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

AI-1 Systems Project (Summer Semester 2026)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic:	Agents in AI, search
Deadline:	May 16, 2026
Extended deadline:	July 18, 2026
Version from:	April 28, 2026
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].
- The extended deadline is granted automatically.
- The extended deadline is after the deadlines of some other assignments, i.e. submitting at the extended deadline means you cannot finish them this semester.

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, including its string representation (left) and a more visual representation with the coordinate system used in the assignment (right).

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. Pits (if they exist) are marked with `p`.

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. `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).

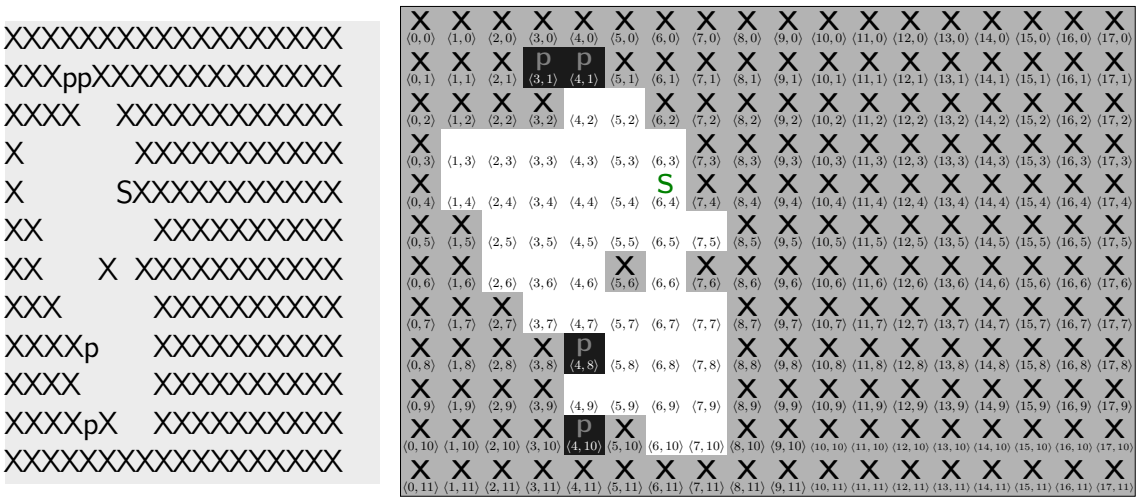


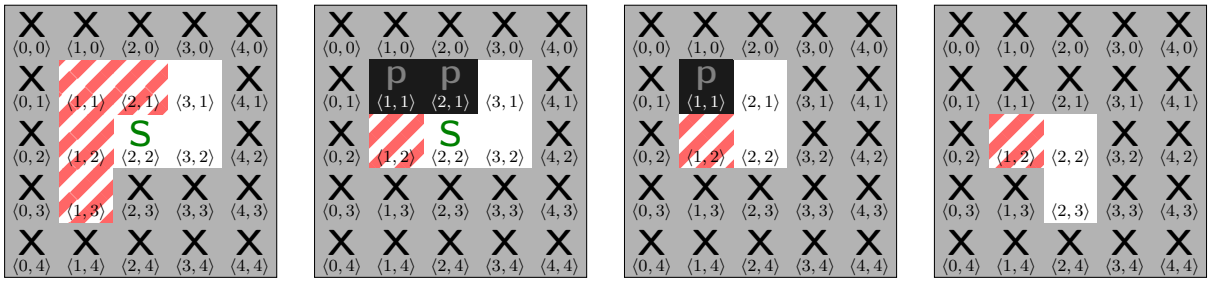
Figure 1: An example map of the Wumpus cave with the starting position marked at $\langle 6, 4\rangle$ and several pits marked with ps. The text representation (left) is used in the problem files.

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.

Plans are either good, bad or terrible. 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. Whenever the vacuum cleaner might fall into a pit when executing the plan, we call the plan **terrible** – no matter whether it cleans all the squares or not.

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

Example: Pits If we execute the plan EN in the cave shown in Figure 2b, we will clean all squares except for $\langle 1, 2\rangle$. However, if we had executed the plan ENWS, the vacuum cleaner would have fallen into the pit at $\langle 2, 1\rangle$, i.e. it would be a terrible plan and no potential missed squares would be have to be listed.



(a) plan EENS (b) plan EN (c) plan ENS (d) plan EENS

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

Example: Pits and initial position unknown This example explores what happens if we execute the plan ENS in the cave shown in Figure 2c. If we start in $\langle 1, 2\rangle$, we will clean the entire cave. However, if we start in $\langle 2, 2\rangle$ or $\langle 2, 1\rangle$, we will only clean $\langle 2, 2\rangle$ and $\langle 2, 1\rangle$ and miss $\langle 1, 2\rangle$. Therefore, we will potentially miss $\langle 1, 2\rangle$. If we had executed the plan EWNS, the vacuum cleaner could have fallen into a pit (when starting in $\langle 2, 1\rangle$), and it would be considered a terrible plan (and no potentially missed squares would have to be listed). A valid plan for the cave would be ENSW.

Example: Initial position unknown In this example, we will explore what happens if we execute the plan EENS in the cave shown in Figure 2d. If we start in position $\langle 1, 2\rangle$, we will clean the entire cave. However, if we start in $\langle 2, 2\rangle$ or $\langle 2, 3\rangle$, we will only clean $\langle 2, 2\rangle$ and $\langle 2, 3\rangle$. Therefore, we will potentially miss $\langle 1, 2\rangle$.[1] EdN:1

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.

[1] EdNote: TODO: the example can be improved (equivalently to fig:e9)

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
```

An exception to this rule are terrible plans, i.e. plans that might make the vacuum cleaner fall into a pit. In this case, the solution file should contain the text TERRIBLE PLAN, without any further information.

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.
5. An empty file submitted.txt to indicate that you have submitted your solution.

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 90 points are awarded for the solutions to the problem files. The last 10 points (for the problem_g_*.txt, which are particularly difficult) are considered bonus points, so you can get more than 100 points for this assignment. This is an exception (most assignments will not have bonus points). 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.

Problems	**Mode**	**Challenges**	**points**
problem_a_*.txt	check	—	10
problem_b_*.txt	check	pits	10
problem_c_*.txt	check	unknown start position pits	10
problem_d_*.txt	find	—	15 if $T \leq 10000$ 10 if $T \leq 25000$ 5 if $T \leq 40000$ 0 if $T > 40000$
problem_e_*.txt	find	pits	15 if $T \leq 10000$ 10 if $T \leq 25000$ 5 if $T \leq 40000$ 0 if $T > 40000$
problem_f_*.txt	find	unknown start position	20 if $T \leq 10000$ 15 if $T \leq 15000$ 10 if $T \leq 25000$ 5 if $T \leq 40000$ 0 if $T > 40000$
problem_g_*.txt	find	unknown start position pits	10 if $T \leq 5000$ 5 if $T \leq 20000$ 0 if $T > 20000$

Figure 3: Points per part.

References

[AG] *Guide for "Assignment 0 (Warm-Up): Clean the Wumpus Cave"*. URL: https://kwarc.info/teaching/AISysProj/SS26/assignment-1.0-guide.pdf.

[AR] *Repository for Assignment 0 (Warm-Up): Clean the Wumpus Cave*. URL: https://gitlab.rrze.fau.de/wrv/AISysProj/ss26/a1.0-clean-wumpus-cave/assignment.

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