COMP 3411 Artificial Intelligence
Session 1, 2017

Project 3, Option 1: Treasure Hunt

Due: Sunday 28 May, 11:59 pm
Marks: 18% of final assessment

For this project you will be implementing an agent to play a simple text-based adventure game. The agent is considered to be stranded on a small group of islands, with a few trees and the ruins of some ancient buildings. It is required to move around a rectangular environment, collecting tools and avoiding (or removing) obstacles along the way.

The obstacles and tools within the environment are represented as follows:

Obstacles		Tools	
`T`	tree	`a`	axe
`-`	door	`k`	key
`*`	wall	`d`	dynamite
`~`	water	`$`	treasure

The agent will be represented by one of the characters `^`, `v`, `<` or `>`, depending on which direction it is pointing. The agent is capable of the following instructions:

`L` turn left
`R` turn right
`F` (try to) move forward
`C` (try to) chop down a tree, using an axe
`B` (try to) blast a wall or tree, using dynamite

When it executes an `L` or `R` instruction, the agent remains in the same location and only its direction changes. When it executes an `F` instruction, the agent attempts to move a single step in whichever direction it is pointing. The `F` instruction will fail (have no effect) if there is a wall or tree directly in front of the agent.

When the agent moves to a location occupied by a tool, it automatically picks up the tool. The agent may use a `C`, `U` or `B` instruction to remove an obstacle immediately in front of it, if it is carrying the appropriate tool. A tree may be removed with a `C` (chop) instruction, if an axe is held. A door may be removed with a `U` (unlock) instruction, if a key is held. A wall, tree or door may be removed with a `B` (blast) instruction, if dynamite is held.

Whenever a tree is chopped, the tree automatically becomes a raft which the agent can use as a tool to move across the water. If the agent is not holding a raft and moves forward into the water, it will drown. If the agent is holding a raft, it can safely move forward into the water, and continue to move around on the water, using the raft. When the agent steps back onto the

land, the raft it was using will sink and cannot be used again. The agent will need to chop down another tree in order to get a new raft.

If the agent attempts to move off the edge of the environment, it dies.

To win the game, the agent must pick up the treasure and then return to its initial location.

Running as a Single Process

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

```
cd src
javac *.java
java Raft -i s0.in
```

You should then see something like this:

```
~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~
~~  d   *     T   a  ~~
~~    *-*     ***    ~~
~~****     v     ****~~
~~TTT**         **TTT~~
~~  $ **   k   **    ~~
~~     **     **     ~~
~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~

Enter Action(s):
```


This allows you to play the role of the agent by typing commands at the keyboard (followed by <Enter>). Note:

- a key can be used to unlock any door; once it is unlocked, it has effectively been removed from the environment and can never be "closed" again.
- an axe or key can be used multiple times, but each dynamite can be used only once.
- the agent can hold multiple dynamites simultaneously, but it can only hold one raft at a time.
- `C`, `U` or `B` instructions will fail (have no effect) if the appropriate tool is not held, or if the location immediately in front of the agent does not contain an appropriate obstacle.

Running in Network Mode

Follow these instructions to see how the game runs in network mode:

1. open two windows, and `cd` to the `src` directory in both of them.
2. choose a port number between 1025 and 65535 - let's suppose you choose 31415.
3. type this in one window:

   ```
   java Raft -p 31415 -i s0.in
   ```

4. type this in the other window:

```
java Agent -p 31415
```

In network mode, the agent runs as a separate process and communicates with the game engine through a TCPIP socket. Notice that the agent cannot see the whole environment, but only a 5-by-5 "window" around its current location, appropriately rotated. From the agent's point of view, locations off the edge of the environment appear as a dot.

We have also provided a C version of the agent, which you can run by typing

```
make
./agent -p 31415
```

Writing an Agent

At each time step, the environment will send a series of 24 characters to the agent, constituting a scan of the 5-by-5 window it is currently seeing; the agent must send back a single character to indicate the action it has chosen.

You are free to write the agent in any language you choose. If you are writing in Java, your main file should be called `Agent.java` (you are free to use the supplied file `Agent.java` as a starting point). If you are writing in C, you are free to use the files `agent.c`, `pipe.c` and `pipe.h` as a starting point. In other languages, you will have to write the socket code for yourself. You must include a `Makefile` with your submission, producing an executable called `agent`.

You may assume that the specified environment is no larger than 80 by 80, but the agent can begin anywhere inside it.

Additional examples of input and output files will be provided in the directory `hw3raft/sample`.

There is a widget on the course Web site which allows you to edit your own input maps, test them using a graphical interface. and share them with others.

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.

Submission

COMP3411 students should submit by typing

```
give cs3411 hw3raft Makefile ...
```

COMP9414/9814 students should submit by typing

```
give cs9414 hw3raft Makefile ...
```

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 one of these commands:

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

The submission deadline is Sunday 28 May, 11:59 pm.
15% penalty will be applied to the (maximum) mark for every 24 hours late after the deadline.

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`

Please ensure that you submit the source files and NOT any binary files. The `give` system will compile your program using your `Makefile` and check that it produces a binary file (or java class files) with the correct name.

Assessment

Your program will be tested on a series of sample inputs with successively more challenging environments. There will be:

- 12 marks for functionality (automarking)
- 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

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!
