FoxGame info

The agents will send moves to each other by communicating through an intermediary server program called *the monitor*. The monitor allows us to watch the game in progress.

Timeslice

The monitor also imposes a time limit on the players. That is, they have some amount of time during which they are allowed to plan the next move. An agent is allowed to submit a move early. In that case, the extra time is discarded, and the opponent's turn starts earlier. Exceeding the time slice is illegal.

An agent is (of course) allowed to plan its next move while it is waiting for the opponent to move, but don't forget that the opponent might finish early.

Components

On the course page you will find some files for download:

- Agent Framework
- Foxgame Helper
- Agent Performance Tests

monitor.jar

This is a GUI program that acts as a server. Agents can connect to it to play each other, and it will display the board as well as some extra information about the game and the players. The monitor will listen for connections on port 6000.

The jar takes no arguments. So java -jar monitor.jar works fine, or you can just double-click it. Create a new game by clicking the "New Game" button. Now the server will wait for players (agents) to connect. Once two players are connected, you will be able to click the "Start Game" button. A new game can only be started once the current game has ended. It will end automatically when one side wins. Alternatively, the game can be canceled at any time using the "Cancel Game" button.

Time Slice

Agents have only a limited amount of time for making moves. This limit is imposed by the monitor and can be set by entering a new value inside the text field above the "Update Timeslice" button and clicking the button. This can be done at any time. When updated during the game, the new time slice will take effect in the following turn.

The format is very flexible. A time unit is optional, and supported units are ms for milliseconds, s for seconds, min for minutes and h for hours. Units are case-insensitive. If no unit is entered, it will be inferred, with seconds being the default. "3" would mean 3 seconds. "0.5" would be the same as entering "500 ms". Hours, minutes and seconds can be separated with colons (:). So "5:30" is interpreted as 5 minutes and 30 seconds. "1:05:30" is interpreted as 1 hour, 5 minutes, and 30 seconds.

agent_framework.zip

This is the project you will base your agent on. It can be imported as an eclipse project. See implementing the agent below for more details.

CLI Arguments

These are the arguments that the `Main` class in the agent framework assumes.

- -a Host name or IP
- -p Port number
- -r Role
- -n Agent name

The role is case insensitive and can be abbreviated. So `-r f` and `-r Sh` will be interpreted as FOX and SHEEP respectively.

`Main` will default to sensible values for missing arguments. It will try to connect to localhost:6000 if no -a or -p argument is given. If no -r (role) argument is given, it will use whatever role is assigned by the monitor.

Implementing the Agent

You only have to complete the `FoxGameEngine` class. This means filling out the `getMove` and `updateState` methods.

Note that `getMove` is only supposed to generate a move. So you don't have to remember moves you generate. The AI will explicitly be asked to update its state with both the opponent's and the agent's own moves.

Your `getMove` method is going to be interested in `GameStatus.timeSlice`. That's the time (in milliseconds) it may spend generating a move. Remember to allow for some small margin so that there's enough time to actually return the move.

The Foxgame Helper

The `Foxgame` class is the only class in the jar that is interesting. It represents the game by implementing the `ai.games.Game` from LO3. So you can choose the best move just like you did in the exercise. The only difference is that you now also need to handle the timeslice. The important point is that you don't have to develop your own game state representation or implement the rules so you can figure out what the possible moves are. That's all taken care of.

Agent Performance Tests

This is a tool that allows you to test the performance of your agent. See the `ExampleTest.java` for more information on how to use it.

This is also the tool that will be used to set a minimum performance requirement. You will get more information about what those requirements are shortly.

Report

When you submit your agent, you must also submit a report describing that agent.

The report should be no longer than one A4 page (approximately 400 words). Include the following sections:

- General description
- Utility function
- Novelty
- Shortcomings

Under *general description*, you will describe the algorithm your agent uses to choose moves. Under *utility function*, you explain how your agent assesses the desirability of different states (features). Use the *novelty*section to highlight any clever tweaks and optimizations your agent is using. Lastly, the *shortcomings* section allows you to discuss any issues you discovered, simplifications you had to make, or things you might have done differently if you were to start over.

Submission

You will have to submit your agent as an executable .jar named `agent_aiYY_ID.jar`. For example: `agent_ai18_crfi0400.jar`

Submit everything (class files, source files, and report) in one file. Ideally, you can put everything directly in the jar, or you can package the jar + other files in a zip or tar(.gz|.bz2).

Don't forget to include the source code and the report!

Once you have submitted your agent, I will check whether it passes the performance requirements. I will let it play vs level 1 repeatedly, with a few different seeds and timeslices, and check that it wins at least 85% of the games. In other words, **in order to pass, your agent has to beat level 1 at least 85% of the time, both as fox and as sheep**. Your agent should work with almost any timeslice, but you can expect it to be tested for values from 100 ms and up.

The Tournament

During the tournament, the two competing agents will run in their own virtual machine. You are expected to be able to answer questions about your agent and comment on its behavior during the game. You will be awarded points based on the outcome of the game. WIN gives 3 points, LOSS 0 points. WIN is defined as follows, and in that order

- contender wins as sheep and fox.
- contender wins as fox and loses as sheep, losing fewer foxes than the opponent in the winning game.
- contender wins as fox and loses as sheep, requiring fewer moves in the winning game.
- contender wins as sheep and loses as fox but lost fewer sheep in the winning game than the opponent.
- contender wins as sheep and loses as fox but required fewer moves to end the winning game.
- otherwise DRAW -> 1 point each (highly unlikely).