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

Topic:	Agents in AI, search
Latest submission:	February 15, 2026
Version from:	October 20, 2025
Author:	Jan Frederik Schaefer
Important notes:	Ask for help if you are stuck (office hours, assignment room, ...)
	Every assignment has a guide with tips – you can find it at [AG]

This assignment has to be solved individually (not as a team).
Using someone else's solution code, even as inspiration, is not allowed!
Sharing your solution code with other AISysProj students is not allowed.

1 Task summary

A recurring theme in the AI lecture is the Wumpus world. The Wumpus is a mysterious creature that lives in a cave that is organized as a grid of squares. 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.

The assignment repository [AR] contains files with problem representations that you have to solve. Your grade will largely be based on those solutions (see Section 6). The assignment repository also contains example solutions that you can use to test your implementation.

Didactic objectives

1. Develop an algorithm to solve a non-trivial problem,
2. implement a small software project from scratch,
3. get hands-on experience with a search problem,
4. improve the efficiency of an algorithm,
5. get to know the AISysProj setup and workflows.

Prerequisites and useful methods

1. The basics of computer science and programming,
2. Search (in a very general sense).

2 Maps

You have a **map** of the Wumpus cave, which consists of 18×12 squares. Figure 1 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, $\langle 0, 0 \rangle$, 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.
4. If the initial orientation of the vacuum cleaner is also known, the starting position is instead marked with a
   (a) ^ to indicate that it faces up,
   (b) > to indicate that it faces right,
   (c) v to indicate that it faces down,
   (d) < to indicate that it faces left.

The maps are stored in the problem files (see Section 4) 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 shows an example map with both the text representation and a more visual representation.

3 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. L: The vacuum cleaner turns 90 ° to the left.
2. R: The vacuum cleaner turns 90 ° to the right.
3. M: The vacuum cleaner moves 1 square in the direction it is currently facing.

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.

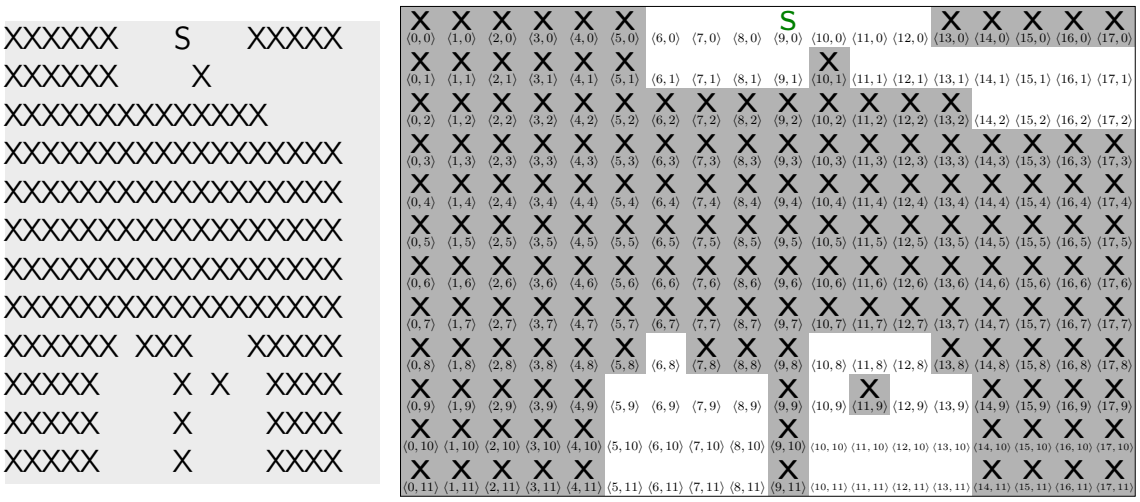


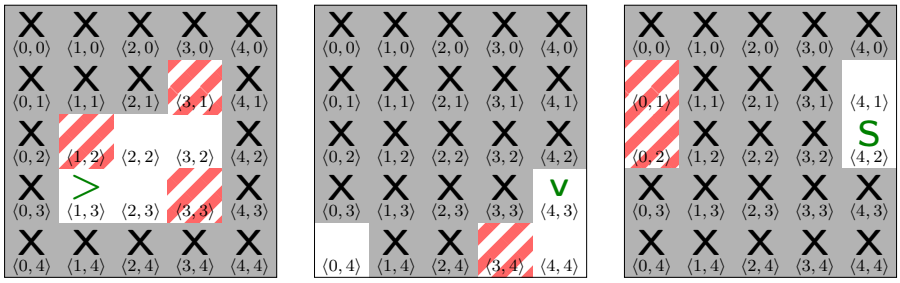
Figure 1: An example map of the Wumpus cave with the starting position marked at $\langle 9, 0\rangle$The text representation (left) is used in the problem files.

Plans are either good or bad. A plan is **good** if every empty square will definitely be cleaned by the vacuum cleaner. However, if some empty squares are potentially missed, the plan is **bad**. The idea of potentially missed squared can be a bit tricky, so we will take a look at a few examples below.

Example: Simple cave In this example, we will explore what happens if we execute the plan MLMMRM in the cave shown in Figure 2a. We start in $\langle 1, 3\rangle$ facing to the right. After executing M, we will be in $\langle 2, 3\rangle$. After executing L, we will face up, which means that after executing M, we will be in $\langle 2, 2\rangle$. If we execute M another time, we will remain in $\langle 2, 2\rangle$ because $\langle 2, 1\rangle$ is blocked. After executing R and M, we will be in $\langle 3, 2\rangle$. Therefore, we will not have cleaned $\langle 3, 1\rangle$, $\langle 1, 2\rangle$ and $\langle 3, 3\rangle$.

Example: No boundary In this example, we will explore what happens if we execute the plan MMLM in Figure 2b. We start in $\langle 4, 3\rangle$ facing south. After executing M, we are in $\langle 4, 4\rangle$. In this position, executing M will not work because $\langle 4, 0\rangle$ is blocked by a wall, so we remain in $\langle 4, 4\rangle$. After executing L, we will face east, which means that after executing M, we will reappear at the left side in $\langle 0, 4\rangle$. Therefore, in the end, $\langle 3, 4\rangle$ will be missed in the cleaning process.

Example: Initial orientation unknown In this example, we will explore what happens if we execute the plan MRMRMRM in the cave shown in Figure 2c. If we initially face left or



(a) plan MLMMRM (b) plan MMLM (c) plan MRMRMRM

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

up, we will clean all the squares. However, if we initially face right, we will not clean $\langle 0, 1\rangle$. And if we initially face down, we will not clean $\langle 0, 2\rangle$. Therefore, we will potentially miss squares $\langle 0, 1\rangle$ and $\langle 0, 2\rangle$.

4 Problem and solution files

The assignment repository [AR] contains many **problem files**. Your implementation is supposed to generate a **solution file** for each problem file. This section describes the format of problem and solution files.

4.1 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, followed by the text representation of a map as described in Section 2.

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 $\langle 2, 3\rangle$ and $\langle 1, 5\rangle$ are potentially missed, the solution file should be

```
BAD PLAN
2, 3
1, 5
```

4.2 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.

If the format is not clear, you can take a look at the assignment repository [AR], which contains example problems and solutions.

> **Important:** The number of points for plan finding problems depends on the plan lengths (see Section 6 for details).

5 What to submit

You should push your solution to your git repository for this assignment. Concretely, your repository should contain:

1. all your code for solving this assignment,
2. a README.md file explaining
   i. dependencies (programming language, version, external libraries and how to get them),
   ii. how to run your code to solve other problems,
   iii. the repository structure,
   iv. anything else we should know,
3. a solution summary (see [SoS] for more details – it should describe the main ideas, not document the code),
4. solution files (as described in Section 4) for the problem files. The solution file for problem_X_YZ.txt should be called solution_X_YZ.txt.

6 Points

The total number of points for this assignment is 100. To pass the assignment, you have to get at least 50 points. Up to 80 points are awarded for the solutions to the problem files. Figure 3 shows how many points can be achieved for each part. For the FIND PLAN problems, the number of points depends on the total plan length T of your solutions, i.e. T is the sum of the lengths of the plans you found for that part.

Note that partial points (for solving only part of a problem range correctly) are only awarded in exceptional cases.

The remaining 20 points are awarded for the submission quality. The points are primarily awarded for the solution summary (see [SoS]), but it also includes the README (instructions on how we can run your code) and the overall organization of your repository (can we find the files? are they in the correct format? etc.) Note that we do not grade the code quality itself.

You cannot get points for the submission quality if you don't get points for the solutions.

If the grading scheme doesn't seem to work well, we might adjust it later on (likely in your favor).

Important: You get points for *correct* solutions. You generally do not e.g. get partial points for code that "looks roughly correct but produces wrong results". The assignment repository contains a script that you can use to check your solutions for the example problems.

References

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

[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.

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

Problems	**Mode**	**Challenges**	**points**
problem_a_*.txt	check	—	10
problem_b_*.txt	check	no boundary	10
problem_c_*.txt	check	unknown orientation no boundary	10
problem_d_*.txt	find	—	$\begin{cases} 15 & \text{if } T \leq 10000 \\ 10 & \text{if } T \leq 30000 \\ 5 & \text{if } T \leq 60000 \\ 0 & \text{if } T > 60000 \end{cases}$
problem_e_*.txt	find	no boundary	$\begin{cases} 15 & \text{if } T \leq 20000 \\ 10 & \text{if } T \leq 40000 \\ 5 & \text{if } T \leq 60000 \\ 0 & \text{if } T > 60000 \end{cases}$
problem_f_*.txt	find	unknown orientation no boundary	$\begin{cases} 20 & \text{if } T \leq 20000 \\ 15 & \text{if } T \leq 40000 \\ 10 & \text{if } T \leq 60000 \\ 5 & \text{if } T \leq 80000 \\ 0 & \text{if } T > 80000 \end{cases}$

Figure 3: Points per part.