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

AI-2 Systems Project (Winter Semester 2023/2024)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic:	Basic probabilities
Due on:	March 15, 2024
Version from:	January 12, 2024
Author:	Jan Frederik Schaefer
Important notes:	To be solved individually
	Earlier deadline for first results (see your repository for submitting solution)

1 Task summary

The Wumpus, a mythical creature that repeatedly shows up in the AI systems project, has left its cave and got lost. Your task is to make a travel plan that will quickly get it back to the cave. Your implementation will get different problem instances from the AISysProj server and has to send back its travel plan. That lets the server evaluate your agent and can help you with debugging. The assignment repository has a script that already implements the server interaction for you.

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 working with basic concepts from probability theory,
4. get to know the AISysProj setup and workflows.

Prerequisites and useful methods

1. The basics of probability theory: condition probabilities, independence, expected values, Bayes' rule, ...

2 Navigating the Wumpus world

The Wumpus got lost. Your task is to implement an AI agent that makes a plan for it to get back to the Wumpus cave. We have a server to help evaluate your agent. It will send you a map along with some other information and your agent will have to respond with a plan to find back to the cave.[1] The server has different environments, corresponding to different difficulty levels. For the easier environments, not everything mentioned in this section is relevant.

2.1 Maps

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

```
CWBBB
RRRBB
WRRRR
BBRWB
MWRRR
```

We have the following cell types:

- `M`: Meadows.
- `B`: Trees (broad-leaf).
- `C`: Trees (coniferous).
- `R`: Rocks.
- `W`: Wumpus cave entrance. Your goal is to reach one of those.

We assume that everything outside the map are meadows (`M`).

2.2 Observations

We do not know where the Wumpus is (after all, the Wumpus got lost). Therefore, we assume that, a priori, the Wumpus is equally likely in any of the cells on the map. However, the Wumpus can make some basic observations to narrow down the possibilities:

[1]The server part may seem daunting, but we provide an example implementation in Python. All you have to then do is implement a function.

- It observes what type of cell it currently is in (`M` or `B` or ...). Unfortunately, the Wumpus has bad vision and misidentifies trees 20 % of the time (i.e. if the Wumpus is in a `B` cell, there is a 20 % chance that it instead thinks it is a `C` cell and vice versa).
- In some environments, it also remembers what cell type it observed in the cell that is west of the current cell. However, there is again a chance of misidentification.

2.3 Plans

Your task is to make a plan for the Wumpus. A plan is a sequence of actions. You have the following actions available:

- `GO [north|east|south|west]`: The Wumpus should go one cell in the indicated direction. Note that the world is not limited to the map and the Wumpus can go to cells outside the map (recall that every cell outside the map is a meadow).
- `DROP climbing-gear`: In some environments, the Wumpus carries climbing gear. This action tells the Wumpus to drop it (which allows it to be faster on most cell types).

Ideally, the Wumpus reaches one of the Wumpus cave entrances while following the plan. How long it takes for the Wumpus to reach the entrance depends on the cell types. Usually, crossing a cell takes 1 hour. With climbing gear, it takes 1.2 hours. However, crossing a cell with rocks (`R`) takes 4 hours, but with climbing gear it only takes 2 hours. We assume that the Wumpus starts in the center of a cell and only has to reach the edge of a cell with a cave entrance. Let us consider the following map as an example:

```
CR
MW
```

If the Wumpus is initially the `C` cell and follows the plan `GO east, GO south`, then it will have to cross $\frac{1}{2}$ of the `C` (as it starts in the center) and $\frac{2}{2}$ of the `R` cell ($\frac{1}{2}$ to reach its center and $\frac{1}{2}$ to reach the edge of the `W` cell). In total, it will therefore take $0.5 + 4 = 4.5$ hours to reach the cave, assuming that the Wumpus has no climbing gear.

To make the assignment easier, we have a maximum time M for each environment. If reaching a cave entrance takes longer than M (or the plan finishes before an entrance has been reached), then we instead assume that the time is M.

3 Evaluation on the server

You should evaluate your implementation with the AISysProj server: `https://aisysproj.kwarc.info/`. You will get JSON requests from the server and have to respond with your plan. For example, a request could have the following content:

```
{
    "initial-equipment": ["climbing-gear"],
    "map": "CWBBB\nRRRBB\nWRRRR\nBBRWB\nMWRRR",
    "max-time": 6,
    "observations": {"cell-west": "R", "current-cell": "B"}
}
```

Your agent should then respond with a plan and the expected time until a Wumpus cave is reached:

```
{"actions": ["GO␣south", "GO␣south", "GO␣east"], "expected-time": 3.65}
```

The server remembers your last 1000 responses and computes the average expected time to rate your agent (the lower, the better). If you submit the wrong `expected-time`, the server will instead use `max-time`. However, afterwards you can see the problem on the server with the correct expected time and some other information that could help you with debugging.

The server has different environments that you can use (some are easier than others). Your repository for this assignment will contain one configuration file for each environment, which contains, among other things, your agent name and a password. The assignment repository [AR] contains an example implementation in Python along with instructions on how to use it, so that you do not have to worry about the technical aspects of communicating with the server. If you want to use a different programming language, you can find the protocol specification in Appendix B. We are happy to help you with implementing it if you are prepared to donate your code for others.

4 What to submit

Your solution should be pushed to your git repository for this assignment. For this warm-up assignment, we have an early deadline. At this deadline, the repository should contain all the code you have so far. It should be enough to get at least 1 point in one of the environments on the server. Otherwise, we might assume that you are not actually interested in the project and give your spot to someone else.

Your grade will be based on your final submission (deadline: March 15, 2024). Concretely, your repository should contain:

1. all your code for solving this assignment,
2. a `README.md` file explaining how to run your code (including e.g. dependencies that have to be installed),
3. a brief evaluation either as a PDF file ($\approx \frac{1}{2}$ page, but it may be longer) or as part of your `README.md`, describing your approach for solving the problem and what worked well/did not work well, ($\approx \frac{1}{2}$ page) or as part of your `README.md`.

5 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 "Guide for Getting Started" (Appendix A).
3. Start with the easier environments (environment 1 is easier than environment 2 etc., see also Section 6).
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 expected time for it.
6. You do not have to find optimal solutions to get full points.

6 Points and environments

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 can get up to 80 points for the performance of your agent according to the server. Each environment allows you to get a certain numbers of points if your agent performs well enough. When grading, we will only consider the environment in which you would get most points (we do not add up results from different environments). That means that you do not have to run your code on every environment and you can get full points if you only run it on the most difficult one where you can get up to 80 points.

Note: It is okay if your agent is a bit "lucky" and gets a slightly higher rating than it usually does. We will use the best rating on the server for your grade, assuming that we can reproduce a similar performance (i.e. if you get a 5.39 rating on the server and we get a 5.48 rating that is okay – but if you get a 5.39 rating on the server and we get a 6.79 rating when testing it, we might ask some questions).

Below is a table that summarizes the properties of the different environments and how many points you can get for each. For each environment, you have a config file.

Config file	**Map size**	**Cells**	**Max. time**	**Equipment**	**Observations**	**Points**
env-1.json	5×5	`C M`	2	–	current-cell	0 if rating > 1.9
						20 if rating ≤ 1.9
						30 if rating ≤ 1.5
env-2.json	5×5	`C M`	5	–	current-cell	0 if rating > 4.0
						30 if rating ≤ 4.0
						40 if rating ≤ 3.0
env-3.json	5×5	`B C M`	5	–	current-cell	0 if rating > 4.0
						40 if rating ≤ 4.0
						50 if rating ≤ 3.0
env-4.json	5×5	`B C M R`	6	climbing-gear	current-cell	0 if rating > 5.0
						45 if rating ≤ 5.0
						55 if rating ≤ 4.0
env-5.json	5×5	`B C M R`	6	climbing-gear	current-cell, cell-west	0 if rating > 5.0
						50 if rating ≤ 5.0
						60 if rating ≤ 4.0
						70 if rating ≤ 3.0
env-6.json	15×15	`B C M R`	24	climbing-gear	current-cell, cell-west	0 if rating > 20.0
						50 if rating ≤ 20.0
						75 if rating ≤ 15.0
						80 if rating ≤ 13.0

References

[AR] *Repository for Assignment 0 (Warm-Up, Variant B): Find Back to the Wumpus Cave.* URL: `https://gitlab.rrze.fau.de/wrv/AISysProj/ws2324/a2.0.b-find-wumpus-cave/assignment`.

A Guide for getting started (optional)

The "guide for getting started" 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 guide useful (then we might provide it in future assignments as well)! We also appreciate if you can suggest improvements.

A.1 How to use this guide?

This guide 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 cells are adjacent to cell (3, 5)?" 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 `Cell` 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 cells in general.

Usually, it is a good idea to test your implementations with more examples to make sure that you cover all the cases. This guide tends to intentionally skip over special cases so that you will discover them at some point yourself, which we consider a valuable teaching moment.

A.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 expected time it takes to reach the cave?

A.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 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 is not a single way to solve the problem and you can get creative.

A.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 Appendix A.2). If you get stuck, you can ask in the assignment room on Matrix or during office hours.

A.5 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 $P(C|O)$ where C is a random variable for the position of the Wumpus.
3. We compute the time $t_\pi(c)$ it takes to reach a cave from a starting position c with a plan π (re-call that there is a maximum time).
4. The expected time it takes to reach a cave with a plan π is now the expected value $E[t_\pi(C)] = \sum_c t_\pi(c) \cdot P(C = c|O)$.
5. Your agent should find a plan π where $E[t_\pi(C)]$ is small, which means that this is an optimization problem where $E[t_\pi(C)]$ is the cost of a solution π. We ideally want to find $\arg\min_\pi E[t_\pi(C)]$, but for this assignment we do not need an optimal solution.

B The AISysProj Server Protocol

For some assignments, agents are tested by interacting with the AISysProj server. This appendix describes the protocol and is **only relevant if you want to implement the server protocol yourself**.

The protocol is rather minimal: You send an HTTP request to the server with your credentials and actions, and the server responds either with an error message or with a list of action requests that you should respond to in the next request. For the first request, you

```
// First request:
{"agent": "MyAgent", "pwd": "r7iUM8o1NLbFdkI2WmBDldsYHD3wLwUQAKoG_2_xBcE", "actions": []}

// First response
{"errors": [], "messages": [], "action-requests": [
   {"run": "40#1", "percept": ...},
   {"run": "7#3", "percept": ...}]}

// Second request
{"agent": "MyAgent", "pwd": "r7iUM8o1NLbFdkI2WmBDldsYHD3wLwUQAKoG_2_xBcE",
   "actions": [{"run": "40#1", "action": ...},
               {"run": "7#3", "action": ...}]}
```

Figure 1: Example interaction with the server.

will have to send an empty list of actions to get action requests from the server (see Figure 1 for an example).

Configuration files The server details and your credentials are stored in configuration files. Usually, we will generate configuration files for you and commit them to your repository. They are JSON files that contain the following fields:

- `url`: the server URL,
- `env`: the environment,
- `agent`: the agent's name, and
- `pwd`: the agent's password.

The request You should send a `PUT` request to `[url]/act/[env]`, where `[url]` and `[env]` are provided by the configuration file. The request body should contain a JSON object with the fields:

- `agent`: the agent's name (from the configuration file).
- `pwd`: the agent's password (from the configuration file).
- `single_request`: `true` or `false`, indicating if a only a single action request should be sent (can be helpful for debugging).
- `actions`: the actions the agent wants to do as a list. Each action is represented as an object with two fields: `run` is an identifier of the action request (provided in the server responses) and `action` is the action the agent wants to do – the value format depends

on the assignment.

The response If the request was accepted, you will receive a JSON response with the fields:

- `action-requests`: a list of action requests that you should send actions for in the next request. Each action request is an object with two fields: `run` is an identifier so that your action can be linked to the request and `percepts` describes what you know about the current state (e.g. the position in a game).
- `errors`: a list of error messages (e.g. if your move was invalid).
- `messages`: a list of other messages.

Error response In case of an error (e.g. invalid credentials), you get a JSON response with the fields:

- `errorcode`: The HTTP error code.
- `errorname`: The name of the error.
- `description`: A more detailed description of the error.