Maze

Core project

Notions: Path-finding algorithms and Algorithm design

Introduction

You are back in the game, young recruit!

This first project will ease you back into coding (if you ever stopped) through an algorithmic challenge. The goal is for you to code a program (an "Intelligent Agent") capable of gathering resources in a given area.

We provide you with a program (a "virtual machine") that defines an area where resources pop randomly, and your IA must move around to get those resources. Sounds simple enough, no?

Your mission

You will code an IA agent that interacts with the maze. The goal of your IA is to move around to gather the resources in the maze, without colluding into the walls represented by `#` (if it does, your IA dies!). Every time you gather a resource, you win a point: your goal is therefore to get the highest score possible.

There are two types of resources you can gather:

- `o` are permanent and worth 1 point
- `!` are ephemeral -they only last 20 turns before disappearing- and are worth 2 points

Your IA must be in a file named `maze_ia.py`, at the root of your git repository.

Note: your Python file must start with a shebang specifying Python as the interpreter. See the example below showing the first line of the IA.

```
$ head -1 maze_ia.py
#!/usr/bin/env python3

$ ./maze /path/to/maze_ia.py
####################
#                  #
#                  #
#                  #
#   o          o   #
#                 A#
#                o #
#                  #
```

You will find the binary of the maze below. The executable takes one or several IAs (up to 26) as arguments.

You can specify different options to the maze, so that you can test your IA in different conditions. The `kind` option allows you to choose the mode.

For the core part of the project, your IA must be able to gather resources without ever dying in all modes except the maze mode.

The Sentinel will only test your IA alone, not with other IAs.

```
Usage: maze [options] IA1 [IA2, ..., IA26]
    -s SIZE, --size=SIZE              Specifies the size of the map (must be in s, m, l,
    -c NBR_COINS, --coins=NBR_COINS   Specifies the number of coins (> 0) default 3
    -l NBR_LOOPS, --loop=NBR_LOOPS    Specifies the number of loops (> 10) default 1000
    -z SLEEP_MS, --sleep=SLEEP_MS     Specifies the time the program will sleep between
    -k KIND, --kind=KIND              Specifies the kind of the shape of the map (must b
    -d, --debug                       Add some debug text
    -h, --help                        Show this help
    --show                            Display the game


```

maze

The communication protocol

Your IA will communicate with the virtual machine through stdin, stdout and stderr:

- It will read the output of the VM on stdin (you need to parse that output to know the state of the maze)
- It will send its next move to the VM on stdout
- You can use stderr to print debugging messages (each IA gets its own color to differentiate between different IAs' debug messages)

The VM loads the IAs in the order they are given on the command-line, and attributes a letter to them. The IA with the letter A (the first one on the command line) will play first, then the IA with the letter B (the second one on the command line) will play, etc.

Here's the communication protocol:

- On loading the IA, the VM will send "HELLO\n\n" → The IA answers with "I AM <name>\n\n" (with its own name in place of <name>)
- The VM then answers with the letter attributed to the IA: "YOU ARE <letter>\n\n" → The IA answers "OK\n\n"
- The VM puts the IA inside the maze and returns "MAZE\n" + the maze to be parsed followed by "\n\n" → From there the IA can send its move with "MOVE [UP/DOWN/RIGHT/LEFT]\n\n"

As you can see all communication messages end with "\n\n".

There's a 2 seconds timeout for IAs. If your IA fails to send its move to the VM within 2 seconds, it will die. Your IA will also die if you send an invalid message to the VM.

1. You should first focus on implementing the protocol properly, with an IA that does nothing useful.

2. Once you can establish the communication with the VM, you need to figure out how to parse & store the maze it sends you.

3. When you have your own copy of the maze to work on, you can then think of gathering the resources and sending back your (smart!) move to the maze.

Bonus: maze mode →