Instruction for PA3: 4x4x4 Tic-Tac-Toe Game

There are several rules you need to know and follow in PA3:

*You can make a team of up to three members.
***Each student** must submit code along with the report.
*Your **last** submission is the only valid submission. Students must submit code along with the report.
*If you submit your work **after 0:00** (starting from 0:01), 20% is the minimum deduction for late penalty. **The submission time** from gsubmit system is the only valid time we judge whether your work is submitted on time.
*We do not accept other forms of submission, including but not limited to **emails** or **shared Google Drive link.**
*There will be **no extension** for this assignment. If you need a personal extension, which we only allow in rare cases, you need to contact Profesor Betke for permisson. **Your request will be recorded and accounted for in your final grade.**

In this assignment, you are going to:

1. Learn what is a tic-tac-toe game and how to win it.
2. Implement minimax or alpha-beta pruning for a tic-tac-toe game AI to break the 4x4x4 game.
3. Compete with AI implemented from other and win a tournament with extra credits!

This assignment is going to take 10-20 hour to complete, depending on your programming skill in Java and understanding of game AI algorithms.
Before getting started, please download the skeleton code used for PA3.

A Java development environment is **required** to complete this assignment, since we assume you have taken relevant class(es) for Java programming as our course required, we won't provide specific help in setting up Java development environment or explaining basic programming knowledge in Java. (Deep copy and shallow copy in Java will be an important thing to be understood before write your PA3 code)

Warm up:

Before diving in game AI and a much more complex form of tic-tac-toe, let's play a few games of the simplest form of tic-tac-toe, a 2-D 3x3 tic-tac-toe game:
https://playtictactoe.org/

If you haven't played any tic-tac-toe game before, take a look at the wikipedia page about tic-tac-toe (or noughts and crosses, as its British name).
In general, a 3x3 tic-tac-toe is
"a game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game." -- from wikipedia

Before heading to the next step, think and answer the following questions:
(you may write your answer down below to help you understand the game)

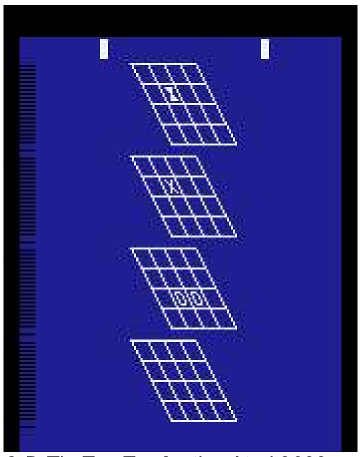
To win the game, which position will you mark first as player going first? Are you always going to win as player going first? If not, why?

It is obvious that 3x3 tic-tac-toe is always going to be a draw if both players know the game well enough and always make the best move. Besides, 3x3 tic-tac-toe is too simple that it is possible for computer and even human to find out all the possible conditions in the game by brute-force searching. Thus, to make the game more interesting to play, let's make it a little bit more complicated…

Part 1: A 4x4x4 Cubic Game

Now, let's consider a cubic variation of tic-tac-toe, and with more rows and columns, which are 4x4x4.
A Wikipedia introduction can be seen in here:
https://en.wikipedia.org/wiki/3D_tic-tac-toe#cite_note-7

The rules are still similar,
"players take turns placing their markers in blank cells in the array. The first player to achieve four of their own markers in a row wins. The winning row can be horizontal, vertical, or diagonal on a single board as in regular tic-tac-toe, or vertically in a column, or a diagonal line through four boards." --from wikipedia


3-D Tic-Tac-Toe for the *Atari 2600*

You can play cubic tic-tac-toe on an Atari 2600 console… well, not necessarily. So it's time to take a look at the provided skeleton code:

There are 3 .java files compressed into our .zip file. They are:
aiTicTacToe.java, positionTicTacToe.java and runTicTacToe.java

aiTicTacToe.java: this file contains a function called “myAIAlgorithm(...)” to decide the next position your AI is going to mark, which will be called by file runTicTacToe.java every time it’s your AI’s turn. Remember, this is the **<u>ONLY FILE</u>** you need to submit in your PA3 submission.

positionTicTacToe.java: a helper class is given under this file called positionTicTacToe, storing position information and its current state for all positions in the cube for a particular game.

runTicTacToe.java: this file contains the main function we use to run the 4x4x4 tic-tac-toe game. We do not suggest you to change things on it frequently unless you have to for AI debugging. However, after changing runTicTacToe.java file, you need to make sure your AI code is still compatible with the original runTicTacToe.java, because we need to use it for competition with other AIs.

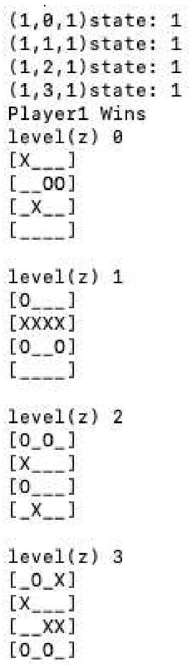
To test the code whether it can be run on your Java development environment, open your terminal (or command prompt), and if you are sure your Java environment is set up successfully, go to the root path of our skeleton code, which contains the three .java files to be compiled, run:

```
javac runTicTacToe.java
```

If .class files are generated, and no error pops up, good!

Then, run:

```
java runTicTacToe
```



```
(1,0,1)state: 1
(1,1,1)state: 1
(1,2,1)state: 1
(1,3,1)state: 1
Player1 Wins
level(z) 0
[X___]
[__OO]
[_X__]
[____]

level(z) 1
[O___]
[XXXX]
[O__O]
[____]

level(z) 2
[O_O_]
[X___]
[O___]
[_X__]

level(z) 3
[_O_X]
[X___]
[__XX]
[O_O_]
```

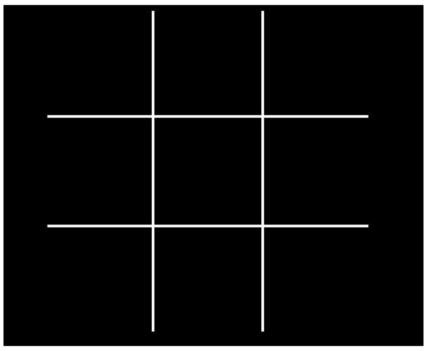
If you see results similar to this, well, it seems your computer just played a 4x4x4 tic-tac-toe game! Repeat these two lines of command each time when you want to run a 4x4x4 tic-tac-toe game with your AI.

Okay, too much to digest on the code side for now, let’s work on the cubic tic-tac-toe before getting deep.

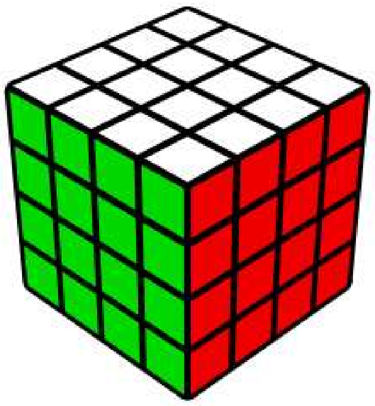
Let’s think about the following questions and try to answer them:

How many winning lines are there in a 3x3 tic-tac-toe game? What are they?

By saying winning lines, they are combination of all “X” or all “O” positions which satisfy the winning condition, regardless of marks in other positions that haven’t resulted in an end condition.

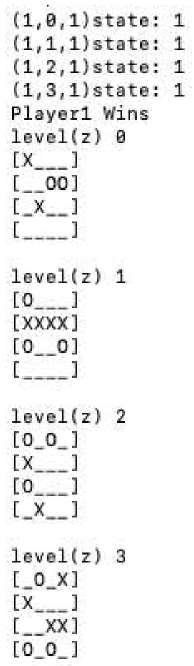


How many winning lines are there in 4x4x4 tic-tac-toe game? What are they?



The answer for 3x3 game is 8 and 76 for a 4x4x4 game, and it’s crucial to understand what are they during implementing your AI to break the game.

Now let’s get back to the screenshot we previously have and see what it is:



```
(1,0,1)state: 1
(1,1,1)state: 1
(1,2,1)state: 1
(1,3,1)state: 1
Player1 Wins
level(z) 0
[X___]
[__OO]
[_X__]
[____]

level(z) 1
[O___]
[XXXX]
[O__O]
[____]

level(z) 2
[O_O_]
[X___]
[O___]
[_X__]

level(z) 3
[_O_X]
[X___]
[__XX]
[O_O_]
```

To represent specific position in a cube space for tic-tac-toe game, we separate the cube into four 4x4 planes by their height in the cube, e.g., "level(z) 0" is the lowest plane among all 4 planes. We use "z" value to represent the height of each position (plane), and accordingly, "x" and "y" are also used for representing the 2-D information for every position on each plane. For example, the "X" in level 0 on the upper left corner is in the position (0,0,0). And as you can see, position (1,0,1), position (1,1,1), position (1,2,1) and position (1,3,1) are 4 "X"s on level(z) 1, which makes player 1 (the "X") win the game.

According to the rule, winning lines are not always on the same plane for a cubic tic-tac-toe game so it's a little bit difficult to find them out with "O" and "X" on each planes. You might know why if you have figured all the winning lines in a 4x4x4 game. Therefore, our result also provide the coordinates for the winning line position, and "state" shows which player has marked these positions, which can be 0 (unmarked), 1 (marked by player 1), 2 (marked by player 2) and -1 (used for initialization, can be considered as "unknown").

If you want to change the printing result, take a look at function "printBoardTicTacToe(...)" and function "isEnded(...)" on runTicTacToe.java file.

Before start implementing AI to win the game, please spend some more time understanding how the program starts a game and ends a game by calling various functions inside. After that, you will find out our current game are played by two AI players randomly mark any unmarked position in the cube.

Part 2: Game AI, Minimax as an Example

Hopefully you still save some energy when we get here, because we are about to get our hands dirty.
There're various kinds of strategies and algorithms which are able to dominate this game, but to ensure everyone have relatively same efforts and workload not to be too low or too high, **only alpha-beta pruning and minimax are allowed to be submitted and graded** as AI algorithms used in your AI program and be presented in demo.
We assume you have already understood what minimax and alpha-beta pruning are on the class and how they can be used, so we won't explain these ideas in this instruction.

Let's go to the pseudo-code from wikipedia and have a look, here we use minimax as an example:

(from wiki https://en.wikipedia.org/wiki/Minimax)

```
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
```

As for alpha-beta pruning, you can find its pseudo code in:
https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

Think about these questions before coding:

1. What are the “node” in our tic-tac-toe game? What are the children of “node” then?
2. What is the “terminal node” in our tic-tac-toe game?

Without knowing what they are, minimax will be impossible to be implemented in your code. So spend some time, take a coffee, have a snap, or simply take a deep breath to answer these questions first.

If you successfully find out what “node” refers to in our tic-tac-toe game, you are almost able to implement your minimax in our code! But for one last thing:

The heuristic value of node.

So you need to give each node a value telling the algorithm how important a position is under certain conditions. There are no perfect answer, at least it seems, for this question, regarding efficiency to compute the value and whether the node is leading to a certain win. **This is going to influence how well your AI is going to be in this game.** There are at least two issues related to the decision of heuristic value for each node:

1. How to make sure your AI always prevent the opponent from winning in the next move if your AI still have a chance to do so by correct values for nodes?
2. If there is a move for your AI to win in the next move, is your AI able to find out the move also by correct values for nodes?

Your decision of heuristic value of node should at least ensure these two requirements are both satisfied. And your AI should be able to make a decision under 5.0 seconds in normal PC.

The hardest part in this assignment is not implementing the minimax, but implementing a way of computing the heuristic value of node both efficiently and “correctly”.
It’s okay to change or add anything you want in our skeleton code, however, make sure:

1. myAIAlgorithm(...) provides the next step your AI choose.
2. Your final aiTicTacToe.java is compatible with our original runTicTacToe.java file.

Now, write, run, test and polish your AI to compete with AI in random behaviour or another AI you developed yourself before sending it to our AI competition!

Part 3: Grading Criteria

In the end, to ensure you can get a good AI as well as good scores, you need to show us several things, on gsubmit system, as well as during presentation.

1. Your code, which is **aiTicTacToe.java** you modify for your AI.
2. A report, similar to your programming assignment one and two. You can write it based on several questions you answer above and code you implement in this instruction.

Grades:
50% for code and 50% for report, and **50% extra credits at most for winning the AI tournament against other students**.
Details of grading criteria and AI tournament will be explained in another post on Piazza.

Looking forward to see your AI in demo presentation!

Some backgrounds if you are interested:

Qubic: 4 × 4 × 4 Tic-Tac-Toe