CSCI6511 Project 2: Tic Tac Toe

Work Distribution

We are team 1197 (Diet Water), a group of 4 people:

Binren Wang – Searching algorithm and multi-threading

Qinyang Li – Chessboard base function, testing, playing

Zhechao Wang – Send/receive message to/from API

Zemao Song – Agent movement and documentation

General Description

It is a AI working on playing generalized Tic Tac Toe with provided web API on an n*n board with target x. User could either start a game or join a game with corresponding input. Then the AI would work automatically until the game ends (win, lose or draw).

Environment

Language: Java 1.8

Library: fastjson 1.2.62 (For parsing return value in json format)

Files

Requests.java: Functions to send requests (get/post) to contact the API

Play.java: A starter of the game

Game.java: Main parts, including the AI and game judgement.

/out/artifacts/Project2_jar/Project2.jar: Runnable project package

The Agent

Minimax search tree with alpha-beta pruning: The searching algorithm is implemented as a 3-level search tree with alpha-beta pruning. In each step, we generate a root node with current condition (level 0). After that, we grow the search tree level by level – For each node in level n-1, find all the "useful" moves on the board for the next player and create nodes at level n as its children. After evaluate (or prune) all the leaf nodes with multi-thread work, we could get the most optimal move(s). Considering the optimal node of 3-level search tree contains 3 steps: 2 for ourselves and 1 for opponent, we select the move from the 2 that gives out a better temporary evaluation score.

Multi-thread programming: At the start of the game, the program detects the available processors to decide the number of threads. During searching, each thread would take a node and evaluate until either time run out or finishing the evaluation of all nodes.

Evaluation Function

The score of n piece in a row depends on target value. For $2 \leq n < target - 2$, one more pieces leads to a double score while for $n \geq target - 2$, the score has a great boost as it's close to win. A 3D array (boardSum/nodeBoardSum) is used to store the current continuous pieces in a row till any place on board in 4 directions. With this array, the evaluation score of each case and whether the game ends at the case could be easily calculated.

The evaluation score could be calculated by adding the score of our group, timing a coefficient and deducting the score of opponent group. The coefficient is based on the next player. If next player is opponent, which means a potential threat and coefficient would be less than 1, vice versa.

Runtime

Run the Project2.jar at /out/artifacts/Project2_jar/ with command:

java -jar Project2.jar

After that, follow the instructions to join a game or create a game. Example 1:

Join game 260, board size is 12, target is 6, you are the first to move. (You may join a game half-way as long as it is your game and has not ended yet)

```
Have you already created a game? (Y/N)
Y
Please enter the game id, board size, target and priority of our team (1 or 2) to join in (each element separated by space):
260 12 6 1
```

Example 2: Create a game with team 1200, board size is 12, target is 6, you are the second to move.

```
Have you already created a game? (Y/N)
N
Please enter the team id of opponent, board size, target and priority of our team (1 or 2) to create a game (each element separated by space):
1200 12 6 2
```

After initialization and every step, the program would print out the latest board (and other status). Example:

```
INFO: Turn 8
Opponent move = (5, 6)
   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
1  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
4  -  -  -  -  X  -  -  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  X  -  -  -  -  -  -  -  -
6  -  -  -  -  -  X  X  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  0  -  -  -  -  -  -  -
8  -  -  -  -  -  -  0  0  -  -  -  -  -  -  -
9  -  -  -  -  -  -  0  -  -  -  -  -  -  -  -
10 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
11 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
12 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
13 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
14 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -

INFO: Turn 9
Created 26208 leaf nodes
Pruned 18669 leaf nodes
Max score = 54
Move = (9, 7), time cost of move = 1 seconds
   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
1  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
4  -  -  -  -  X  -  -  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  X  -  -  -  -  -  -  -  -
6  -  -  -  -  -  X  X  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  0  -  -  -  -  -  -  -
8  -  -  -  -  -  -  0  0  -  -  -  -  -  -  -
9  -  -  -  -  -  -  0  0  -  -  -  -  -  -  -
10 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
11 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
12 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
13 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
14 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
```