Comp2230/6230 Algorithms Assignment
Due: Friday, Week 12, 2019-11-01 23:59

Task: Design an engine that plays the two player game *connect four*.

Connect Four

Connect four is a two player game. The objective of the game is to connect four markers in sequence, in either a row, column or diagonal, while preventing your opponent from doing the same. The game is won when a player makes four markers in a sequence.

The game is played by placing a marker in one of seven columns, and the marker is placed on the lowest row in that column. There are six rows in each column to make a play field of 7×6. Players take turns placing markers.

Requirements

- Language: Java 1.8
- The engine must use `minimax`, or some derivation of `minimax`, such as the refactored `negamax` or the faster, but more complex alpha-beta pruning heuristic.
- The evaluation function associated with the engine can be as complicated or as simple as you like, as long as it correctly identifies a win, loss, or draw with appropriate values. Appropriate values might be for example a large positive number for the first player winning, a large negative number for the second player winning, and 0 for a draw, although this specific heuristic is not required, other heuristics can work.
- The engine must parse the input correctly to set up the game state internally. It must also pass out the output in the correct format.
- There is no fixed requirement on how to store the game state, whichever you deem to be the best for your individual engine is acceptable.
- All communication will be done over standard input and output.
- The engine must correctly behave under all possible coordinator commands.

Coordinator Commands

The following are the commands required to interface with the connect four coordinator. All commands will be sent to the engine via `System.in` in java, and all responses sent from the engine to the coordinator should be done via `System.out.println` or `System.out.print`. Note that a new line character needs to be at the end of each command, so using the first option is probably easiest. Each command is case sensitive.

Sent From the Coordinator

- `name` – request for the engines name. The reply should be the name of the engine. The name should be whatever name you want to give it, followed by your student number, that is, `enginename-cXXXXXXX`.
- `isready` – A command used to synchronise the engine with the coordinator. The reply should be `readyok`.
- `position startpos <moves>` – The current order of moves played from the start position. For example, `position startpos 0123456`. Only the numbers $0-6$ will ever be used. This command will always be followed by the `isready` command. The only operation allowed to be performed by the engine between receiving this command is to update the internal board with the moves, no searching is allowed.
- `go ftime x stime y` – Tells the engine to start calculating. The `x` string will be the time remaining for the first player in milliseconds. The `y` string will be how long the second player has left. The reply should be `bestmove z v` where `z` is the column number, $0-6$, that the engine wants to place its next marker in. If it is invalid it will be an automatic forfeit. `v` is the value of the evaluation function after that move. All searching is done here.
- `perft x` – Tells the engine to count how many nodes are in the entire search tree when expanded to depth `x` from the current position. The reply should be `perft x y` where `x` is the same value passed in, and `y` is the number of nodes in the search tree when it is expanded to depth `x`. This should be a separate algorithm to your minimax algorithm, mostly used for debugging and performance testing (hence the name) purposes and be ready to take in any depth.
- `quit` – Tells the engine to quit. Reply should be `quitting`.

Sent From Your Engine

- `readyok` – The response to any `isready` command only when the engine is ready.
- `bestmove z v` – The response to the `go` command. `z` is the column the engine wants the next marker in, labeled $0-6$. `v` is the value of the evaluation function after that move.
- `info` – If used, must be after a `go` command from the coordinator, and before the `bestmove` response from the engine. It can be followed by a number of optional strings. Each string must be paired with the corresponding data. Not a requirement to use `info`, it is mainly used for debugging purposes.
  - `depth x` – The depth searched by the engine so far.
  - `nodes x` – The number of nodes searched by the engine so far.
  - `score x` – The current best evaluation by the engine so far.
  - `string x` – Any string the engine wishes to display. If there is a string command, the rest of the line will be interpreted as the string corresponding to the string command.
- `quitting` – The response to the command `quit`. The engine should exit at this point.

Coordinator.jar

A file called `connect_4_coordinator.jar` has been supplied. It is created against java 1.8. It will be used to verify your engine is working correctly and implements everything mentioned in the coordinator commands section. To use the file refer to your IDE's documentation to see how to run a jar file in it, and adjust the command line arguments for it. If you wish to use it on the command

line to run it, I would advise using an ANSI complaint terminal, although this is not necessary. To run the file from the command line you need to use `java -jar connect_4_coordinator.jar` along with the arguments to pass to the program. The arguments are:

- (required) `e0 <path>` where `<path>` is the directory containing the file called `Interface` which includes the main method of your engine.
- (optional) `e1 <path>` where `<path>` is the directory containing a file called `Interface` which includes the main method of another engine. If this is not used the first engine will be used again, so it is easy to set up a self game for the engine.
- (optional) `time x` where `x` is the amount of time in milliseconds each engine is allowed to use. If this is not used, `x` will be 120000 (2 minutes).
- (optional) `perft x` where `x` is the depth the engine should calculate the `perft` function to.

Sample Communication

The following is the communication of the setup and first two moves of a game of the same engine playing against itself.

```
Engine 0 set up.
Engine 1 set up.
Coordinator->AI0:       name AI 0
AI0->Coordinator:               testengine-c3127893
Coordinator->AI1:       name AI 1
AI1->Coordinator:               testengine-c3127893
Coordinator: Duplicate names, adjusted names.
Coordinator: Engine 0 name is: testengine-c3127893-0
Coordinator: Engine 1 name is: testengine-c3127893-1
Coordinator->testengine-c3127893-0:         isready AI 0
testengine-c3127893-0->Coordinator:         readyok
Coordinator->testengine-c3127893-1:         isready AI 1
testengine-c3127893-1->Coordinator:         readyok
Coordinator->testengine-c3127893-0:         position startpos
Coordinator->testengine-c3127893-0:         isready AI 0
testengine-c3127893-0->Coordinator:         readyok
Coordinator->testengine-c3127893-0:         go ftime 120000 stime 120000
testengine-c3127893-0->Coordinator:         info depth 0 nodes 1 score 0 string this is a string
testengine-c3127893-0->Coordinator:         bestmove 0 0
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
| x |   |   |   |   |   |   |
_____________________________
Coordinator->testengine-c3127893-1:         position startpos 0
Coordinator->testengine-c3127893-1:         isready AI 1
testengine-c3127893-1->Coordinator:         readyok
Coordinator->testengine-c3127893-1:         go ftime 120000 stime 120000
testengine-c3127893-1->Coordinator:         info depth 0 nodes 1 score 0 string this is a string
testengine-c3127893-1->Coordinator:         bestmove 1 0
```

The following is the setup of an engine, along with the perft test of an engine.

```
Coordinator->AI0:       name AI 0
AI0->Coordinator:               testengine-c3127893
Coordinator->AI1:       name AI 1
AI1->Coordinator:               testengine-c3127893
Coordinator: Duplicate names, adjusted names.
Coordinator: Engine 0 name is: testengine-c3127893-0
Coordinator: Engine 1 name is: testengine-c3127893-1
Coordinator->testengine-c3127893-0:         isready AI 0
testengine-c3127893-0->Coordinator:         readyok
Coordinator->testengine-c3127893-1:         isready AI 1
testengine-c3127893-1->Coordinator:         readyok
Coordinator->testengine-c3127893-0:         position startpos
Coordinator->testengine-c3127893-0:         isready AI 0
Coordinator->testengine-c3127893-0:         perft 1
testengine-c3127893-0->Coordinator:         7
```

Submission

Your submission should consist of the code, and a short report of no more than two pages stating the main aspects of your engine, including the type of search you are using (for example minimax), how your evaluation function has appropriate values for the win, loss, and draw results, and any other aspects of your engine you find interesting. Comment your code sufficiently. Only submit the `.java` files, the report, and a readme if you think it is necessary, all in a zip file with the naming convection `surname_firstname_studentnumber`. The java file with the main function must be called `Interface`, it is the only file which should do IO with the coordinator.

Bonus Marks Tournament

In week 13 we will have a tournament of engines that are made by whoever shows up to the lecture. Please indicate before week 13 if you wish to participate in it so I can prepare appropriately for it. The prize for winning will be 3 bonus marks. Second and third place will get 2 and 1 bonus marks respectively. The structure of the tournament will be dependent on the number of participants, and as I get a feel for how many people will be there I will decide the structure of the tournament. There will be time restrictions for each player.

If it is a small number of participants, it will be a round robin tournament, where every engine will play every other engine in a best of two match.

If there is a medium amount of participants, we will play multiple groups, with the winners of the groups proceeding to a group of finalists.

If there is a large amount of participants, we will play multiple groups, which will be used as the seeding for a double elimination tournament.

Marking Criteria

1. 30 marks for correctly implementing `minimax` or a derivative of `minimax`, including an evaluation function that works as intended.
2. 10 marks for correctly implementing the `perft` function.
3. 10 marks for correctly passing strings back and forth between the engine and the coordinator.
4. 10 marks for clear comments.

5. 10 marks for a clear report. A description of the evaluation function should be included in the report.
6. 5 marks for correct file and engine naming conventions.
7. -5 marks for each time the marker must modify the submission to make it work. The first error message on compile will be addressed each time. This will only be done to address compile problems, not run time problems. Modifications will only be performed if they are extremely easy and quick to do so.