An n-in-a-row is worth about an order of magnitude more than an (n-1)-in-a-row. The player's rows are worth more than the opponent's rows, which means an aggressive attack strategy.

```
private static byte[][] winningLine = new byte[76][4];
private static final int[] playerSequenceValue = new int[] {1, 15, 130, 100000};
private static final int[] opponentSequenceValue = new int[] {-1, -10, -100, -100000};

    int value = 0;
    for( int i = 0; i < 4; i++ ) {
        value += playerSequenceNum[i] * playerSequenceValue[i];
        value += opponentSequenceNum[i] * opponentSequenceValue[i];
    }
```

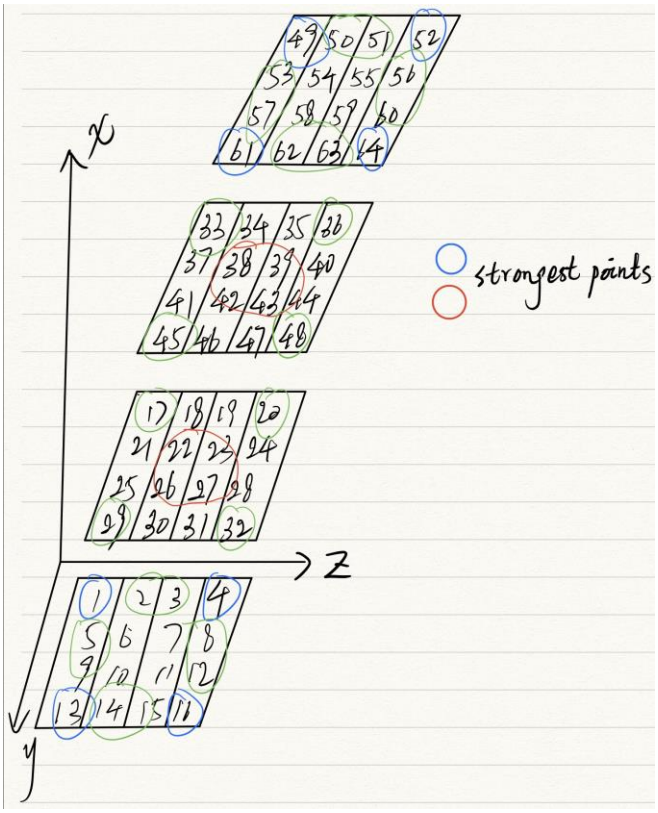
4.2 Change Data Structure

```
private byte[] curBoard = new byte[64];
```

Without deep copy of a list of objects, this board data structure takes much less time and memory.

In the skeleton code, position in the cube is represented by an object that contains x, y, z and status which occupies a lot of memory to store when the scale of the cube becomes larger. What's more, the for-loop in the makeMove method is terribly time cost and not necessarily. We have already known the exact node that we want to mark after doing minimax and alpha-beta pruning, however, the board is store in a List that means we must iterate the items in the List to get the position we want, which will cost O(n) in the worst case. The efficiency will be obviously better if the board is store in an array, we can mark the position simply by the subscript of the array and it only takes O(1). For this reason, we came up with the version 2 of this design.

4.3 Occupy the strongest points as fast as possible



The strongest point is the point that cross by most of the winning lines. The strongest points shown in the picture has 7 wining lines respectively. It is easy to understand that the 8 nodes in the center of the cube give players more winning strategies. Each player will not win in the beginning of the game, so it is important for players to occupy the strongest nodes as fast as possible when the game begins.

4.4 Winning move and force move

As the game begins, the board will be different after players finish their turns. Player will check if 3 nodes of his/her own are filled in the same line, if yes, then player can take a winning move immediately. Otherwise player should check if 3 nodes of opponent are filled in the same line, is yes, then player has no choice but have to stop opponent from winning which is called force move.

4.5 Minimax

If the current chess board is not fit in the situation above then AI should make decisions by minimax and alpha-beta pruning method to maximize the its profit and minimize the opponent's profit at the same time.

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

Pseudocode of minimax

4.6 Alpha-Beta Pruning

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cut-off *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β := min(β, value)
            if α ≥ β then
                break (* α cut-off *)
        return value
```

Pseudocode of alpha-beta pruning

4.7 Progressive Deepening

Analyze game situation to depth = 1, depth = 2, depth = 3, ··· until time is up.

4.8 Heuristic Pruning

Define the traverse order. We traverse the strongest points first, which are obviously promising move.

```
private static final int[] traverseOrder = {
    21, 22, 25, 26, 37, 38, 41, 42,
    0, 3, 12, 16, 48, 51, 60, 63,
    1, 2, 4, 5, 6, 7, 8, 9,
    10, 11, 13, 14, 15, 17, 18, 19,
    20, 23, 24, 27, 28, 29,
    30, 31, 32, 33, 34, 35, 36, 39,
    40, 43, 44, 45, 46, 47, 49,
    50, 52, 53, 54, 55, 56, 57, 58, 59,
    61, 62
};
```

5 Algorithm Design

1. Pseudocode of myAIAlgorithm

Algorithm 1 myAIAlgorithm

Input List<positionTicTacToe> *board*, int *player*
Return positionTicTacToe *myNextMove*

procedure MYAIALGORITHM(*board*, *player*)

initialization

winMove ← *getWinMove*(*player*) ▷ If we have a win move
if *winMove* exists **then**
  return *winMove*

forceMove ← *getForceMove*(*player*) ▷ If we have a force move
if *forceMove* exists **then**
  return *forceMove*

coreMove ← *getFirstTwoSteps*(*player*) ▷ Occupy the strongest points
if *coreMove* exists **then**
  return *coreMove*

maxValue ← $-\infty$
positionTicTacToe myNextMove

do ▷ Progressive deepening
  for <each available move *curMove*> **do**
    <make current move>
    newVale ← *miniMax*(*depth*, *player*, *false*, $-\infty$, $+\infty$)
    if *newValue* > *maxValue* **then**
      maxValue ← *newValue*
      myNextMove ← *curMove*
    <cancel current move> ▷ Backtracking
while <time is still enough>
return *myNextMove*

2. Pseudocode of miniMax

Algorithm 2 miniMax

```
Input int depth, int player, boolean maximizingPlayer, int alpha, int beta
Return int value
procedure MINIMAX
    if depth == 0 then                                          ▷ search finish
        return evaluation(player)                ▷ evaluate the board configutaion

    if maximizingPlayer then                                    ▷ Maximizer
        value ← −∞
        for <each available move curMove> do
            winMove ← getWinMove(player)                  ▷ win move pruning
            if winMove exists then
                <make this win move>
                value ← evaluation(player)
                <cancel this win move>                          ▷ Backtracking
                break
            forceMove ← getForceMove(player)              ▷ force move pruning
            if forceMove exists then
                <make this force move>                    ▷ DFS and deepening
                value ← max(value, miniMax(depth, player, false, alpha, beta)
                <cancel this force move>                        ▷ Backtracking
            else                                                ▷ Naive miniMax
                <make current move>                                   ▷ DFS
                value ←
                max(value, miniMax(depth − 1, player, false, alpha, beta))
                <cancel current move>                           ▷ Backtracking
                alpha ← max(alpha, value)
                if alpha >= beta then
                    break
        return value
    else                                                        ▷ Minimizer

        value ← +∞
        opponent ←!player
        for <each available move curMove> do
            winMove ← getWinMove(opponent)                ▷ win move pruning
            if winMove exists then
                <make this win move>
                value ← evaluation(player)
                <cancel this win move>                          ▷ Backtracking
                break
            forceMove ← getForceMove(opponent)            ▷ force move pruning
            if forceMove exists then
                <make this force move>                    ▷ DFS and deepening
                value ← min(value, miniMax(depth, player, true, alpha, beta))
                <cancel this force move>                        ▷ Backtracking
            else                                                ▷ Naive miniMax
                <make current move>                                   ▷ DFS
                value ←
                min(value, miniMax(depth − 1, player, true, alpha, beta))
                <cancel current move>                           ▷ Backtracking
                beta ← min(beta, value)
                if alpha >= beta then
                    break
        return value
```

6 Results and Discussion

6.1 Battle with randomness AI

```
407     //run the game once
408     public static void main(String[] args) {
409
410         //run game loop
411         /*runTicTacToe rttt = new runTicTacToe();
412         long time1 = System.currentTimeMillis();
413         rttt.run();
414         long time2 = System.currentTimeMillis();
415         System.out.println("Program run time: " + (time2 - time1) + " ms");*/
416
417         int rounds = 100;
418
419         while( rounds > 0 ) {
420             runTicTacToe rttt = new runTicTacToe();
421             long time1 = System.currentTimeMillis();
422             rttt.run();
423             long time2 = System.currentTimeMillis();
424             System.out.println("Program run time: " + (time2 - time1) + " ms");
425             rounds--;
426             System.out.println("Player " + firstPlayer + " is the first player.");
427             System.out.println("player1: " + player1wins + " player2: " + player2wins);
428             System.out.println("Player 1 Overtime: " + overtime);
429         }
430     }
431 }
432
433
434
```

Problems Declaration Console

```
<terminated> runTicTacToe (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.0.2.jdk/Contents/Home/bin/java (2019年1
[_____]

level(z) 3
[___X]
[_OO_]
[____]
[____]

Program run time: 19338 ms
Player 1 is the first player.
player1: 100 player2: 0
Player 1 Overtime: 0
```

Our AI can completely beat randomness AI. Player 1 is our AI player and player 2 is randomness AI. The first player is chosen randomly.

6.2 AI vs. AI with the same settings

Battle between final version as a first player and final version as a second player.

```
Player 1 is the first player.
player1: 32 player2: 68
```

Result: the second player performs much better than the first player. The second players win 68 rounds in 100.

The reason may be that

- the second player has much more time to do progressive deepening as the number of available moves gets less and less.
- Besides, the heuristic value is not always correct.

```
private static final int[] playerSequenceValue = new int[] {1, 15, 130, 100000};
private static final int[] opponentSequenceValue = new int[] {-1, -10, -100, -100000};
```

- Additionally, in the first move of each player, they randomly choose one

strongest point in the center of the cubic, which means the first step is extremely important for each player in this game.

6.3 AI vs. AI with different settings

Player 1 settings:

```
private static final int[] playerSequenceValue = new int[] {1, 15, 150, 100000};
private static final int[] opponentSequenceValue = new int[] {-1, -10, -100, -100000};
```

Player 2 settings:

```
private final int[] playerSequenceValue = new int[] {1, 15, 130, 100000};
private static final int[] opponentSequenceValue = new int[] {-1, -10, -100, -100000};
```

Result:
Player 1 wins 12 rounds in 50, when player 1 is the first player.
Player 1 wins 14 rounds in 50, when player 2 is the first player.
The result shows that the settings of player 2 is much better than those of player 1, because no matter who is the first player, player 2 always wins.

6.4 AI behavior analysis

See more details in appendix. In the first two steps, AI occupy the strongest points in the center of the cubic randomly, which makes our AI not totally deterministic. Then, AI will take move according to heuristic value. Before that, AI will find a win move or a force move first. AI will fully make use of 10 seconds to search a better move.

The first player is much worse than the second. We make a possible explanation. The second can search more when the board is getting smaller, which means the second player can search more possible moves to decide which move to take.

The random process in the first two step can have critical influence on the result of the game, which means the first two steps is of significant importance.

7 Conclusions

In this assignment, we develop an AI for 4×4×4 Tic-Tac-Toe Game. We adopt minimax with alpha-beta pruning to make a move decision. Besides, we improve our AI by changing the data structure and using progressive deepening and heuristic pruning. As a result, our AI can look ahead for a least 4 moves and completely defeat the randomness AI. When doing self-battling, the second player using our AI performs better than the first hand player.

References

[1] Patashnik, Oren. "**Qubic: 4 × 4 × 4 Tic-Tac-Toe**." Mathematics Magazine 53, no. 4 (1980): 202-16. doi:10.2307/2689613.
[2] **3D Tic Tac Toe Algorithms** - Rochester CS
https://www.cs.rochester.edu/u/brown/242/assts/studprojs/ttt10.pdf

Appendix

AI behaviors.
Both players are our AI. Player 1 is the first player.

```
Player1' turn:
Evaluation 0
Core move
myNextMove: 2 1 2
Player 1 run time: 0 ms
Player2' turn:
Evaluation -7
Core move
myNextMove: 2 2 1
Player 2 run time: 0 ms
level(z) 0
[____]
[____]
[____]
[____]

level(z) 1
[____]
[____]
[__O_]
[____]

level(z) 2
[____]
[____]
[_X__]
[____]

level(z) 3
[____]
[____]
[____]
[____]


Player1' turn:
Evaluation 0
```

```
myNextMove: 0 1 2 Value: 59
Player 1 run time: 8158 ms
Player2' turn:
Evaluation -12
myNextMove: 1 1 2 Value: 34
Player 2 run time: 9802 ms
level(z) 0
[____]
[____]
[____]
[____]

level(z) 1
[____]
[____]
[__O_]
[____]

level(z) 2
[_X__]
[_O__]
[_X__]
[____]

level(z) 3
[____]
[____]
[____]
[____]


Player1' turn:
Evaluation 2
myNextMove: 0 1 0 Value: 134
Player 1 run time: 9803 ms

Player2' turn:
Evaluation -12
myNextMove: 0 0 3 Value: 237
Player 2 run time: 9802 ms
level(z) 0
[_X__]
[____]
[____]
```

```
[____]

level(z) 1
[____]
[____]
[__O_]
[____]

level(z) 2
[_X__]
[_O__]
[_X__]
[____]

level(z) 3
[O___]
[____]
[____]
[____]


Player1' turn:
Evaluation -64
Force move.
myNextMove: 3 3 0
Player 1 run time: 0 ms


Player2' turn:
Evaluation 102
myNextMove: 1 1 1 Value: 356
Player 2 run time: 9801 ms
level(z) 0
[_X__]
[____]
[____]
[___X]

level(z) 1
[____]
[_O__]
[__O_]
[____]
```

```
level(z) 2
[_X__]
[_O__]
[_X__]
[____]

level(z) 3
[O___]
[____]
[____]
[____]


Player1' turn:
Evaluation -80
myNextMove: 0 0 0 Value: 22
Player 1 run time: 9801 ms
Player2' turn:
Evaluation 112
myNextMove: 0 0 1 Value: 223
Player 2 run time: 8258 ms
level(z) 0
[XX__]
[____]
[____]
[___X]

level(z) 1
[O___]
[_O__]
[__O_]
[____]

level(z) 2
[_X__]
[_O__]
[_X__]
[____]

level(z) 3
[O___]
[____]
[____]
[____]
```

```
Player1' turn:
Evaluation -148
Force move.
myNextMove: 3 3 1
Player 1 run time: 0 ms
Player2' turn:
Evaluation 231
myNextMove: 1 1 0 Value: 107
Player 2 run time: 9801 ms
level(z) 0
[XX__]
[_O__]
[____]
[___X]

level(z) 1
[O___]
[_O__]
[__O_]
[___X]

level(z) 2
[_X__]
[_O__]
[_X__]
[____]

level(z) 3
[O___]
[____]
[____]
[____]


Player1' turn:
Evaluation -224
Force move.
myNextMove: 1 1 3
Player 1 run time: 0 ms

Player2' turn:
Evaluation 345
```

```
myNextMove: 0 0 2 Value: -17
Player 2 run time: 9801 ms
level(z) 0
[XX__]
[_O__]
[____]
[___X]

level(z) 1
[O___]
[_O__]
[__O_]
[___X]

level(z) 2
[OX__]
[_O__]
[_X__]
[____]

level(z) 3
[O___]
[_X__]
[____]
[____]


Player1' turn:
Evaluation -321
myNextMove: 0 1 3 Value: 100349
Player 1 run time: 9808 ms


Player2' turn:
Evaluation 375
Force move.
myNextMove: 0 1 1
Player 2 run time: 0 ms
level(z) 0
[XX__]
[_O__]
[____]
[___X]
```

```
level(z) 1
[OO__]
[_O__]
[__O_]
[___X]

level(z) 2
[OX__]
[_O__]
[_X__]
[____]

level(z) 3
[OX__]
[_X__]
[____]
[____]


Player1' turn:
Evaluation -210
myNextMove: 0 3 0 Value: 100468
Player 1 run time: 9805 ms

Player2' turn:
Evaluation 285
Force move.
myNextMove: 0 2 0
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O__]
[____]
[___X]

level(z) 1
[OO__]
[_O__]
[__O_]
[___X]

level(z) 2
[OX__]
[_O__]
```

```
[_X__]
[____]

level(z) 3
[OX__]
[_X__]
[____]
[____]


Player1' turn:
myNextMove: 1 3 0 Value: 100574
Player 1 run time: 9806 ms

Player2' turn:
Evaluation 196
Force move.
myNextMove: 2 3 0
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[___O]
[___X]

level(z) 1
[OO__]
[_O__]
[__O_]
[___X]

level(z) 2
[OX__]
[_O__]
[_X__]
[____]

level(z) 3
[OX__]
[_X__]
[____]
[____]
```

```
Player1' turn:
Evaluation 52
myNextMove: 3 0 3 Value: 100698
Player 1 run time: 9803 ms

Player2' turn:
Evaluation 101
Force move.
myNextMove: 1 2 1
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[___O]
[___X]

level(z) 1
[OO__]
[_OO_]
[__O_]
[___X]

level(z) 2
[OX__]
[_O__]
[_X__]
[____]

level(z) 3
[OX__]
[_X__]
[____]
[X___]


Player1' turn:
Evaluation 160
myNextMove: 2 1 3 Value: 100836
Player 1 run time: 9622 ms

Player2' turn:
Evaluation 51
Force move.
myNextMove: 3 1 3
```

```
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[___O]
[___X]

level(z) 1
[OO__]
[_OO_]
[__O_]
[___X]

level(z) 2
[OX__]
[_O__]
[_X__]
[____]

level(z) 3
[OX__]
[_X__]
[_X__]
[XO__]


Player1' turn:
Evaluation 292
myNextMove: 0 3 3 Value: 101048
Player 1 run time: 9804 ms

Player2' turn:
Evaluation -60
Force move.
myNextMove: 1 2 3
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[___O]
[___X]

level(z) 1
[OO__]
```

```
[_OO_]
[__O_]
[___X]

level(z) 2
[OX__]
[_O__]
[_X__]
[____]

level(z) 3
[OX_X]
[_XO_]
[_X__]
[XO__]


Player1' turn:
Evaluation 453
myNextMove: 1 3 2 Value: 101207
Player 1 run time: 9099 ms
Player2' turn:
Evaluation -144
Force move.
myNextMove: 2 3 1
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[___O]
[___X]

level(z) 1
[OO__]
[_OO_]
[__OO]
[___X]

level(z) 2
[OX__]
[_O_X]
[_X__]
[____]
```

```
level(z) 3
[OX_X]
[_XO_]
[_X__]
[XO__]


Player1' turn:
Evaluation 564
Time out alert!
myNextMove: 2 0 2 Value: 101392
Player 1 run time: 9808 ms
Player2' turn:
Evaluation -215
Force move.
myNextMove: 1 0 1
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[___O]
[___X]

level(z) 1
[OO__]
[OOO_]
[__OO]
[___X]

level(z) 2
[OX__]
[_O_X]
[XX__]
[____]

level(z) 3
[OX_X]
[_XO_]
[_X__]
[XO__]


Player1' turn:
Evaluation 594
```

```
Force move.
myNextMove: 1 3 1
Player 1 run time: 0 ms
Player2' turn:
Evaluation -193
Force move.
myNextMove: 1 3 3
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[___O]
[___X]

level(z) 1
[OO__]
[OOOX]
[__OO]
[___X]

level(z) 2
[OX__]
[_O_X]
[XX__]
[____]

level(z) 3
[OX_X]
[_XOO]
[_X__]
[XO__]


Player1' turn:
Evaluation 718
myNextMove: 2 1 1 Value: 101495
Player 1 run time: 9804 ms
Player2' turn:
Evaluation -275
Force move.
myNextMove: 2 1 0
Player 2 run time: 0 ms
level(z) 0
[XXOX]
```

```
[_O_X]
[_O_O]
[___X]

level(z) 1
[OO__]
[OOOX]
[_XOO]
[___X]

level(z) 2
[OX__]
[_O_X]
[XX__]
[____]

level(z) 3
[OX_X]
[_XOO]
[_X__]
[XO__]


Player1' turn:
Evaluation 845
myNextMove: 1 2 2 Value: 101575
Player 1 run time: 9808 ms
Player2' turn:
Evaluation -357
Force move.
myNextMove: 3 0 0
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[_O_O]
[O__X]

level(z) 1
[OO__]
[OOOX]
[_XOO]
[___X]
```

```
level(z) 2
[OX__]
[_OXX]
[XX__]
[____]

level(z) 3
[OX_X]
[_XOO]
[_X__]
[XO__]


Player1' turn:
Evaluation 969
myNextMove: 3 0 2 Value: 101575
Player 1 run time: 9802 ms
Player2' turn:
Evaluation -433
Force move.
myNextMove: 0 3 2
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[_O_O]
[O__X]

level(z) 1
[OO__]
[OOOX]
[_XOO]
[___X]

level(z) 2
[OX_O]
[_OXX]
[XX__]
[X___]

level(z) 3
[OX_X]
[_XOO]
[_X__]
```

```
[XO__]


Player1' turn:
Evaluation 1101
myNextMove: 3 3 2 Value: 101575
Player 1 run time: 9802 ms
Player2' turn:
Evaluation -525
Force move.
myNextMove: 3 3 3
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[_O_O]
[O__X]

level(z) 1
[OO__]
[OOOX]
[_XOO]
[___X]

level(z) 2
[OX_O]
[_OXX]
[XX__]
[X__X]

level(z) 3
[OX_X]
[_XOO]
[_X__]
[XO_O]


Player1' turn:
Evaluation 1198
myNextMove: 3 2 2 Value: 101575
Player 1 run time: 9802 ms
Player2' turn:
Evaluation -554
Force move.
```

```
myNextMove: 3 1 2
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[_O_O]
[O__X]

level(z) 1
[OO__]
[OOOX]
[_XOO]
[___X]

level(z) 2
[OX_O]
[_OXX]
[XX__]
[XOXX]

level(z) 3
[OX_X]
[_XOO]
[_X__]
[XO_O]


Player1' turn:
Evaluation 1309
myNextMove: 2 2 2 Value: 101575
Player 1 run time: 9802 ms
Player2' turn:
Evaluation -733
Force move.
myNextMove: 0 2 2
Player 2 run time: 0 ms
level(z) 0
[XXOX]
[_O_X]
[_O_O]
[O__X]

level(z) 1
[OO__]
```

```
[OOOX]
[_XOO]
[___X]

level(z) 2
[OXOO]
[_OXX]
[XXX_]
[XOXX]

level(z) 3
[OX_X]
[_XOO]
[_X__]
[XO_O]

Player1' turn:
Evaluation 1474
Win move
myNextMove: 2 3 2
Player 1 run time: 0 ms

(2,0,2)state: 1
(2,1,2)state: 1
(2,2,2)state: 1
(2,3,2)state: 1
Player1 Wins
level(z) 0
[XXOX]
[_O_X]
[_O_O]
[O__X]

level(z) 1
[OO__]
[OOOX]
[_XOO]
[___X]

level(z) 2
[OXOO]
[_OXX]
[XXXX]
[XOXX]
```

```
level(z) 3
[OX_X]
[_XOO]
[_X__]
[XO_O]

Program run time: 221420 ms
Player 1 is the first player.
```