— GUIDE —

Assignment 0 (Warm-Up): Clean the Wumpus Cave

AI-1 Systems Project (Winter Semester 2025/2026)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

This document is intended to help you solve the assignment "Assignment 0 (Warm-Up): Clean the Wumpus Cave" [AS]. You do not have to read it, but we do recommend to at least take a look at the tips and common issues.

1 A few tips

1. If you do not really understand what you are supposed to do, please consider asking about it in the office hours. This is fairly common and it would be a pity if you drop out just because you are a bit lost in the beginning. The project is probably quite different from the courses you are used to.
2. If you do not really know where to start, you might want to read the "How to get started"-section (Section 3).
3. Start with the simpler problems (in particular, the plan checking).
4. The assignment repository [AR] contains example files and example solutions that you can use to test your implementation. It also contains a script for checking your solutions for the example problems.
5. Finding plans is not easy. Here are a few tips to help you get started:
   (a) Do not try to find optimal (shortest) plans. It is too difficult for the larger caves.
   (b) A very simple starting point is to make a random plan: if you have a long sequence of randomly selected instructions, it is probably a valid plan. Then you can try to think of ways to make the generated plan shorter. You can also try to modify the random generation in a way that it tends to select instructions that will lead to a shorter plan. How could you identify "good instructions" to continue your plan?
   (c) Another starting point is to build up a plan step by step and always try to do something that gets you a little closer to the goal (e.g. clean one more square).

(d) For problems with unknown initial states, you can start by finding a plan that works for one particular initial state. Then you can add more actions to make it work for another possible initial state and so on. Typically, a plan that works for one initial state, will already clean many squares for other possible initial states.

6. Use your tools for checking plans to make sure that the plans you find are correct.
7. Efficiency matters for these problems, but you do not have to use an extremely fast programming language (e.g. Python is completely sufficient). What really matters is the asymptotic time complexity.
8. Implement a loop for solving all problem files (instead of manually changing the file name in the code).

2 Common issues

Common problems in the submissions

1. The repository does not contain the solution files.
2. The repository does not contain the solution summary (see [SoS]).
3. The solution files are not in the correct format/have wrong file names.
4. The solution files contain mistakes (you generally do not get points for these).

Clarifications on common misconceptions

1. For problems where the initial state is unknown, the plan has to work for *every* possible initial state.
2. We do not require your code to be very efficient. If it takes an hour to compute all the solutions, that is okay with us.

3 How to get started (optional)

The "How to get started"-section 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 section useful (then we might provide it in future assignments as well)! We also appreciate if you suggest improvements.

3.1 How to use this section?

This section 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 $\langle 3, 5\rangle$?" 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. We tend to intentionally skip over special cases so that you will discover them at some point yourself, which we consider a valuable teaching moment.

3.2 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.

3.3 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 the assignment sheet. "Inspirational" questions and comments for you to think about and write some code for (see also Section 3.1):

1. Does problem_a_04.txt[1] 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)
2. What is the plan that has to be checked in problem_a_03.txt?
3. 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.)
4. 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.
5. Load the map from problem_a_05.txt into that representation.
6. Is square $\langle 4, 2\rangle$ 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.)
7. Design your code in a way that it is very easy to check if a square $\langle x, y\rangle$ is blocked (we might have to do that a lot).
8. Does problem_a_08.txt have a starting position?
9. What are the coordinates of the starting position in problem_a_08.txt?
10. What is the initial orientation 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.

3.4 Solving part a

Goal: solve the first problems. "Inspirational" questions and comments:

1. What square do I land on if I walk one square (action M) from square $\langle 2, 8\rangle$ when if I initially face up? (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.
3. Make sure that everything from Section 3.3 is ready:
   (a) What is the plan in problem_a_00.txt?
   (b) What is the starting position in problem_a_00.txt?
   (c) What is the initial orientation in problem_a_00.txt?
4. What is the first instruction of the plan in problem_a_00.txt?
5. What happens after executing the first action of the plan in problem_a_00.txt?
6. What squares do you cover when following all the instructions of problem_a_00.txt?
7. What are the free squares in problem_a_00.txt?

[1] There is nothing special about problem_a_04.txt – it is just an example. Try out different problem files and see if your code works for all of them. Same with all the other problems mentioned in this guide.

8. Are all the free squares covered by the instructions in problem_a_00.txt?
9. What squares are not covered by the instructions in problem_a_00.txt?
10. Create a solution file solution_a_00.txt for problem_a_00.txt (make sure you precisely follow the specification for solution files from the assignment).

3.5 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. You can use that for debugging and testing your code. It also has a script for checking whether solutions for the example problems are correct. We strongly recommend that you use your code to create solutions for the example problems and check them with the script.

3.6 Continuing

We will not provide detailed instructions for the next problems, but we hope that the previous sections helped you get started. Section 1 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.

References

[AR] *Repository for Assignment 0 (Warm-Up): Clean the Wumpus Cave*. URL: https://gitlab.rrze.fau.de/wrv/AISysProj/ws2526/a1.0-clean-wumpus-cave/assignment.

[AS] *Assignment 0 (Warm-Up): Clean the Wumpus Cave*. URL: https://kwarc.info/teaching/AISysProj/WS2526/assignment-1.0.pdf.

[SoS] *Solution Summary*. URL: https://gitlab.rrze.fau.de/wrv/AISysProj/admin/general/-/blob/main/solution-summary.md.