Programming and Algorithmic Thinking Assignment 2023

Dr. Paris Mavromoustakos Blom

Last updated: July 10, 2023

Contents

1 Assignment description and goals

In this assignment, your goal is to program a robot vacuum cleaner in order to clean up “stains” from the floor of a room as efficiently as possible. As a measure of efficiency, we use the number of moves that the robot makes in order to clean up all the stains. The fewer moves required, the more efficient the robot’s algorithm.

The robot moves horizontally and/or vertically, one cell at a time, in a square grid (also referred to as a ”map”). An example is provided in Figure 1. The robot can not move outside the map limits (think of it as the room’s walls). Therefore, in Figure 1, the robot can only move right or down from its current position.

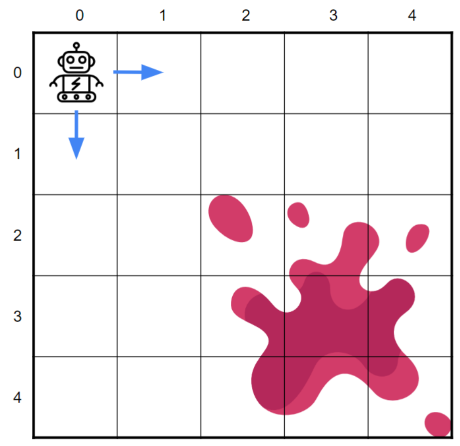


Figure 1: Illustration of the robot vacuum cleaner in a grid-like map. The robot’s current position is (0,0) and it can only move right to (1,0) or down to (0,1).

Whenever the robot moves to a map position that contains a stain, that grid cell is “cleaned”; it will not contain a stain anymore. In order to “solve” a map, the robot vacuum has to visit (and therefore clean) all the stained grid cells.

Each move the robot makes, horizontally or vertically, costs 1 point of energy. If the robot runs out of energy, the game is over; if there are still stains left, the robot has failed to clean up the room. The robot always starts with $2 \times nr\_cols \times nr\_rows$ energy points, at the top-left corner of the map.

The robot does not “know” the layout of the map; from its current position, it can merely “see” the 8 surrounding cells through its sensors. An illustration is provided in Figure 2. More information about the scoring, rules and attributes of the assignment follows in Section 2.

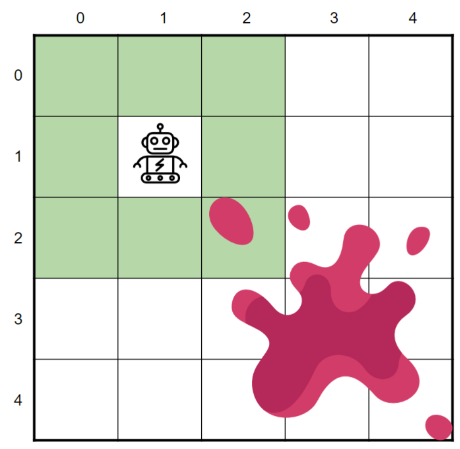


Figure 2: Illustration of the robot vacuum cleaner's range of "vision". It can only see the 8 surrounding grid cells through its sensors.

2 Scoring, rules and attributes

2.1 Scoring

As mentioned above, the robot vacuum cleaner always starts at the top-left corner of the map, with $total\_energy = 2 \times nr\_cols \times nr\_rows$ energy points. For example, on a 30×30 map, the total energy will be $2 \times 30 \times 30 = 1800$. The game ends when either of the following occurs: a) the robot visits the last remaining stained grid cell, or b) the robot reaches 0 energy. In the latter case, the final score is 0; in the former case, the score is equal to $total\_energy - number\_steps\_taken$. If, for example, the robot required 250 moves to clean up the entire map, the final score is $1800 - 250 = \mathbf{1550}$.

2.2 Map layout

Figures 1 and 2 depict a robot vacuum cleaner in a simple, 5×5 map. The map coordinates are represented by two integers in the format of `map[X][Y]` where X represents the row and Y represents the column of the map grid that the robot is in. For each of the coordinates, counting starts at 0. Programmatically, the map is represented by a $N \times N$ matrix; a list of length N, where each of the N elements is another list of length N. For example, looking at Figure 2, the robot's current position is `map[1][1]`, where map is a list of length 5, and each element is another list of length 5 (`map = [[x,x,x,x,x], [x,x,x,x,x], [x,x,x,x,x], [x,x,x,x,x], [x,x,x,x,x]]`). For now, assume that each 'x' represents a position in the map.

On initialisation, the game engine reads the map from a csv file. That csv file contains a row for each corresponding row of a map's grid. An example is illustrated in Figure 3. In a map csv file, the following encoding is used:

- ‘x’ represents walls (map limits and obstacles)
- ‘.’ represents “clean” floor (not stained)
- ‘@’ represents a stain
- ‘#’ represents the starting square. The starting square’s coordinates are (1,1) since the first row and column are occupied by room walls.

The map is always surrounded by walls, but when there are obstacles included in the map, the walls may have outcroppings. The robot sensors interpret the starting square as a clean floor.

Important to notice:

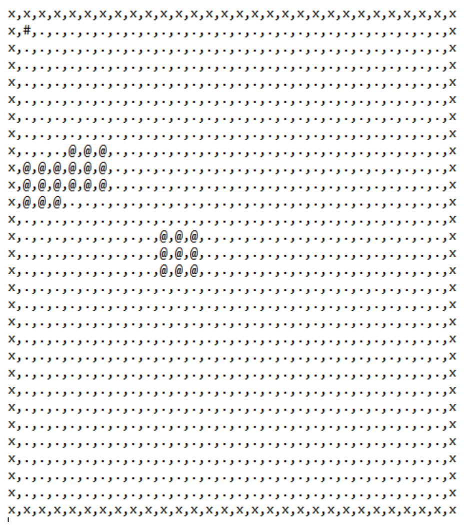
- The map dimensions (nrRows, nrCols) may differ per map. For the sake of simplicity, the maps will always be square (NxN) - even though that should not have an impact on your bots, if you don’t manually pre-assign the dimensions. That being said, remove any constants from your implementations (e.g. replace 30 with self.nrRows/self.nrCols) as it will deem your bot incapable of traversing differently sized maps.
- The checkpoint/starting point will always be the top-left corner of the map (1,1)
- The total energy will be 2*nrRows*nrCols. So in a 30x30 map your bots will start with 1800 energy. I believe this amount to be enough to solve any map, with a considerably efficient algorithm.
- Your bots will be allowed to run for a maximum of 2 minutes per map (with LATENCY=0 and no visuals). If your algorithm exceeds 2 minutes in runtime, it will be automatically terminated and will be scored with a 0 for that map. This step is necessary to avoid infinite loops which may occur when your bot is ”thinking”.

A sample map is provided in the project file, see “map1.csv” (suggestion: open this file with a simple text editor).

2.3 Walls, stains and obstacles

The map’s grid cells (except for the starting square, (1,1)) can contain either (clean) floor, walls or stains. There are some rules that govern the layout of stains, walls and obstacles. These rules can be found in `app.py` and are also listed below:

- Maps are always square (the height and width are equal). See `nrRows` and `nrCols` in the `settings` dictionary (`app.py`).
- The number and size of stains is pre-defined. Stains are always square. See `nrStains` and `sizeStains` in the `settings` dictionary (`app.py`).

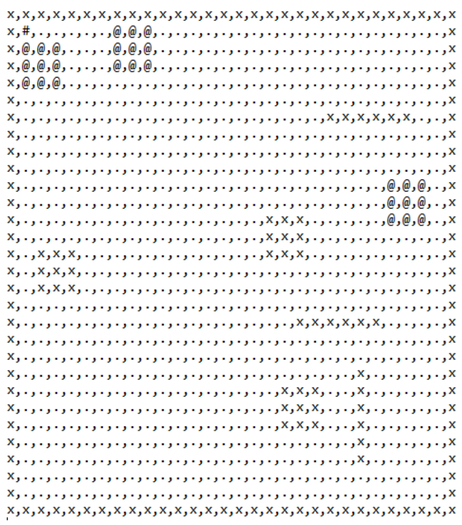


```
x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x
x,#,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,@,@,@,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,@,@,@,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x
```

Figure 3: An example of a csv file that represents a 30×30 map.

- Pillars are convex obstacles of pre-defined number and size. Pillars are always square. See `nrPillars` and `sizePillars` in the `settings` dictionary (`app.py`).
- The map can contain additional obstacles in the form of straight walls (vertical or horizontal). The length and number of such walls is pre-defined. See `nrWalls` and `sizeWalls` in the `settings` dictionary (`app.py`).
- Stain, pillar and wall sizes will be (individually) constant per map, meaning that a single map cannot contain two differently sized stains, two differently sized pillars or two differently sized walls. However, the stain and pillar sizes might differ.
- "Non-labyrinth" maps (8 & 9 grade maps) can contain an arbitrary number of pillars and/or wall obstacles. These obstacles will not "touch" each other (be placed next to each other) to create more complicated, convex obstacles. Obstacles are allowed to touch the outer walls of the map. Obstacles are allowed touch each other only in difficulty (grade) 10 maps, to create "labyrinths".
- Stains can be placed next to each other in all maps.

An illustration of a map containing pillars and additional walls is provided in Figure 4.



```
x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x
x,#,.,.,.,.,.,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,@,@,@,.,.,.,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,@,@,@,.,.,.,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,@,@,@,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,x,x,x,x,x,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,@,@,@,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,@,@,@,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,x,x,.,.,.,.,.,@,@,@,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,x,x,.,.,.,.,.,.,.,.,.,x
x,.,x,x,x,.,.,.,.,.,.,.,.,.,.,.,.,x,x,x,.,.,.,.,.,.,.,.,.,x
x,.,x,x,x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,x,x,x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,x,x,x,x,x,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,x,x,.,.,x,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,x,x,.,.,x,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,x,x,.,.,x,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x
```

Figure 4: An example of a csv file that represents a 30×30 map, with three stains, three pillars and three additional wall obstacles.

3 Running the game on jupyter

To run the game, download "project.zip" from canvas. Unzip the file and upload the folder contents in a new folder on your jupyter notebook. After having created the folder and uploaded all the files from the zipped folder into the (empty) jupyter folder, open a new terminal window and type: `cd [your_folder_name]`, press Enter; then type `python app.py` and press Enter again. An illustration follows in Figure 5.

4 Code outline

This game engine is written using Python 3, without the use of external libraries such as pandas or numpy. The module `csv` is used to load maps into the game. The engine uses an object-oriented architecture, which you are not familiar with. However, you will not require to learn object-orientation in python to be able to complete the assignment.

The game is divided into several files:

- `Bot.py` implements the Bot class, which represents the robot vacuum cleaner. Bot implements a method called `nextMove`; this method receives specific information from the game engine, decides where to move next, and returns that decision. You will not need to (and should not) modify this file.
- `Game.py` implements the Game class, which represents the game engine.

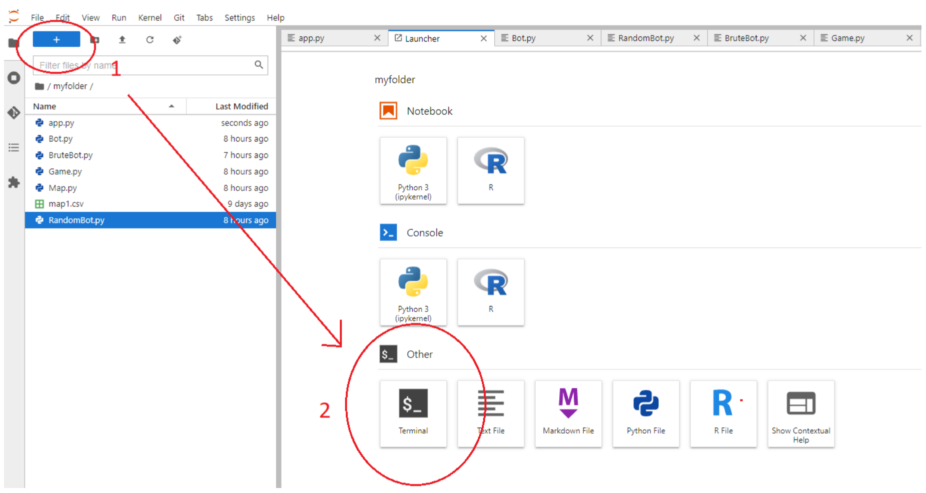


Figure 5: In order to run the game, unzip the project folder and upload its contents in a new folder in jupyter. Then, open a new terminal window. In there, type “cd [your_folder_name]” and press Enter. Then, type “python app.py” and press Enter. In my example the folder is called “myfolder”, therefore in order to run the game, in terminal I type `cd myfolder`, press Enter, then type `python app.py` and press Enter again.

The scoring and rules of the game are implemented there. You will not need to (and should not) modify this file.

- `Map.py` implements the Map class, which represents the game’s map. There, the grid and types of grid cells are defined as well as the method that reads the map from a csv file. You will not need to (and should not) modify this file.

- `app.py` is the game’s executable file. It initialises all game elements, runs the game and outputs the final score. Furthermore, the game’s settings can be defined there (see the `settings` variable).

4.1 What is required from you

To submit your algorithms, you will need to build a class module that inherits `Bot`. To help you understand what that means, two examples have been added as files: `RandomBot` (picks random moves) and `BruteBot` (traverses the entire map, row by row).

Create a copy of either `RandomBot.py` or `BruteBot.py` and give it a different name: ‘BotXXXXXX’ (where XXXXXX is your student number (ANR), e.g. `Bot123456`). Open BotXXXXXX.py and change row 3 into:
`class BotXXXXXX(Bot)`. Moreover, you can give a “nickname” to your bot using the `self.setName()` method.

After applying the above changes, you need to implement your robot's "logic". This should be done within method `nextMove`. In case you need to implement additional methods for your algorithm to run, you should either implement these as class methods or as local functions (within the nextMove() method). Please consult the teaching stuff if you need help implementing additional class methods. Ultimately, nextMove() should return your robot's next move in the form of: `UP, DOWN, LEFT or RIGHT`.

To run the game, change line 28 of `app.py`: `botName = '<bot_name_here>'` (e.g. botName = 'Bot123456'). Lines 24 to 26 of `app.py` can be used for visualisation purposes, namely:

- LATENCY is an integer that defines the "pause" in seconds between successive bot moves. 0 latency means the game will run as fast as your processor allows.
- VISUALS is a boolean which if set to True, will visualise the map at every step of the game.
- CLS is a boolean which if set to True, will clear the terminal output at every step of the game, in order to keep the map at the top of the terminal screen. Set to False if you would like to see the entire history of moves your robot vacuum has made.

The `nextMove` method receives four arguments:

- currentCell is a 2-length list that contains the coordinates of the robot vacuum's current position: [x,y]
- `currentEnergy` is an integer that tells your robot vacuum how much energy it has left
- `vision` is a 3×3 matrix that represents your robot vacuum's current field of vision (see Figure 2)
- `remainingStainCells` is an integer that tells your robot vacuum how many stain cells are left in the map grid.

You will probably need to define new variables for your algorithm to use. These can be defined within the `__init__()` method (see line 4 of BruteBot). Please use `self.` before any variable you define and use. This is necessary when working in object-oriented programs.

Disclaimer: The only information that your robot receives about the map, are the four arguments that `nextMove` receives, alongside the `settings` dictionary that contains some of the game's constants. While you could easily write a method to read the map csv file, this is not allowed and will be considered cheating. However, your robot vacuum is allowed to maintain a "history" of the map cells it has visited or seen. More details about what your bots are allowed to do follow in Sections 9 and 12.

5 Deliverables

The deliverable for this assignment is a) a single .py file, specifically your equivalent of BotXXXXXX class and b) a short report (max 2 pages) that contains your name, ANR, and a description of the algorithm you have implemented. Please name your .py file using the format `BotXXXXXX.py` (e.g. Bot123456.py, where 123456 is your student number (ANR)). Use the same name for your class (see line 3 of BruteBot): `class Bot123456(Bot):`. You can get creative with your robot vacuum's nickname (e.g. self.setName('superCleaner')).

The final step is to upload your .py file and .pdf report on canvas (not zipped). Only a single .py file can be submitted.

6 Grading

Your robot vacuums will be tested in various maps, which may or may not contain obstacles. We will ensure that stains are always reachable (never surrounded by walls). Your algorithms should *at least* solve a map without obstacles with a higher score than `BruteBot` (a simple brute-force search). Below is the grading scheme:

- To get a 6, your robot vacuum should be able to consistently solve maps without obstacles faster than (making less moves than) BruteBot. These maps will have constant map dimensions (30×30) and a constant stain size (3×3).
- To get a 7, your robot vacuum should be able to consistently solve maps without obstacles faster than (making less moves than) BruteBot. These maps will have various dimensions ($N \times N$) and various stain sizes. Note that stain size can be equal to 1, representing a 1-cell stain. Maps will **not** contain stains of different sizes (size will be consistent per map).
- To get an 8, your robot vacuum should be able to consistently solve a map which contains any amount of pillar obstacles (non-labyrinth).
- To get an 9, your robot vacuum should be able to consistently solve a map which contains any amount of pillar and wall obstacles (non-labyrinth).
- To get a 10, your robot vacuum should be able to consistently solve a "labyrinth"-like map (a map that contains non-convex obstacles – non-convex obstacles can be created by placing multiple convex obstacles against each other).

7 Competition

All robot vacuums will be tested on various maps which scale in difficulty. The submission that manages to score the highest average score will receive a custom souvenir gift, plus a +1 point bonus for the assignment grade. If the assignment already received a 10, no bonus point will be granted. Note: this counts for the first submission attempt only.

8 Examples

Two examples are provided: `RandomBot` and `BruteBot`. Either of these can serve as the basis for your implementations. Please create a copy of one of these files and rename it in order to create your own bots.

To run the example bots, change line 28 of `app.py`: `botName = 'RandomBot'` will run the random bot.

9 Submission checklist

1. Make sure your bot class module (.py file) inherits the `Bot` class. Both RandomBot and BruteBot do that; if you use these files as a basis you should face no problems.
2. Make sure you submit a single .py file (your bot class module).
3. Make sure you submit a .pdf report of maximum 2 pages, containing your name and ANR
4. Make sure you name your bot class and module file as follows: `<BotXXXXXX.py>` and `class <BotXXXXXX(Bot):>` where `XXXXXX` is your ANR (e.g. Bot123456.py and class Bot123456(Bot):).
5. Make sure your bot class implements the nextMove method and returns either `UP, DOWN, LEFT` or `RIGHT`.
6. Make sure your bots only use the information provided to them as arguments of the nextMove method and the elements of the settings dictionary. Reading the map file or accessing any other important variables will be considered cheating and will be reported.
7. Make sure your bots do not overwrite any files (e.g. class files or map files). Any intentional modification to any file made by your bot will be considered cheating and will be reported.
8. Make sure your bots do not cause infinite loops and do not "crash" python. If a bot causes a crash or infinite loop in a map, the score for that map will be considered 0.

Submissions that do not fulfil the above conditions will be failed automatically (except for the last point).

10 Using external modules

External modules that are allowed and can be imported are the following:

- numpy
- pandas
- math
- sklearn
- matplotlib
- seaborn
- scipy
- pygame

11 Common issues - troubleshooting

1. Indentation error

   Some of you face errors with indentation when trying to run the assingment. In Jupyter Lab, go to Settings → Text editor indentation and select "Indent with Tab". Also, if you are annoyed by the grey indent tab lines that jupyter tends to display, go to Settings → Text editor theme and select "codemirror". Lastly, make sure that you indent your code using the Tab key and not spacebar. If you indent with 4 spacebars instead of 1 tab, you might still get indentation errors.

2. super() takes at least one argument

   This error occurs when your machine uses Python 2.7. It comes as the default Python installation for some Mac laptops. Please upgrade the Python version that your machine is using to the latest Python (at least 3.10). Quickest solution: use the online jupyter server.

3. My bot keeps going down/right/etc. when it shouldn't

   Your bot will do exactly what your algorithm commands it to do. So if you experience unexpected bot behavior, I suggest you print all the variables that your logic is based on, in every step of the algorithm, and try to validate what should be happening by going through your code line by line. 99% of the time, there is a logical mistake in your implementation.

12 Frequently Asked Questions

- Q: Do I have to pass the assignment to pass the course? A: Yes, as well as the final exam.
- Q: What happens if I fail the assignment? A: There will be an opportunity to re-submit the assignment and improve your grade.
- Q: What is considered a "clever" algorithm? A: At this stage, any algorithm that is more efficient than a brute-force search.
- Q: Can I create my own maps to test my bot? A: Yes, and it is highly encouraged to do so.