Vacuum cleaner

# Task summary

We want to clean the Wumpus cave using a vacuum cleaner robot, which we can control with a sequence of instructions. You have to implement two tasks:

1. Check if a sequence of instructions cleans the entire Wumpus cave.
2. Come up with a sequence of instructions yourself – the shorter, the better.

|  |  |  |
| --- | --- | --- |
| XXXXXXX XXXXXX | | |
| XXXXXXXX XXXXXX | | |
| XXXXXXXX | S | XXXXXX |
| XXXXXXXX | X | XXXXXX |
| XXXXXXXX XXXXXXX | | |
| XXXXXXXX XXXXXX | | |
| XXXXXXXXX XXXXXX | | |
| XXXXXXXXXXXXXXXXXX | | |
| XXXXXXXXXXXXXXXXXX | | |
| XXXXX XXXXXXX | | |
| XXXXXX X XXXXXXX | | |
| XXXXXXXX XXXXXX | | |

*⟨*0*,* 0*⟩ ⟨*1*,* 0*⟩ ⟨*2*,* 0*⟩ ⟨*3*,* 0*⟩ ⟨*4*,* 0*⟩ ⟨*5*,* 0*⟩ ⟨*6*,* 0*⟩ ⟨*7*,* 0*⟩ ⟨*8*,* 0*⟩ ⟨*9*,* 0*⟩ ⟨*10*,* 0*⟩ ⟨*11*,* 0*⟩ ⟨*12*,* 0*⟩ ⟨*13*,* 0*⟩ ⟨*14*,* 0*⟩ ⟨*15*,* 0*⟩ ⟨*16*,* 0*⟩ ⟨*17*,* 0*⟩*

**X X X X X X X**

**X X X X X X**

*⟨*0*,* 1*⟩ ⟨*1*,* 1*⟩ ⟨*2*,* 1*⟩ ⟨*3*,* 1*⟩ ⟨*4*,* 1*⟩ ⟨*5*,* 1*⟩ ⟨*6*,* 1*⟩ ⟨*7*,* 1*⟩ ⟨*8*,* 1*⟩ ⟨*9*,* 1*⟩ ⟨*10*,* 1*⟩ ⟨*11*,* 1*⟩ ⟨*12*,* 1*⟩ ⟨*13*,* 1*⟩ ⟨*14*,* 1*⟩ ⟨*15*,* 1*⟩ ⟨*16*,* 1*⟩ ⟨*17*,* 1*⟩*

**X X X X X X X X**

**X X X X X X**

*⟨*0*,* 2*⟩ ⟨*1*,* 2*⟩ ⟨*2*,* 2*⟩ ⟨*3*,* 2*⟩ ⟨*4*,* 2*⟩ ⟨*5*,* 2*⟩ ⟨*6*,* 2*⟩ ⟨*7*,* 2*⟩ ⟨*8*,* 2*⟩ ⟨*9*,* 2*⟩ ⟨*10*,* 2*⟩ ⟨*11*,* 2*⟩ ⟨*12*,* 2*⟩ ⟨*13*,* 2*⟩ ⟨*14*,* 2*⟩ ⟨*15*,* 2*⟩ ⟨*16*,* 2*⟩ ⟨*17*,* 2*⟩*

**X X X X X X X**  **X**

**S**

**X X X X X X**

*⟨*0*,* 3*⟩ ⟨*1*,* 3*⟩ ⟨*2*,* 3*⟩ ⟨*3*,* 3*⟩ ⟨*4*,* 3*⟩ ⟨*5*,* 3*⟩ ⟨*6*,* 3*⟩ ⟨*7*,* 3*⟩ ⟨*8*,* 3*⟩ ⟨*9*,* 3*⟩ ⟨*10*,* 3*⟩ ⟨*11*,* 3*⟩ ⟨*12*,* 3*⟩ ⟨*13*,* 3*⟩ ⟨*14*,* 3*⟩ ⟨*15*,* 3*⟩ ⟨*16*,* 3*⟩ ⟨*17*,* 3*⟩*

**X X X X X X X**  **X**

**X**

**X X X X X X**

*⟨*0*,* 4*⟩ ⟨*1*,* 4*⟩ ⟨*2*,* 4*⟩ ⟨*3*,* 4*⟩ ⟨*4*,* 4*⟩ ⟨*5*,* 4*⟩ ⟨*6*,* 4*⟩ ⟨*7*,* 4*⟩ ⟨*8*,* 4*⟩ ⟨*9*,* 4*⟩ ⟨*10*,* 4*⟩ ⟨*11*,* 4*⟩ ⟨*12*,* 4*⟩ ⟨*13*,* 4*⟩ ⟨*14*,* 4*⟩ ⟨*15*,* 4*⟩ ⟨*16*,* 4*⟩ ⟨*17*,* 4*⟩*

**X X X X X X X**  **X**

**X X X X X X X**

*⟨*0*,* 5*⟩ ⟨*1*,* 5*⟩ ⟨*2*,* 5*⟩ ⟨*3*,* 5*⟩ ⟨*4*,* 5*⟩ ⟨*5*,* 5*⟩ ⟨*6*,* 5*⟩ ⟨*7*,* 5*⟩ ⟨*8*,* 5*⟩ ⟨*9*,* 5*⟩ ⟨*10*,* 5*⟩ ⟨*11*,* 5*⟩ ⟨*12*,* 5*⟩ ⟨*13*,* 5*⟩ ⟨*14*,* 5*⟩ ⟨*15*,* 5*⟩ ⟨*16*,* 5*⟩ ⟨*17*,* 5*⟩*

**X X X X X X X**  **X**

**X X X X X X**

*⟨*0*,* 6*⟩ ⟨*1*,* 6*⟩ ⟨*2*,* 6*⟩ ⟨*3*,* 6*⟩ ⟨*4*,* 6*⟩ ⟨*5*,* 6*⟩ ⟨*6*,* 6*⟩ ⟨*7*,* 6*⟩ ⟨*8*,* 6*⟩ ⟨*9*,* 6*⟩ ⟨*10*,* 6*⟩ ⟨*11*,* 6*⟩ ⟨*12*,* 6*⟩ ⟨*13*,* 6*⟩ ⟨*14*,* 6*⟩ ⟨*15*,* 6*⟩ ⟨*16*,* 6*⟩ ⟨*17*,* 6*⟩*

**X X X X X X X**  **X** **X**

**X X X X X X**

*⟨*0*,* 7*⟩ ⟨*1*,* 7*⟩ ⟨*2*,* 7*⟩ ⟨*3*,* 7*⟩ ⟨*4*,* 7*⟩ ⟨*5*,* 7*⟩ ⟨*6*,* 7*⟩ ⟨*7*,* 7*⟩ ⟨*8*,* 7*⟩ ⟨*9*,* 7*⟩ ⟨*10*,* 7*⟩ ⟨*11*,* 7*⟩ ⟨*12*,* 7*⟩ ⟨*13*,* 7*⟩ ⟨*14*,* 7*⟩ ⟨*15*,* 7*⟩ ⟨*16*,* 7*⟩ ⟨*17*,* 7*⟩*

**X X X X X X X**  **X** **X** **X X X X X X X X X**

*⟨*0*,* 8*⟩ ⟨*1*,* 8*⟩ ⟨*2*,* 8*⟩ ⟨*3*,* 8*⟩ ⟨*4*,* 8*⟩ ⟨*5*,* 8*⟩ ⟨*6*,* 8*⟩ ⟨*7*,* 8*⟩ ⟨*8*,* 8*⟩ ⟨*9*,* 8*⟩ ⟨*10*,* 8*⟩ ⟨*11*,* 8*⟩ ⟨*12*,* 8*⟩ ⟨*13*,* 8*⟩ ⟨*14*,* 8*⟩ ⟨*15*,* 8*⟩ ⟨*16*,* 8*⟩ ⟨*17*,* 8*⟩*

**X X X X X X X**  **X** **X** **X X X X X X X X X**

**X X X X X**

*⟨*0*,* 9*⟩ ⟨*1*,* 9*⟩ ⟨*2*,* 9*⟩ ⟨*3*,* 9*⟩ ⟨*4*,* 9*⟩ ⟨*5*,* 9*⟩ ⟨*6*,* 9*⟩ ⟨*7*,* 9*⟩ ⟨*8*,* 9*⟩ ⟨*9*,* 9*⟩ ⟨*10*,* 9*⟩ ⟨*11*,* 9*⟩ ⟨*12*,* 9*⟩ ⟨*13*,* 9*⟩ ⟨*14*,* 9*⟩ ⟨*15*,* 9*⟩ ⟨*16*,* 9*⟩ ⟨*17*,* 9*⟩*

**X X X X X X X**

**X X X X X** **X**

*⟨*0*,* 10*⟩ ⟨*1*,* 10*⟩ ⟨*2*,* 10*⟩ ⟨*3*,* 10*⟩ ⟨*4*,* 10*⟩ ⟨*5*,* 10*⟩ ⟨*6*,* 10*⟩ ⟨*7*,* 10*⟩ ⟨*8*,* 10*⟩ ⟨*9*,* 10*⟩ ⟨*10*,* 10*⟩ ⟨*11*,* 10*⟩ ⟨*12*,* 10*⟩ ⟨*13*,* 10*⟩ ⟨*14*,* 10*⟩ ⟨*15*,* 10*⟩ ⟨*16*,* 10*⟩ ⟨*17*,* 10*⟩*

**X**

**X X X X X X X**

**X X X X X** **X X**  **X**

*⟨*0*,* 11*⟩ ⟨*1*,* 11*⟩ ⟨*2*,* 11*⟩ ⟨*3*,* 11*⟩ ⟨*4*,* 11*⟩ ⟨*5*,* 11*⟩ ⟨*6*,* 11*⟩ ⟨*7*,* 11*⟩ ⟨*8*,* 11*⟩ ⟨*9*,* 11*⟩ ⟨*10*,* 11*⟩ ⟨*11*,* 11*⟩ ⟨*12*,* 11*⟩ ⟨*13*,* 11*⟩ ⟨*14*,* 11*⟩ ⟨*15*,* 11*⟩ ⟨*16*,* 11*⟩ ⟨*17*,* 11*⟩*

**X X X X X X**

Figure 1: An example map of the Wumpus cave with the starting position marked at *⟨*9*,* 2*⟩*. The text representation (left) is used in the problem files.

# Maps

You have a **map** of the Wumpus cave, which consists of 18 *×* 12 squares. Figure [1](#_bookmark0) shows an example map. Every square has coordinates associated with it. As is common in computer science, the *y*-axis points down and the origin, *⟨*0*,* 0*⟩*, is in the top-left square.

The properties of each square are represented by a single character:

1. Walls are marked with an X.
2. Empty squares are marked with a space.
3. The starting position of the vacuum cleaner (if it is known) is marked with an S.

The maps are stored in the problem files (see Section [4)](#_bookmark4) using a text representation: each row of the map corresponds to a line in the text representation and each square to a character. Figure [1](#_bookmark0) shows an example map with both the text representation and a more visual representation.

# Plans and potentially missed squares

You can control the vacuum cleaner by making a **plan**, which is a sequence of instructions. The following instructions are available:

1. **N**: The vacuum cleaner moves one square *north* (up).
2. **E**: The vacuum cleaner moves one square *east* (right).
3. **S**: The vacuum cleaner moves one square *south* (down).
4. **W**: The vacuum cleaner moves one square *west* (left).

We assume that the vacuum cleaner cleans every square it is on. If the instruction would move the vacuum cleaner onto a wall, it will instead remain on its current square. If the vacuum cleaner leaves the map, it will immediately reappear on the opposite side of the map. A good plan for the vacuum cleaner makes sure that everything gets cleaned. More concretely, we are interested in the **potentially missed squares**, which might still be dirty

after executing the cleaning plan. To illustrate this, we will take a look at a few examples.

**Example: Simple cave** In this example, we will explore what happens if we execute the plan EENS in the cave shown in Figure [2a.](#_bookmark3) We start in *⟨*2*,* 2*⟩*. After the instruction E, we will be in *⟨*3*,* 2*⟩*. Now there is a wall to the east, which means that we will stay in *⟨*3*,* 2*⟩* when we try to go east again. Next, we go north (*⟨*3*,* 1*⟩*) and then south (*⟨*3*,* 2*⟩* again). Therefore, we will not have cleaned *⟨*1*,* 1*⟩*, *⟨*2*,* 1*⟩* and *⟨*1*,* 2*⟩*.

**Example: No boundary** In this example, we will explore what happens if we execute the plan WWWNN in Figure [2b.](#_bookmark3) When going west once, we are in *⟨*0*,* 1*⟩*. Going west again, we will reappear on the right side in *⟨*4*,* 1*⟩* (we assume that the map has size 5 *×* 5 instead of the actual size 18 *×* 12). After going west and then north, we will be in *⟨*3*,* 0*⟩* and after going north one more time we will reappear at the bottom in *⟨*3*,* 4*⟩*. Therefore, *⟨*3*,* 3*⟩* and

*⟨*3*,* 2*⟩* will be missed in the cleaning process.

**Example: Initial position unknown** In this example, we will explore what happens if we execute the plan WENS in the cave shown in Figure [2c.](#_bookmark3) If we start in position *⟨*0*,* 1*⟩* or *⟨*4*,* 1*⟩*, we will clean everything except for *⟨*0*,* 2*⟩*. Similarly, if we start in *⟨*0*,* 2*⟩* or *⟨*4*,* 2*⟩*, we

will clean everything except for *⟨*0*,* 1*⟩*. Therefore, we will potentially miss *⟨*0*,* 1*⟩* and *⟨*0*,* 2*⟩*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **X**  *⟨*0*,* 0*⟩* | **X**  *⟨*1*,* 0*⟩* | **X**  *⟨*2*,* 0*⟩* | **X**  *⟨*3*,* 0*⟩* | **X**  *⟨*4*,* 0*⟩* |
| **X**  *⟨*0*,* 1*⟩* | *⟨*1*,* 1*⟩* | *⟨*2*,* 1*⟩* | *⟨*3*,* 1*⟩* | **X**  *⟨*4*,* 1*⟩* |
| **X** |  | **S** |  | **X** |
|  |  |  |  |  |
| **X** |  | **X** | **X** | **X** |
|  |  |  |  |  |
| **X**  *⟨*0*,* 4*⟩* | **X**  *⟨*1*,* 4*⟩* | **X**  *⟨*2*,* 4*⟩* | **X**  *⟨*3*,* 4*⟩* | **X**  *⟨*4*,* 4*⟩* |

|  |  |  |  |
| --- | --- | --- | --- |
| **X**  *⟨*0*,* 0*⟩* | **X X**  *⟨*1*,* 0*⟩ ⟨*2*,* 0*⟩* | *⟨*3*,* 0*⟩* | **X**  *⟨*4*,* 0*⟩* |
| *⟨*0*,* 1*⟩*  **X**  *⟨*0*,* 2*⟩* | **S**  **X**  *⟨*1*,* 1*⟩ ⟨*2*,* 1*⟩*  **X**  *⟨*1*,* 2*⟩ ⟨*2*,* 2*⟩*  **X** **X**  *⟨*1*,* 3*⟩ ⟨*2*,* 3*⟩* | *⟨*3*,* 1*⟩*  *⟨*3*,* 2*⟩* | *⟨*4*,* 1*⟩*  **X**  *⟨*4*,* 2*⟩* |
| **X**  *⟨*0*,* 3*⟩* | **X**  *⟨*4*,* 3*⟩* |
| *⟨*3*,* 3*⟩* |
| **X** **X** **X**  *⟨*0*,* 4*⟩ ⟨*1*,* 4*⟩ ⟨*2*,* 4*⟩* | | **X**  *⟨*4*,* 4*⟩* |
| *⟨*3*,* 4*⟩* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **X**  *⟨*0*,* 0*⟩* | **X**  *⟨*1*,* 0*⟩* | **X**  *⟨*2*,* 0*⟩* | **X**  *⟨*3*,* 0*⟩* | **X**  *⟨*4*,* 0*⟩* |
|  | **X** | **X** | **X** |  |
|  |  |  |  |  |
|  | **X** | **X** | **X** |  |
|  |  |  |  |  |
| **X** | **X** | **X** | **X** | **X** |
|  |  |  |  |  |
| **X**  *⟨*0*,* 4*⟩* | **X**  *⟨*1*,* 4*⟩* | **X**  *⟨*2*,* 4*⟩* | **X**  *⟨*3*,* 4*⟩* | **X**  *⟨*4*,* 4*⟩* |

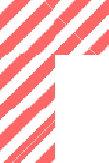
* 1. plan EENS
  2. plan WWWNN
  3. plan WENS

Figure 2: Example caves. The potentially missed squares are marked with diagonal lines ().

# Problem and solution files

Your implementation is supposed to generate a **solution file** for each problem file. This section describes the format of problem and solution files.

## Checking plans

The easier problem files require you to check a cleaning plan. They begin with the line CHECK PLAN, followed by a plan as described in Section [3,](#_bookmark2) followed by the text representa- tion of a map as described in Section [2.](#_bookmark1)

If there are no potentially missed squares, the solution file should contain the text GOOD PLAN. Otherwise, the solution file should contain the text BAD PLAN, followed by a list of the potentially missed squares (the order does not matter). For example, if the squares

*⟨*2*,* 3*⟩* and *⟨*1*,* 5*⟩* are potentially missed, the solution file should be

BAD PLAN 2, 3

1, 5

## Finding plans

The more difficult problem files require you to find a cleaning plan. They begin with the line FIND PLAN, followed by the text representation of a map. The solution file should then

contain the plan as described in Section [3.](#_bookmark2)

# Guide for getting started (optional)

The “guide for getting started” is a new attempt to support students with less programming experience. Concretely, we want to guide you with a series of comments and questions towards a (partial) solution and give you an idea how larger programming problems can be tackled. If you think that you do not need it, you can simply ignore it.

Please let us know if you find this guide useful (then we might provide it in future assignments as well)! We also appreciate if you suggest improvements.

## How to use this guide?

This guide poses a lot of questions. You should try to think about them for yourself and implement some code for it – our hope is that that process will help you get started.

For example, if there is a question “what squares are adjacent to *⟨*3*,* 5*⟩*?” you could create

the following (Python) code:

**def** get\_neighbours(x, y):

**return** [(x+1, y), (x-1, y), (x, y-1), (x, y+1)]

**print**(get\_neighbours(3, 5))

Of course there are many other ways (e.g. creating a Square class with a get\_neighbours

method). However, doing

**print**([(4, 5), (2, 5), (3, 4), (3, 6)]

or

**print**([(3+1, 5), (3-1, 5), (3, 5-1), (3, 5+1)])

would not be enough because it does not solve the problem of finding adjacent squares in general.

Usually, it is a good idea to test your implementations with more examples to make sure that you cover all the cases. This guide tends to intentionally skip over special cases so that you will discover them at some point yourself, which we consider a valuable teaching moment.

## Overall approach

There are different ways to approach an assignment like this one. Here, we will first load the data from a problem file and afterwards try to work on the actual problem solving.

Another option would be to hard-code a map and a plan in the source code and implement the problem solving part before trying to actually load problem files.

## Loading a problem file

Goal: load the data from a problem file into a representation that is easy to work with. The format of a problem file is described in Section [4.](#_bookmark4) “Inspirational” questions and comments for you to think about and write some code for (see also Appendix [A.1](#_bookmark10)):

* + 1. Does problem\_a\_04.txt require you to check a plan or to find one? (check the first

line, and remember that you should create code for answering the question)

* + 1. What is the plan that has to be checked in problem\_a\_03.txt?
    2. What is the text representation of the map in problem\_a\_03.txt? (This could e.g. be a string or a list of strings.)
    3. Try to think of a good way to represent the map. There are many ways to do this. You could even keep it as a string.
    4. Load the map from problem\_a\_05.txt into that representation.
    5. Is square *⟨*4*,* 2*⟩* from problem\_a\_05.txt blocked? (Test this thoroughly by checking more squares – if you make a mistake, it might get very confusing later on.)
    6. Design your code in a way that it is very easy to check if a square *⟨x, y⟩* is blocked (we might have to do that a lot).
    7. Does problem\_a\_08.txt have a starting position?
    8. What are the coordinates of the starting position in problem\_a\_08.txt?

At this point you should be able to load a problem file with your code and access the map data conveniently. With that in place, we can try to solve the first problem files.

## Solving part a

Goal: solve the first problems. “Inspirational” questions and comments:

* + 1. What square do I land on of if I walk N (north) from square *⟨*2*,* 8*⟩*? (Try this with other directions and make sure there are no bugs.)
    2. Answer the same question for different squares, but this time use a map (e.g. from

problem\_a\_00.txt) and only change the position if the new square is free.

* + 1. Make sure that everything from Appendix [A.3](#_bookmark11) is ready:
       1. What is the plan in problem\_a\_00.txt?
       2. What is the starting position in problem\_a\_00.txt?
    2. What is the first instruction of the plan in problem\_a\_00.txt?
    3. What happens after executing the first action of the plan in problem\_a\_00.txt?
    4. What squares do you cover when following all the instructions of problem\_a\_00.txt?
    5. What are the free squares in problem\_a\_00.txt?
    6. Are all the free squares covered by the instructions in problem\_a\_00.txt?
    7. What squares are not covered by the instructions in problem\_a\_00.txt?
    8. Create a solution file solution\_a\_00.txt for problem\_a\_00.txt (see Section [4).](#_bookmark4)

## Automation

You should now be able to create solution files for the first problem files. If you haven’t done so yet, it is a good idea to automate everything as much as possible. Basically, you want to have code that will automatically create all the solution files. You should not have to change file paths in your script to solve different problems or manually copy-paste the output into a solution file.

The assignment repository contains example problems files with their solutions. It makes sense to have a tool that automatically compares all your solutions to the example solutions to help you find and debug potential bugs.

## Continuing

We will not provide detailed instructions for the next problems, but we hope that the previous sections helped you get started. Section [6](#_bookmark5) might contain some helpful tips. If you feel like you need some more guidance, you are invited to come to the office hours or ask in the matrix room for this assignment.