BPP Final Project

August 13, 2024

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.

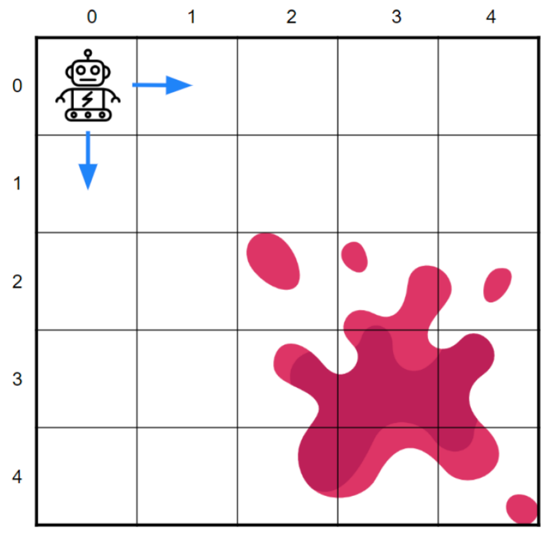


Figure 1: Illustration of the robot vacuum cleaner in a grid-like map. The robot's current position is (0,0) and it can only move right to (1,0) or down to(0,1).

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 2.3

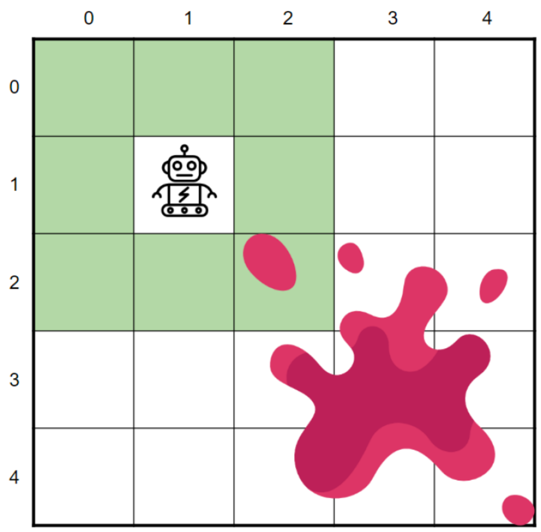


Figure 2: Illustration of the robot vacuum cleaner's range of "vision". It canonly see the 8 surrounding grid cells through its sensors.

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 $\times$ `nrCols` $\times$ `nrRows` energy points. For example, on a 30 $\times$ 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

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 = 1550.

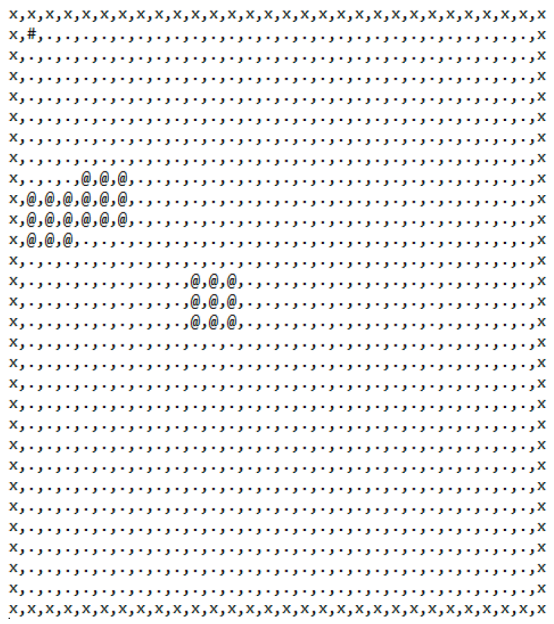
2.2 Map layout

- '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. This should be a sufficient amount of energy 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 ofstains, 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).

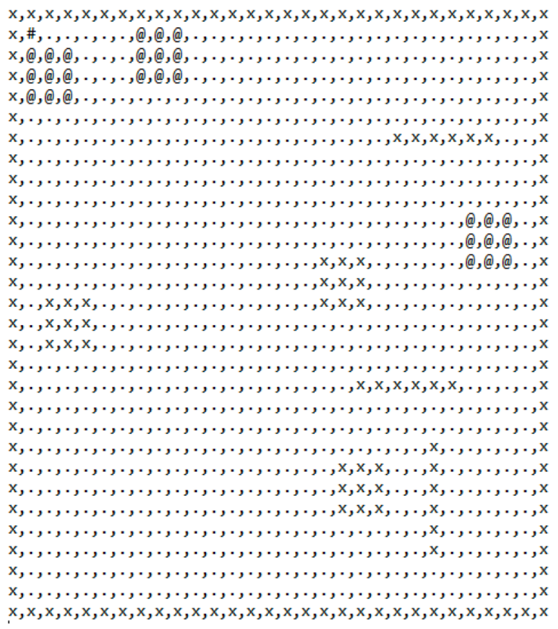


```
x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x,x
x,#,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
x,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,x
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: Figure 3: An example of a csv file that represents a 30 × 30 map.

- The number and size of stains is pre-defined. Stains are always square. See `nrStains` and `sizeStains` in the settings dictionary (app.py).
- 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 in PyCharm

3.1 Getting the files

On Brightspace, you can download the `Final Project.zip` file. Download this file and unzip it.

3.2 Put the Final Project folder in your BPP Project

Find your BPP PyCharm project (you should only be using one project for the entire course) in a file explorer. Move the final folder into your BPP project folder. An example of this is shown in Figure 5.

After you do this, you should be able to see the project in PyCharm, as can be seen in Figure 8.

3.3 Installing necessary packages

In order to run the game, you need to install the `colorama` package. You can do this by opening the terminal (clicking "Terminal" at the bottom of the window)

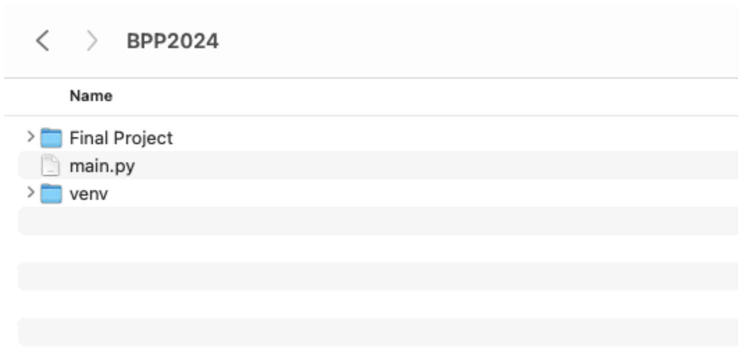


Figure 5: Copying final project folder into BPP project folder

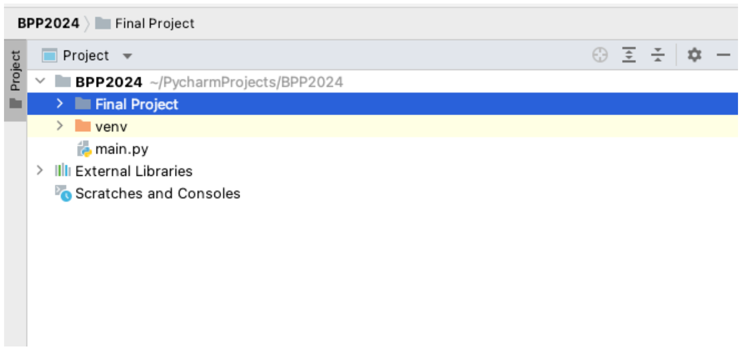
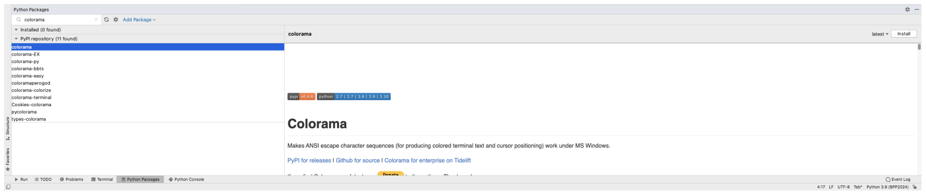


Figure 6: Final project folder in PyCharm

in PyCharm and typing `pip install colorama`. This will install the package for you. Alternatively, you can click on the "Packages" tab at the bottom of the window. Search for colorama, and press install. An example of this is shown in Figure 7.



Colorama

Makes ANSI escape character sequences (for producing colored terminal text and cursor positioning) work under MS Windows.

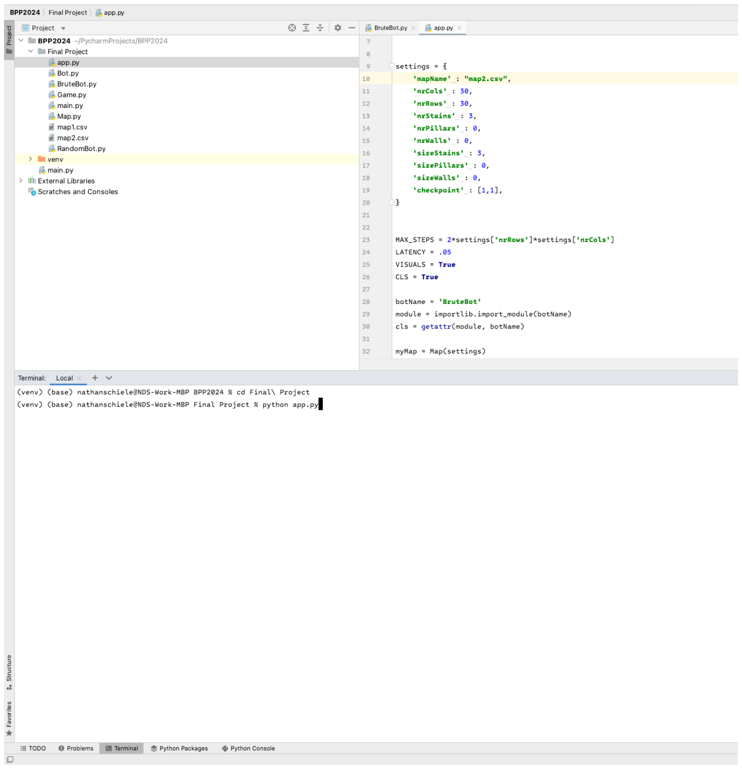
PyPI for releases | Github for source | Colorama for enterprise on Tidelift

Figure 7: Installing colorama

3.4 Running the game

To run the game, you can directly run the `app.py` file the way you've been running file during this course. Unfortunately, the visualization of the game is not great when doing it this way (the display of the game in the console doesn't work the way we'd want it to).

We recommend you open a terminal (press "Terminal" at the bottom of the window), and use the following two commands. The first is `cd Final\ Project`. This will move you into the Final Project folder (`cd` means "change directory"). Then you can run the game by running the command `python app.py`. If you make the terminal window really large, this will cause the visualization of the game to look the best. And example of the commands is shown in Figure **??**.



```python
settings = {
    'mapName' : "map2.csv",
    'nrCols' : 30,
    'nrRows' : 30,
    'nrStains' : 3,
    'nrPillars' : 0,
    'nrWalls' : 0,
    'sizeStains' : 3,
    'sizePillars' : 0,
    'sizeWalls' : 0,
    'checkpoint' : [1,1],
}


MAX_STEPS = 2*settings['nrRows']*settings['nrCols']
LATENCY = .05
VISUALS = True
CLS = True

botName = 'BruteBot'
module = importlib.import_module(botName)
cls = getattr(module, botName)

myMap = Map(settings)
```

```
(venv) (base) nathanschiele@NDS-Work-MBP BPP2024 % cd Final\ Project
(venv) (base) nathanschiele@NDS-Work-MBP Final Project % python app.py
```

Figure 8: Commands needed to run the game in PyCharm

To quit the game, press `ctrl + c` on your keyboard. You can rerun the game by just running the `python app.py` command again. You don't need to change directory again, you only have to do it the first time you open the terminal.

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 will be introduced in this course. The game is divided into several files:

- `Bot.py` implements the Bot class, which represents the robot vacuum cleaner. Bot implements a function called nextMove; this function 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.
- `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 function 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 initializes 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: 'BotXXXXXXX' (where XXXXXXX is your student number, e.g.Bot1234567). Open BotXXXXXXX.py and change row 3 into:
`class BotXXXXXXX(Bot)`. Moreover, you can give a "nickname" to your bot using the `self.setName()` function.

After applying the above changes, you need to implement your robot's "logic". This should be done within function `nextMove`. In case you need to implement additional functions for your algorithm to run, you should either implement these as class functions or as local functions (within the `nextMove()` function). Please consult the lecture materials if you need help implementing additional class functions. 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 visualization 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 visualize 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` function receives four arguments:
- `currentCell` is a 2-length list that contains the coordinates of the robot vacuum's current position: [row, column]
- `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 () function (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 function 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 BotXXXXXXX class

b) a short report (max 2 pages) that contains your name, student number, and a description of the algorithm you have implemented.

Name your .py file using the format BotXXXXXXX.py (e.g. Bot123456.py, where 123456 is your student number). 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 Brightspace (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).
- Half grades are possible, in the case where your robot vacuum is able tocomplete some (but not all) of the maps assigned to a specific grade.

We provide 3 maps of each level, and keep 3 as part of our grading set. We use all six maps for each level for grading. For example, if your robot vacuum can solve three out of six 8-graded maps and six out of six 7-graded maps, it will get a 7.5.

Failure to include the required report on your algorithm will result in a 50% modifier for your grade. The report is used to show that your work is your own; it's your opportunity to show your understanding. If you do not take this opportunity, we may interpret this as your solution not being original. As with all assignments, **you solution must be the result of only your work**. Substantially similar solutions will be considered plagiarized and will be reported to the LIACS Board of Examiners.

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 +1 point bonus for the assignment grade. If the assignment already received a 10, that student will earn a +0.5 bonus point on the final exam grade (worth of +6 out of the 120 point exam).

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 student number
4. Make sure you name your bot class and module file as follows: `BotXXXXXXX.py` and `class BotXXXXXXX(Bot):` where *XXXXXXX* is your student number. (e.g. `Bot1234567.py` and `class Bot1234567(Bot):`).
5. Make sure your bot class implements the nextMove function 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 function 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

Using external modules outside of these may result in something not working. Errors caused by using external modules outside of the above list may result in poor grade.

11 Common issues - troubleshooting

1. Interpreter error

   This occurs when your PyCharm hasn't been set up correctly. Come to the workgroup and have a TA set the interpreter for you.

2. 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 byline. 99.99% of the time, there is a logical mistake in your implementation. While it's possible that my game code contains an error, it's far, far more likely that your bot's logic is flawed.

12 Frequently Asked Questions

- Q: Do I have to pass the assignment to pass the course? A: Technically it is possible to pass this course without passing this assignment, but it requires a very high grade on all the other homework assignments AND the final exam. It's highly recommended to pass this assignment.
- Q: Is there another opportunity to submit the assignment? A: No. You get one attempt to pass the assignment.
- Q: Can I submit the assignment late? A: Yes, however there is a 1 point penalty for every 3 hours you are late. After 27 hours, it is no longer possible to submit the assignment.

- Q: What is considered a “clever” algorithm? A: For this class, 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.

13 Acknowledgements

Credits to Dr. Paris Blom for the assignment design.

References

[1] Al Sweigart. *Automate the boring stuff with Python: practical programming for total beginners.* No Starch Press, 2019.