Assignment 5: Escape the Wumpus Cave

AI-1 Systems Project (Summer Semester 2025)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic:	Planning
Due on:	September 30, 2025
Version from:	July 30, 2025
Author:	Jan Frederik Schaefer

Make sure you sign up before working on this assignment.[a]
Using someone else's solution code, even as inspiration, is not allowed!
Sharing your solution code with other AISysProj students is not allowed.

[a]You can still decide to postpone the assignment. Signing up includes an elligibility check, which avoids situations where you invest work into an assignment that you are not supposed to take.

1 Task summary

Your agent is trapped in the Wumpus world. Find a way for your agent to leave it using a PDDL planner. The assignment repository [AR] contains maps for you to solve. It also contains a script for checking your solutions.

Didactic objectives

1. Get to know the PDDL format,
2. learn how to encode a problem as a planning problem.

Prerequisites and useful methods

1. Planning and the PDDL format.

2 The Wumpus world

Maps of the Wumpus world are encoded as a text file, where each line corresponds to a row of cells and each character describes the properties of an individual cell. Here is an example map:

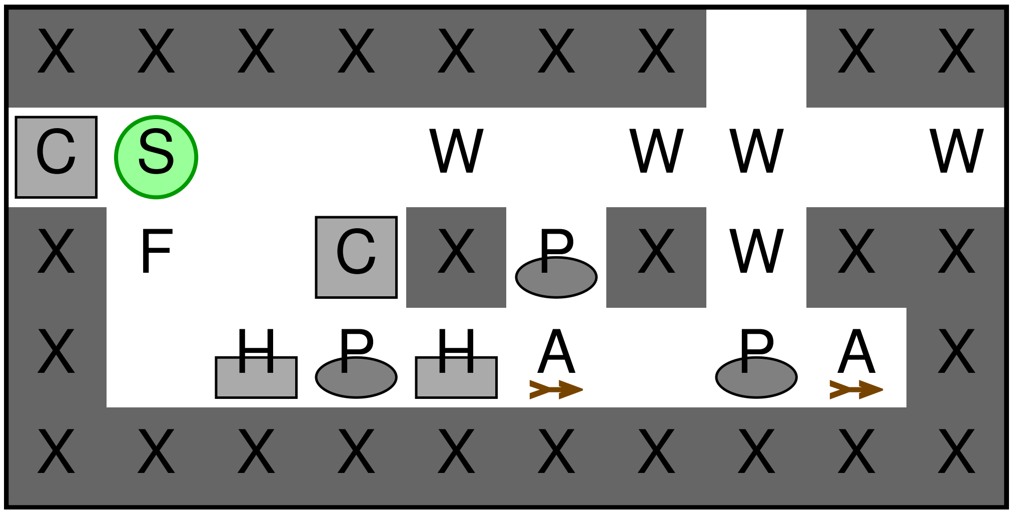


Figure 1: Example map of the Wumpus world.

```
XXXXXXX␣XX
CS␣␣W␣WW␣W
XF␣CXPXWXX
X␣HPHA␣PAX
XXXXXXXXXX
```

Your starting position is marked with an `S`. Leaving the wumpus world is not easy because cells can be blocked by e.g. walls (`X`) or a wumpus (`W`). Section 2.2 describes the different cells in more detail. Figure 1 shows a visualization of the example map.

2.1 Plans

Your plan is a sequence of actions. After performing those actions, your agent should be "off the map". You can use the following actions:

- `walk [north|east|south|west]`: The agent walks to the neighboring cell in the indicated direction. If there is no cell in that direction, the agent has left the map (which is the goal). Whether you are allowed to walk to a cell depends of course on the map – for example, you cannot walk into a wall. Section 2.2 has more details.
- `push [north|east|south|west]`: The agent pushes an object to the neighboring cell in the indicated direction. The agent will also walk one step in the indicated direction. Section 2.2 describes what things can be pushed under what circumstances.
- `shoot [north|east|south|west]`: Shoot a Wumpus. This requires a Wumpus to be

in the neighboring cell in the indicated direction. You also must have at least one arrow. Afterwards, you have one arrow less and the Wumpus is gone.

- `scare [north|east|south|west]`: Scare the Wumpus with fireworks. This requires that you have at least one fireworks set and that there is a Wumpus in the neighboring cell in the indicated direction. It also requires that the cell behind the Wumpus is free, (no crates/pits/... or anything that would normally prevent the agent from walking there, see Section 2.2 for more details). The fireworks scare the Wumpus and it will move one cell away from the agent. You cannot re-use the fireworks.

Note that these actions do not have to correspond to the actions that you use for the planning itself. Instead, you can convert the planner actions to the ones above in a post-processing step (however, it should be a fairly simple transformation, i.e. the planner has to do the hard work).

2.2 Cell properties

As mentioned earlier, the properties of each cell are summarized by a single character:

- `S`: The starting position of the agent. Otherwise the cell is empty. Note that we also use `S` to mark the current position of the agent in visualizations.
- `␣` (a single whitespace): An empty cell.
- `X`: A wall. Your agent cannot walk into a cell with a wall.
- `W`: A wumpus. Your agent cannot walk into a cell with a Wumpus. However, the Wumpus can be shot with an arrow. Afterwards, the cell is free. You can also scare the Wumpus with fireworks to make it move one cell.
- `A`: An arrow. If you walk into a cell with an arrow, you automatically pick it up (and have one more arrow). Afterwards, the cell is empty.
- `F`: A fireworks set. If you walk into a cell with a fireworks set, you automatically pick it up and can use it in a future action. Afterwards, the cell is empty.
- `C`: A crate. Your agent cannot walk into a cell with a crate. However, your agent can `push` (see Section 2.1) the crate to an adjacent cell if the adjacent cell fullfills one of the following conditions:
  - It is empty.
  - It contains only an item that can be picked up. The adjacent cell will then contain both the item and the crate.
  - It has a pit (see the description of pits for details).
- `H`: A half-crate. Your agent cannot walk into a cell with a half-crate. However, you can

push a half-crate like a normal crate. Furthermore, you can also push two half-crates next to each other at the same time if the cell that the second half-crate gets pushed into fulfills the conditions listed in the crate entry.

- `P`: A pit. Your agent cannot walk into a cell with a pit. However, a pit can be filled by pushing objects into it:
  - If you push a crate into an empty pit, it gets filled. You can now treat the cell like an empty cell.
  - If you push a half-crate into an empty pit, it gets half-filled. You cannot walk into a half-filled pit either.
  - If you push a half-crate into a half-filled pit, the pit gets completely filled and you can treat the cell like an empty cell.
  - If you push a crate into a half-filled pit, you permanently block the cell (the crate cannot be removed anymore, you cannot enter the cell or push anything else into the pit).

3 Planners

You should use a PDDL planner for this problem. PDDL planners require two files: a domain file and a problem file. The domain file should describe the rules of the Wumpus world in general and the problem file specifies the details of a particular map. A planner can then find a plan that solves the problem. The plan may use different actions than the expected solution format, so you can convert the planner output to the expected format in a post-processing step.

The assignment repository [AR] describes different planners that you can use and how to run them.

4 What to submit

You should submit

- All your code for solving this assignment.
- A `README.md` file explaining
  i. dependencies (programming language, version, external libraries and how to get them),
  ii. what planner you used (there are differences in what PDDL subset they support),

Maps	Used cell types
`map000.txt` – `map009.txt`	`S ␣ X`
`map010.txt` – `map019.txt`	`S ␣ C X`
`map020.txt` – `map029.txt`	`S ␣ A W X`
`map030.txt` – `map039.txt`	`S ␣ F W X`
`map040.txt` – `map049.txt`	`S ␣ A F W X`
`map050.txt` – `map059.txt`	`S ␣ C H`
`map060.txt` – `map069.txt`	`S ␣ C H P X`
`map070.txt` – `map079.txt`	`S ␣ C F H P W X`

Figure 2: Overview of cell types used in maps.

   iii. how to run your code on different environments,
   iv. the repository structure,
   v. anything else we should know.
- A solution summary (see [SoS] for more details – it should describe the main ideas, not document the code).
- Your domain file(s).
- Solutions to all the example maps in the assignment repository [AR]. Concretely, you should submit for every map `mapXYZ.txt` the problem file `mapXYZ.pddl`, the solution of the PDDL problem `mapXYZ.pddl.soln`, and the actual plan as specified in Section 2.1 listed line by line in a file `mapXYZ-solution.txt`.

5 Points

The total number of points for this assignment is 100. You can get up to 20 points for the quality of the submission (README, evaluation, ...). Furthermore, you will get 1 point for every correctly solved map, which means that you can get up to 80 points for the solutions.

References

[AR] *Repository for Assignment 5: Escape the Wumpus Cave.* URL: `https://gitlab.rrze.fau.de/wrv/AISysProj/ss25/a1.5-escape-wumpus-cave/assignment`.

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

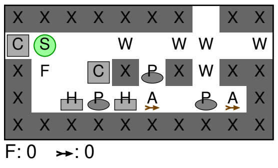
A Solution to the Example Problem

A solution to the example problem from Section 2 would be:

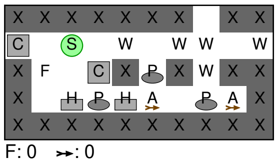
```
walk east
walk east
push south
walk west
walk west
walk south
push east
push east
push east
push east
push east
walk east
walk east
walk west
shoot north
walk north
scare north
walk north
walk east
shoot east
walk east
walk east
```

which could be visualized as

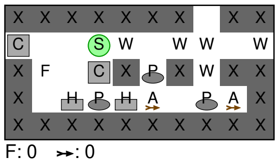
0) Initial state:



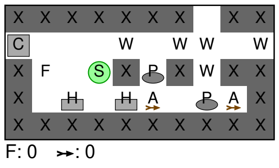
1) `walk east`:



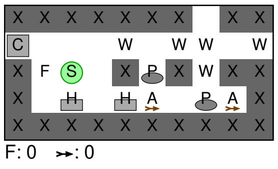
2) `walk east`:



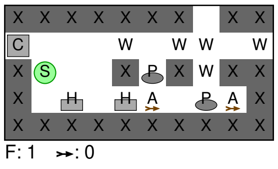
3) `push south`:



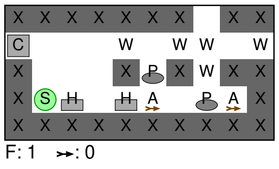
4) `walk west`:



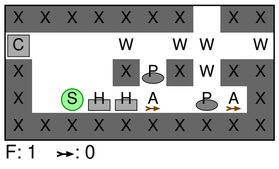
5) `walk west`:



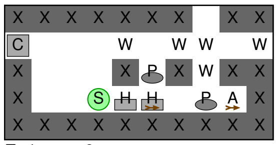
6) `walk south`:



7) `push east`:

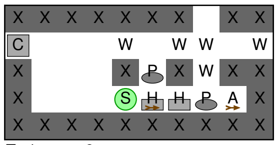


8) `push east:`



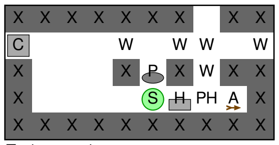
F: 1 ➣: 0

9) `push east:`



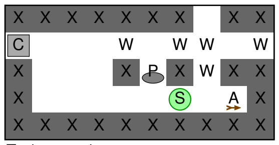
F: 1 ➣: 0

10) `push east:`



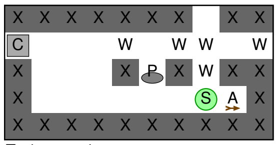
F: 1 ➣: 1

11) `push east:`



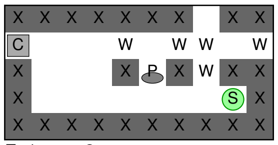
F: 1 ➣: 1

12) `walk east:`



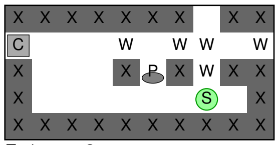
F: 1 ➣: 1

13) `walk east:`



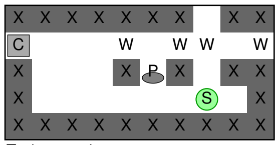
F: 1 ➣: 2

14) `walk west:`



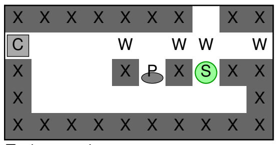
F: 1 ➣: 2

15) `shoot north:`



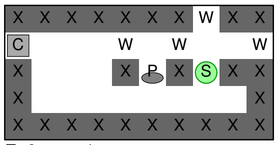
F: 1 ➣: 1

16) `walk north:`



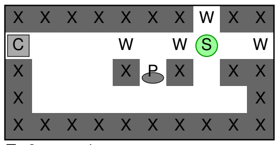
F: 1 ➣: 1

17) `scare north:`



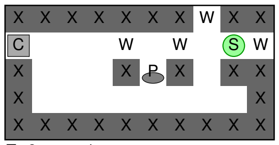
F: 0 ➣: 1

18) `walk north:`



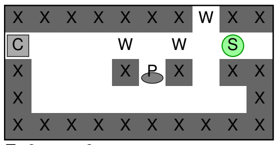
F: 0 ➣: 1

19) `walk east:`



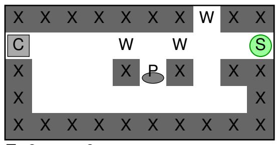
F: 0 ➣: 1

20) `shoot east:`



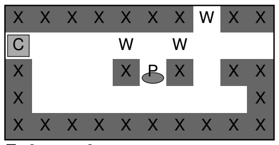
F: 0 ➣: 0

21) `walk east:`



F: 0 ➣: 0

22) `walk east:`



F: 0 ➣: 0