Programming Project

Updated Details

CS4725/CS6705

Fall 2017

Introduction

- The programming project will be completed (in Java) alone or in teams of 2-3 people.

- Goal of the project:
  - to use AI techniques learned in class, and perhaps on your own, to create an agent to play a 2-player game as intelligently as possible

Introduction

- You will be provided with:
  - All of the Java code needed for your client program to communicate with the game server
  - The game server code, so that you can test your player against simple opponents, test your player against itself, or test different versions of your player against each other
  - At least one very simple player implementation

The game: Quarto

- Quarto is a commercially available board game, by a company in France called *Gigamic*.
- The game you will be working with is a *variation* on Quarto, using a 5x5 board instead of 4x4.

Rules of the game

- Our version of the game is played on a 5x5 square board.
- There are 32 different pieces that can be played, each with a unique combination of five characteristics:
  - Tall or short
  - Solid or hollow
  - White or black
  - Wood or metal
  - Round or square

Quarto game pieces

- These are pieces from the actual Quarto game (which uses only 4 characteristics instead of 5)

Object of the game

- The object of the game is to be the player who places a piece that creates a line of five pieces (in a row, column or diagonal) that share at least one characteristic:
  - e.g., five square pieces in the same row or five tall pieces in the same column

Game play

- An interesting twist to this game:
  - When it is Player 1's turn to play a piece, it is **Player 2** who decides ***which*** piece Player 1 gets to place on the board.
  - Therefore, the game sequence is:
    - P2 chooses a piece.
    - P1 chooses where to play that piece.
    - P1 choose a piece.
    - P2 chooses where to play that piece.
    - P2 chooses a piece.
    - etc.

Ending the game

- The game is won when a player places a piece that creates a line of five pieces that share at least one characteristic. (It does not matter who placed the other pieces in the line; the winner is the player who placed the final piece.)
- If all 25 squares are filled, with no line of pieces that share a characteristic, then the game is a draw (tie).

Implementation details

- In our implementation, there is no graphical user interface, so everything will be text-based.
- Each piece is represented by a five-bit binary number:

  e.g., 11111 = tall, solid, white, wood, round
  00000 = short, hollow, black, metal, square
  01110 = short, solid, white, wood, square

- The game is won if five pieces in a line share the same binary digit in the same location:

  e.g., 0**1**000, 1**1**101, 0**1**010, 1**1**110, 0**1**101

Communicating game moves

- During the game, there are two types of messages that your client will send to the server:
  - which piece to give to the opponent to play: e.g., “10011”
  - the location at which to play the piece that has been given to you: e.g., “3,1”
- You will have to write the necessary code to select pieces and to choose moves (within a time limit).

Files provided

- Several files are provided to you. Take the time to look through them, to understand how the code works.

- For your final project submission, you will be required to submit a file called QuartoPlayerAgent.java containing your implementation.

- Assignment 1 includes an exercise in which you will add to the implementation of the simple player in QuartoSemiRandomAgent.java

List of provided Java files

- GameClient.java
- GameServer.java
- QuartoAgent.java
- QuartoServer.java
- QuartoBoard.java
- QuartoPiece.java
- QuartoRandomAgent.java
- QuartoSemiRandomAgent.java *(incomplete)*

Simple agents

- The two provided example agents work as follows.
  - QuartoRandomAgent.java:
    - Returns a random (unplayed) piece for the opponent to play
    - Returns a random (unoccupied) square in which to play a piece
  - QuartoSemiRandomAgent.java *(incomplete):*
    - Returns the first (unplayed) piece it finds that will **not** allow the opponent to win on the next move; if no such piece exists, returns a random piece
    - *[to be completed by you for Assignment 1]* Returns the first (unoccupied) square it finds that leads immediately to a win; if no such square exists, returns a random square

How to obtain and run the code

- In one of the Faculty of Computer Science labs, create a directory for your project and move to that directory.

  ```
  mkdir quarto
  cd quarto
  ```

- Copy all provided files into your directory, either by downloading them from D2L or by using the following command. (Be sure to include the ‘ . ’ at the end.)

  ```
  cp /fcs/courses/cs4725/F17/quarto/*  .
  ```

- Use the java compiler to compile the code.

  ```
  javac *java
  ```

How to obtain and run the code

- Start the server.

  java QuartoServer

- Note that there is an option to start the server with a specific starting board position (other than an empty board). One sample starting board is provided in the file state.quarto

  java QuartoServer state.quarto

- You can create your own files with different starting board positions, using the same format as state.quarto

How to obtain and run the code

- In separate terminal windows, you can run two copies of the random client program and observe the results of them playing against each other. For example:

  [in one window] java QuartoRandomAgent localhost

  [in another window] java QuartoRandomAgent localhost

- If you wish to start the game with a specific board position, then this must be done for the server (see previous slide) and for each client.

  [in one window] java QuartoRandomAgent localhost state.quarto

  [in another window] java QuartoRandomAgent localhost state.quarto

How to obtain and run the code

- After completing the implementation of the QuartoSemiRandomAgent, you can try running it against the QuartoRandomAgent, to ensure that it wins most of the time.

```
[in one window]          java QuartoSemiRandomAgent localhost
[in another window]    java QuartoRandomAgent localhost
```

Time limit

- You should ensure that your agent returns moves within a specified time limit.
- The value of this time limit is sent by the server to the clients at the beginning of the game and stored in the variable timeLimitForResponse
- If the server does not receive a move from your agent within the time limit, or if it receives an invalid response, it will just choose a random move for you.
- You can experiment with setting different time limits in the server code, but I will provide you before the end of the term with a specific time limit that we will use.

Time limit (continued)

- There is some sample code in QuartoSemiRandomAgent.java showing how you can check whether you are getting close to the time limit and therefore must return a move.

Messages from server

- Also in QuartoSemiRandomAgent.java, there is code (currently commented out) to check whether any messages have arrived from the server.
- This would be useful, for example, if the server had sent a message indicating that it had already chosen a move on your behalf (because of the time limit expiring).
- Your agent would then know not to bother sending a move for that turn.

Possible ideas for agent strategies

- Once we have covered Chapter 5 in class, we will talk about some possible strategies that you might use for your QuartoPlayerAgent.

Documentation to hand in

- In addition to the code for your player, you will be required to submit a short document describing the approach that you have used.
- This document does not have to be long (1 to 3 pages) and should just describe at a high level how your player works.

How submissions will be evaluated

- Performance against simple opponents
- Performance against classmates' programs
- Ability to win or avoid losing with certain starting board positions
- My subjective opinion of the amount of work you have done on your project, based on the description in your document and my inspection of your code