— GUIDE —

Assignment 0 (Warm-Up, Variant A): Find Back to the Wumpus Cave

AI-2 Systems Project (Winter Semester 2024/2025)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

This document is intended to help you solve the assignment "Assignment 0 (Warm-Up, Variant A): Find Back to 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 know how to get started, checkout the "How to get started"-section (Section 3).
3. Start with the easier environments (environment 1 is easier than environment 2, etc.).
4. To compute the correct probabilities: Make sure you understand conditional probabilities. Pick suitable random variables to express everything you know. Then you should be able to compute the probabilities using the rules discussed in the lecture. It might be instructive to compute a small example by hand.
5. You can already get points if you hard-code one action sequence and correctly compute the success chance and expected time for it.
6. You do not have to find optimal solutions to get full points.

2 A more formal perspective on the assignment

We want you to learn how to take a rather informal problem description and link it to concepts and solution strategies that you learned in your studies. That is not easy and we want to show you one way to think about the assignment in this guide:

1. The Wumpus makes different observations O about the environment, which we can think of as random variables.
2. We can use that to compute a probability distribution $P(C \mid O)$ where C is a random variable for the position of the Wumpus.
3. We now make a plan π to reach a cave with the Wumpus.
4. We denote the time it takes to reach a cave with a plan π from cell c as $t_\pi(c)$ (it is undefined if the plan does not reach the cave from c).
5. The success chance is $P(\pi \text{ succeeds} \mid O) = \sum_{c \text{ for which } \pi \text{ succeeds}} P(C = c \mid O)$. A plan succeeds for a cell c if it reaches the cave from c within the time limit.
6. The expected time is the expected value $E[t_\pi \mid \pi \text{ succeeds}, O] = \sum_{c \text{ for which } \pi \text{ is sucessful}} t_\pi(c) \cdot P(C = c \mid O)$.
7. Your agent should find a plan for which these two values are small. Concretely, you should minimize the rating $r(\pi)$, which combines these values:

   $$r(\pi) := P(\pi \text{ succeeds} \mid O) \cdot E[t_\pi \mid \pi \text{ succeeds}, O] + (1 - P(\pi \text{ succeeds} \mid O)) \cdot \texttt{max-time}$$

   which means that this is an optimization problem where $r(\pi)$ is the cost of a solution π. We ideally want to find $\arg\min_\pi r(\pi)$, but for this assignment we do not need an optimal solution.

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 Getting started

Let us assume you get the following query:

```
{
    "initial-equipment": [],
    "map": "CMM\nCCM\nMWM",
    "max-time": 5,
    "observations": {"current-cell": "M"}
}
```

1. What is a good representation of the map in your code?
2. Let K be the coordinates of the `W` cell according to your representation. What cell is north of it? What cell is east of it?
3. What is the probability that the Wumpus is in the top-left `M` cell?

4. For each cell, what is the probability that the Wumpus is in it?
5. If we pretend that the observation is `C`, what is the probability that the Wumpus is in each cell?
6. Assume the plan `GO south`, `GO east`. How long does it take to reach the Wumpus cave if you start in the center cell?
7. Using the same plan, how long does it take to reach the Wumpus cave from each of the other `M` cells?
8. What is the success chance of the plan?
9. What is the expected time it takes to reach the cave?

3.3 Testing your approach on environment 1

Instead of the JSON above, you can now switch to using the JSON provided by the server. If you use Python, you can simply use the example implementation provided in the assignment repository [AR]. In the beginning, you can simply hard-code one plan (e.g. `GO south`, `GO east`) and send that plan along with the success chance and expected time you computed for it. If the results are wrong, you should take a look at the AISysProj server to get some information that could help you with debugging.

Once everything works, you can try to improve the results by experimenting different plans. Here are some options:

1. Generate all possible plans of a certain length and use the best one.
2. Generate n random plans and use the best one.
3. Use a search algorithm (e.g. greedy search, which requires you to come up with a heuristic).
4. Use a combination of the options mentioned above.

There many ways to solve the problem and you can get creative.

3.4 Continuing

For the next environments, your code will need increasingly many features. Whenever a new feature is required, you can make yourself a small example to test your ideas (similar to the steps described in Section 3.2). If you get stuck, you can ask in the assignment room on Matrix or during office hours.