Assignment 6: Find a plan to leave the Wumpus cave

AI-1 Systems Project (Winter Semester 2022/2023)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic:	Planning
Due on:	April 11, 2023 (*early track:* February 21, 2023)
Version from:	January 30, 2023
Author:	Jan Frederik Schaefer

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.

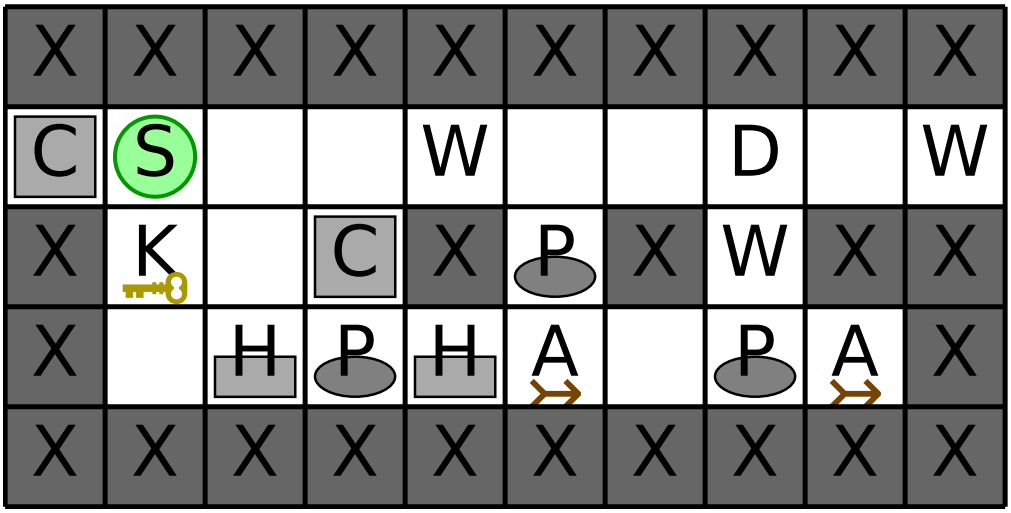


Figure 1: Example map of the Wumpus world.

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:

```
XXXXXXXXXX
CS␣␣W␣␣D␣W
XK␣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.
- `unlock [north|east|south|west]`: Unlock a door. This requires that there is a locked door in the neighboring cell in the indicated direction. You also must have at least one key. Afterwards, you have one key less and the door is unlocked.

Note that these actions do not have to correspond to the actions that you use for the planning itself.

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.
- `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.
- `K`: A key. If you walk into a cell with an key, you pick it up automatically (and have one more key). Afterwards, the cell is empty.
- `D`: A (locked) door. You cannot walk into a cell with a locked door. If you have a key, you can `unlock` (see Section 2.1) the door. After unlocking the door, you can treat the cell like an empty cell. Note that you cannot re-use the key for a different door.
- `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 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).
- `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. How ever, 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.

3 Planners

You should use a PDDL planner for this problem. PDDL planners require two files: a domain file, which we will call `wumpus.pddl`, and a problem file, which we will call `map.pddl`. The domain file should describe the rules of the Wumpus world in general and the problem file specifies the details of a particular map.

You can use the `fast-downward` planner or `pyperplan` (see below). Note that `pyperplan` may not be efficient enough to solve all the problems in a reasonable amount of time. If you want to use a different planner, please ask in the problem channel first.

3.1 `fast-downward` (recommended)

`fast-downward` [FD] is a heavily optimized planner that was quite successful in the international planning competition (IPC). You can build it yourself (which worked very well for me) or run it using docker.

3.1.1 Building and Running `fast-downward` yourself

After cloning the repository, you can build `fast-downward` with

```
python3 build_configs.py
python3 build.py
```

Then you should be able to run it with

```
python3 fast-downward.py --alias lama-first \
    --plan-file /path/for/outputfile \
    /path/to/wumpus.pddl \
    /path/to/map.pddl
```

You can pick other modes than `lama-first`. Note that the `lama-first` mode does not attempt to find an optimal plan (i.e. a plan with minimal cost/a minimal number of actions).

3.1.2 Using Docker

If you have docker installed, you can simply run `fast-downward` with

```
docker run --rm -v "/path/to/my/files:/files" \
    aibasel/downward --alias lama-first \
    --plan-file /files/outputfile \
    /files/wumpus.pddl /files/map.pddl
```

The first line maps the directory `/path/to/my/files` to the docker volume `/files`. That means that `/path/to/my/files/xyz` on your machine will correspond to `/files/xyz`.

3.2 `pyperplan`

`pyperplan` [Alk+20] is implemented in Python and used as a teaching tool. As such, it is not heavily optimized, but it might still be useful (at least to get started). The README file on github provides good instructions for installation and use.

4 What to Submit

You should submit

- All your code and a README explaining how to use it for finding a plan for a map. Your README should also state what planner you used (there are some differences in the PDDL subset they support).
- A brief summary of how you solved the problem either as a PDF file ($\approx$ 1 page) or as part of your `README.md`.
- 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 Random Tips

- Don't try to support all cell types at once. It might be easier to start with the easier maps to get a feeling for PDDL. Figure 2 provides an overview of the maps.

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

Figure 2: Overview of cell types used in maps.

- The actions in your PDDL file don't have to directly correspond to the "solution" actions described in Section 2.1. The only condition is that the "solution" actions can easily be generated from the PDDL actions.
- Be aware that planners usually support only a subset of the PDDL format.
- Make your own minimal test maps to debug your implementation.
- Have fun!

6 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, explanation, ...). 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

[Alk+20] Yusra Alkhazraji et al. *Pyperplan*. `https://doi.org/10.5281/zenodo.3700819`. 2020. DOI: `10.5281/zenodo.3700819`. URL: `https://doi.org/10.5281/zenodo.3700819`.

[AR] *Repository for Assignment 6: Find a plan to leave the Wumpus cave*. URL: `https://gitlab.rrze.fau.de/wrv/AISysProj/ws2223/a6-escape-wumpus-world/assignment`.

[FD] *Fast Downward*. URL: https://www.fast-downward.org/ (visited on 02/15/2022).

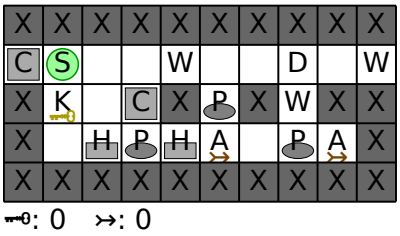
A Solution to the Example Problem

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

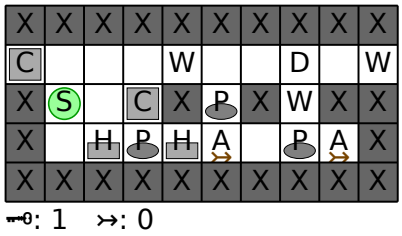
```
walk south
walk east
walk north
walk east
push south
walk west
walk west
walk south
push east
push east
push east
push east
push east
walk east
walk east
walk west
shoot north
walk north
unlock north
walk north
walk east
shoot east
walk east
walk east
```

which could be visualized as

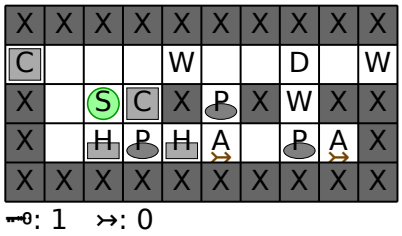
0) Initial state:



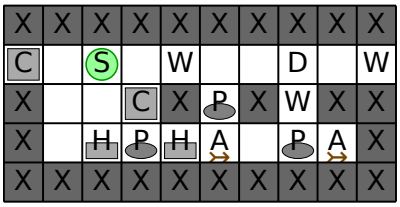
1) After `walk south`:



2) After `walk east`:

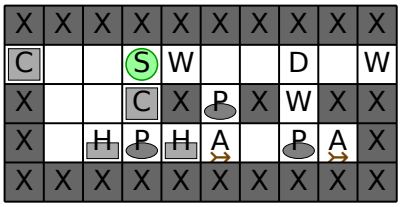


3) After `walk north`:



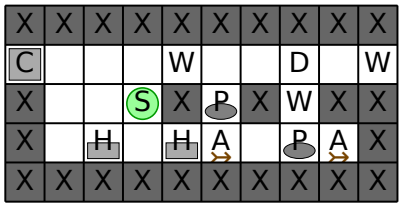
⊸: 1 ↣: 0

4) After `walk east`:



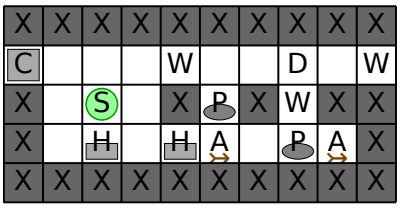
⊸: 1 ↣: 0

5) After `push south`:



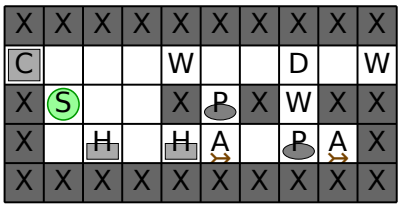
⊸: 1 ↣: 0

6) After `walk west`:



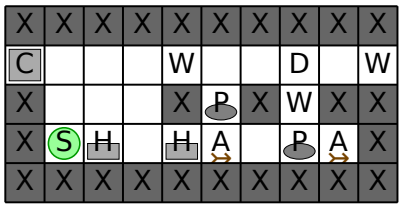
⊸: 1 ↣: 0

7) After `walk west`:



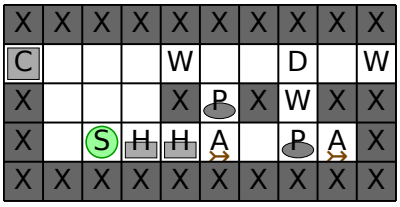
⊸: 1 ↣: 0

8) After `walk south`:



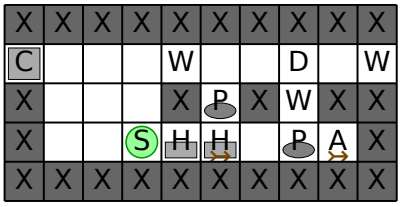
⊸: 1 ↣: 0

9) After `push east`:



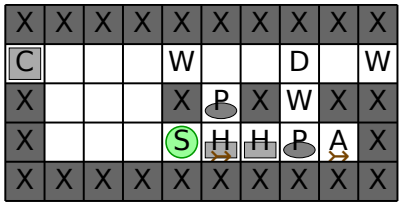
⊸: 1 ↣: 0

10) After `push east`:



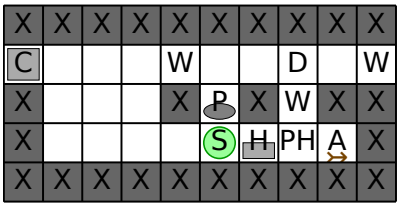
⊸: 1 ↣: 0

11) After `push east`:



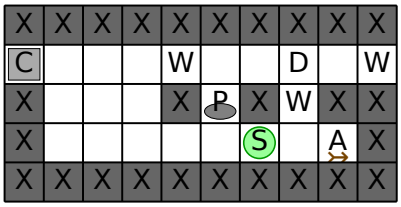
⊸: 1 ↣: 0

12) After `push east`:



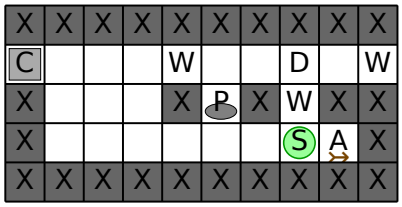
⊸: 1 ↣: 1

13) After `push east`:



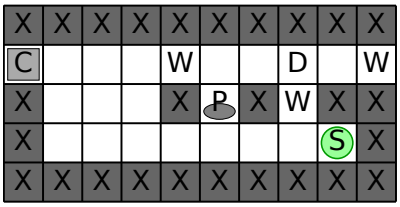
⊸: 1 ↣: 1

14) After `walk east`:



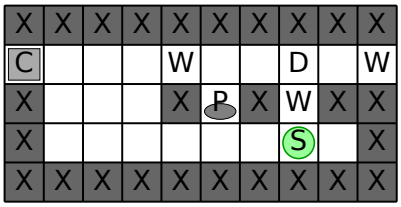
⊸: 1 ↣: 1

15) After `walk east`:



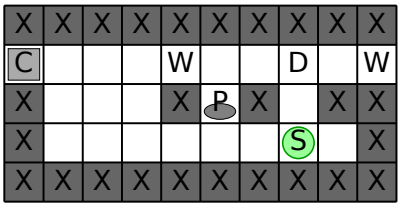
⊸: 1 ↣: 2

16) After `walk west`:



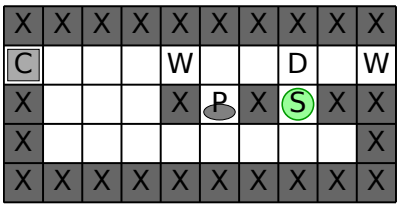
⊸: 1 ↣: 2

17) After `shoot north`:



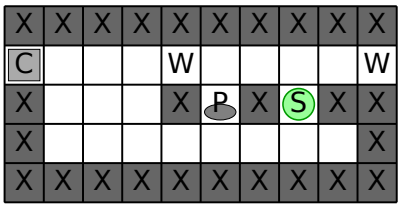
⊸: 1 ↣: 1

18) After `walk north`:



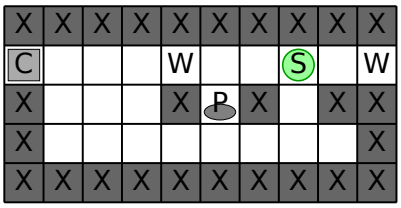
⊸: 1 ↣: 1

19) After `unlock north`:



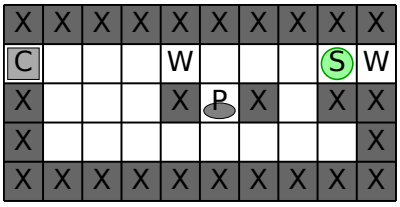
⊸: 0 ↣: 1

20) After `walk north`:



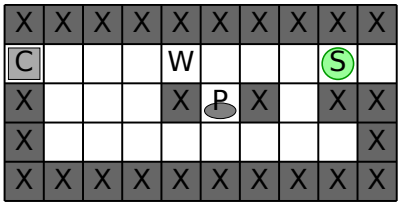
⊸: 0 ↣: 1

21) After `walk east`:



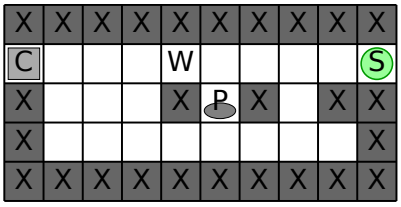
⊸: 0 ↣: 1

22) After `shoot east`:



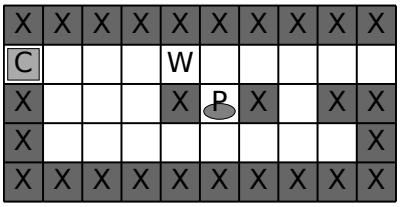
⊸: 0 ↣: 0

23) After `walk east`:



⊸: 0 ↣: 0

24) After `walk east`:



⊸: 0 ↣: 0