COMP3411/9414 Artificial Intelligence

Term 1, 2019

Project 3: Nine-Board Tic-Tac-Toe

Due: Wednesday 1 May, 11:59 pm
Marks: 16% of final assessment

Introduction

In this project you will be writing an agent to play the game of Nine-Board Tic-Tac-Toe.

This game is played on a 3 x 3 array of 3 x 3 Tic-Tac-Toe boards. The first move is made by placing an x in a randomly chosen cell of a randomly chosen board. After that, the two players take turns placing an o or x alternately into an empty cell of the board corresponding to the cell of the previous move. (For example, if the previous move was into the upper right corner of a board, the next move must be made into the upper right board.)

The game is won by getting three-in-a row either horizontally, vertically or diagonally in one of the nine boards. If a player is unable to make their move (because the relevant board is already full) the game ends in a draw.

Getting Started

Copy the archive src.zip into your own filespace and unzip it. Then type

```
cd src
make all

./servt -x -o
```

You should then see something like this:

```
 . . . | . . . | . . .
 . . . | . . . | . . .
 . . . | . . . | . . .
------+-------+------
 . . . | . . . | . . .
 . . . | . . . | . . .
 . . . | . . x | . . .
------+-------+------
 . . . | . . . | . . .
 . . . | . . . | . . .
 . . . | . . . | . . .

next move for O ?
```

You can now play Nine-Board Tic-Tac-Toe against yourself, by typing a number for each move. The cells in each board are numbered 1, 2, 3, 4, 5, 6, 7, 8, 9 as follows:

```
+-----+
|1 2 3|
|4 5 6|
|7 8 9|
+-----+
```

To play against a computer player, you need to open another terminal window (and `cd` to the `src` directory).

Type this into the first window:

```
./servt -p 12345 -x
```

This tells the server to use port `12345` for communication, and that the moves for `x` will be chosen by you, the human, typing at the keyboard. (If port `12345` is busy, choose another 5-digit number.)

You should then type this into the second window (using the same port number):

```
./randt -p 12345
```

The program `randt` simply chooses each move randomly among the available legal moves.

The Prolog program `random.pl` behaves in exactly the same way. You can play against it by typing this into the second window:

```
prolog 12345 < agent.wrap
```

You can play against a slightly more sophisticated player by typing this into the second window:

```
./lookt -p 12345
```

(If you are using a Mac, type `./lookt.mac` instead of `./lookt`)

To play two computer programs against each other, you may need to open three windows. For example, to play `agent` against `lookt` using port `54321`, type as follows:

window 1: `./servt -p 54321`
window 2: `./agent -p 54321`
window 3: `./lookt -p 54321`

(Whichever program connects first will play x; the other program will play o.)
Alternatively, you can launch all three programs from a single window by typing

```
./servt -p 54321 &
./agent -p 54321 &
./lookt -p 54321
```

or, using a shell script:

```
./playc.sh lookt 54321
```

To play the prolog program `agent.pl` against `lookt` using port `23232`, you can type

```
./servt -p 23232 &
prolog 23232 < agent.wrap &
./lookt -p 23232
```

or, using a shell script:

```
./playpl.sh lookt 23232
```

(If you are using a Mac, edit `playpl.sh` and replace `"prolog"` with `"swipl"`)

The strength of `lookt` can be adjusted by specifying a maximum search depth (default value is 9; reasonable range is 1 to 18), e.g.

```
./lookt -p 12345 -d 6
```

or

```
./playc.sh "lookt -d 16" 54321
```

Writing a Player

Your task is to write a program to play the game of nine-board tic-tac-toe as well as you can.

Your program will receive commands from the server (`init`, `start()`, `second_move()`, `third_move()`, `last_move()`, `win()`, `loss()`, `draw()`, `end()`) and must send back a single digit specifying the chosen move.
(the parameters for these commands are explained in the comments of `agent.pl`)

Communication between the server and the player(s) is illustrated in this brief example:

Player X	**Server**	**Player O**
	← init	
		init →
	← start(x)	
		start(o) →
	second_move(6,1) →	
		← 7
	← third_move(6,1,7)	
9 →		
	next_move(9) →	
		← 6
	← next_move(6)	
5 →		
		last_move(5) →
	← win(triple)	
		loss(triple) →
	← end	
		end →

Language Options

You are free to write your player in any language you wish.

1. If you write in Prolog, your program will be invoked like this:

   ```
   prolog (port) < agent.wrap
   ```

   You should submit your `.pl` files (including `agent.pl`). Feel free to use `agent.pl` (identical to `randt.pl`) as a starting point, as well as `alphabeta.pl` (which implements alpha-beta search for regular Tic-Tac-Toe).

2. If you write in Java, your program will be invoked by

   ```
   java Agent -p (port)
   ```

You should submit your `.java` files (no `.class` files). The main file must be called `Agent.java`

3. If you write in Python, your program will be invoked by

   ```
   ./agent.py -p (port)
   ```

   You should submit your `.py` files (including `agent.py`). The first line of your code must specify which version of Prolog you are using, e.g.

   ```
   #!/usr/bin/python
   ```

4. If you write in C or C++, your program will be invoked by:

   ```
   ./agent -p (port)
   ```

   You should submit your source files (no object files) as well as a `Makefile` which, when invoked with the command "`make`", will produce an executable called `agent` . Feel free to use the supplied files as a starting point (especially `agent.c` which is identical to `randt.c`)

If you wish to write in some other language, let me know.

Question

At the top of your code, in a block of comments, you must provide a brief answer (one or two paragraphs) to this Question:

> Briefly describe how your program works, including any algorithms and data structures employed, and explain any design decisions you made along the way.

Groups

This assignment may be done individually, or in groups of two students. Groups are determined by an SMS field called `hw3group`. Every student has initially been assigned a unique `hw3group` which is `"h"` followed by their studentID number, e.g. `h1234567`.

1. If you plan to complete the assignment individually, you don't need to do anything (but, if you do create a group with only you as a member, that's ok too).
2. If both members of the group are enrolled in COMP3411, you should go to this WebCMS page and click on "Groups" in the left hand column, then click "Create". Click on the menu for "Group Type" and select "hw3". After creating a group, click "Edit", search for the other member, and click "Add". WebCMS assigns a unique group ID to each group, in the form of `"g"` followed by six digits (e.g. `g012345`). We will periodically run a script to load these values into SMS.
3. If both members of the group are enrolled in COMP9414, go instead to this WebCMS page and follow the same instructions as above.
4. If one group member is enrolled in COMP3411 and the other in COMP9414, please send email to `blair@cse.unsw.edu.au` stating the name and student number of the two group members.

Submission

COMP3411 students should submit by typing

```
give cs3411 hw3 ...
```

COMP9414 students should submit by typing

```
give cs9414 hw3 ...
```

You can submit as many times as you like - later submissions will overwrite earlier ones. You can check that your submission has been received by using the following command:

```
3411 classrun -check
9414 classrun -check
```

The submission deadline is Wednesday 1 May, 11:59 pm.
15% penalty will be applied to the (maximum) mark for every 24 hours late after the deadline.

Additional information may be found in the FAQ and will be considered as part of the specification for the project.

Questions relating to the project can be posted to the Forums on the course Web site. If you have a question that has not already been answered on the Forum, you can email it to `blair@cse.unsw.edu.au`

Marking scheme

- 10 marks for performance against a number of pre-defined opponents.
- 6 marks for Algorithms, Style, Comments and answer to the Question

You should always adhere to good coding practices and style. In general, a program that attempts a substantial part of the job but does that part correctly will receive more marks than one attempting to do the entire job but with many errors.

Plagiarism Policy

DO NOT COPY CODE FROM THE INTERNET. This approach has a very specific structure.
Copying/adapting code is likely to take much longer than understanding the logic behind the provided file. It's also plagiarism.

Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for any similar projects from previous years) and serious penalties will be applied, particularly in the case of repeat offences.

DO NOT COPY FROM OTHERS; DO NOT ALLOW ANYONE TO SEE YOUR CODE

Please refer to the UNSW Policy on Academic Honesty and Plagiarism if you require further clarification on this matter.

Good luck!
