Assignment 3: Wumpus Quest

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

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic:	Markov Decision Processes
Due on:	February 12, 2024
Version from:	November 23, 2023
Author:	Jan Frederik Schaefer

1 Task summary

Your goal is to collect gold in the Wumpus cave and bring it back safely. There are many dangers in the cave, but, fortunately, you have a map that can help you decide on the best course of action. You can test and evaluate your implementation on the server[1].

Didactic objectives

1. Gain experience modelling a complex problem as a Markov Decision Process.

Prerequisites and useful methods

1. Basics of working with probabilities,
2. Markov Decision Processes (and value iteration) as discussed in the lecture.

2 Detailed problem description

You have a 14×12 map of the Wumpus cave (Figure 1 shows an example). Your goal is to guide the agent to collect the gold and leave the cave. Depending on the environment, there are different obstacles that might kill your agent. Before entering the cave, your agent can train, which means that you can allocate a certain number of skill points, which will e.g. improve its chances when fighting the Wumpus. Your agent only has enough time for 100 actions until it runs out of food (in practice, that is plenty of time and mostly serves as a mechanism to stop runs where your agent somehow got stuck).

[1] `https://aisysproj.kwarc.info`

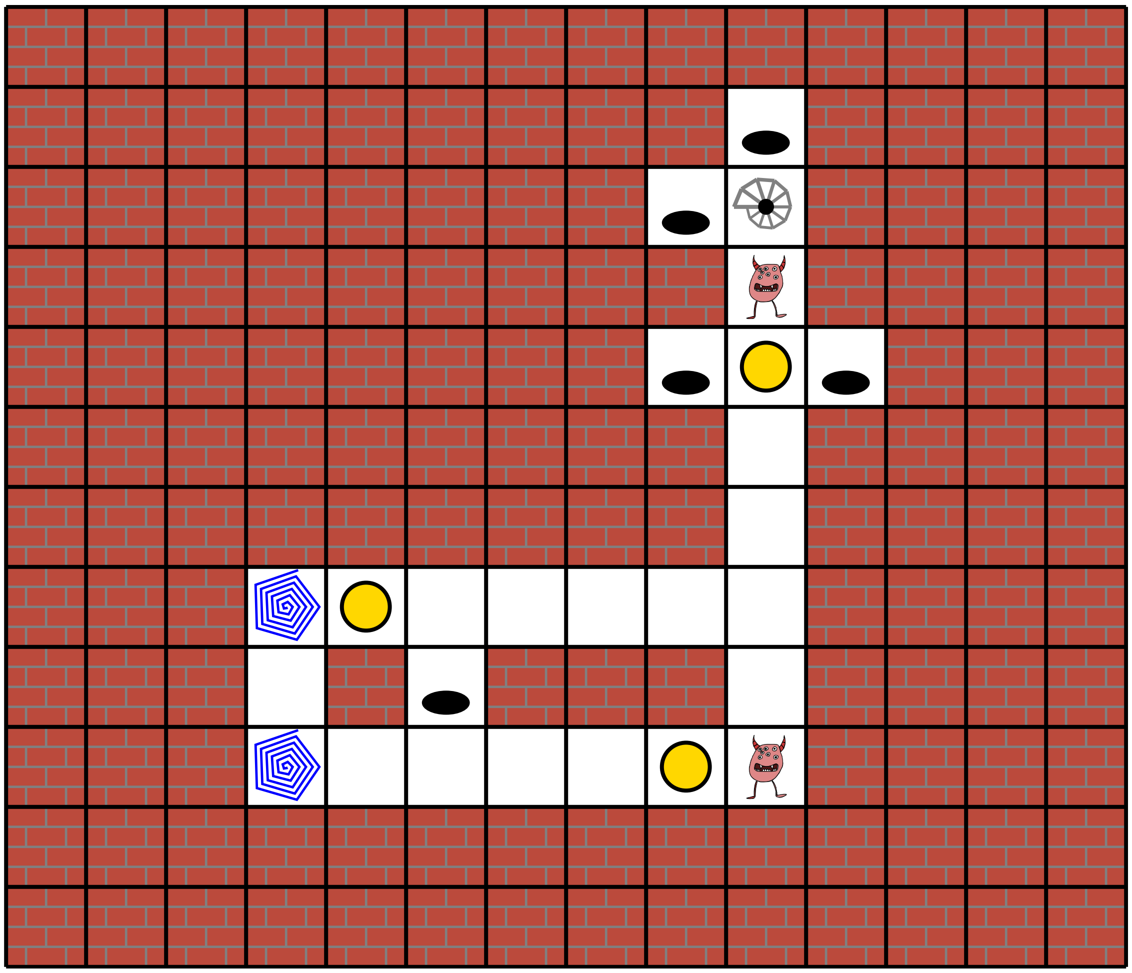
Figure 1: The Wumpus cave.

Your agent will be rated based on how much gold it carries out of the cave (0 if it dies). Note that your agent starts with 1 gold, so, sometimes, the best strategy might be to take no risk and exit the cave right away.

In the following subsections, we will explore what actions your agent can perform and what obstacles it faces in the cave.

2.1 Maps

The map is represented as a string, where each line corresponds to a row of the map and each character describes the properties of a cell. For example, the map shown in Figure 1 would have the following string representation:

```
XXXXXXXXXXXXXX
XXXXXXXXXPXXXX
XXXXXXXXPSXXXX
XXXXXXXXXWXXXX
XXXXXXXXPGPXXX
XXXXXXXXX␣XXXX
XXXXXXXXX␣XXXX
XXXTG␣␣␣␣␣XXXX
XXX␣XPXXX␣XXXX
XXXT␣␣␣␣GWXXXX
XXXXXXXXXXXXXX
XXXXXXXXXXXXXX
```

The letters have the following meanings:

- A space: An empty square.
- **S**: The stairs (your agent will start there, and it is the only exit).
- **G**: Gold.
- **P**: A pit.
- **W**: A Wumpus.
- **T**: A teleporter.

Section 2.4 will describe some of the obstacles in more detail.

2.2 Skill points

Depending on the environment, you have a certain number of skill points that you can allocate to two different skills: navigation and fighting. For example, if you can distribute 10 skill points, you could allocate 6 points for navigation and the remaining 4 points for fighting. If you need to use a skill with n allocated skill points, your agent will role an n-sided fair die. For $n = 0$, your agent will always fail. Otherwise, it will depend on whether the result is sufficiently high.

For example, if your agent has to fight the Wumpus, it will need to use the fighting skill. With 4 allocated skill points, it would roll a 4-sided die. To win against the Wumpus, the result has to be at least 3. In this case, the probability of winning is therefore 50 % (both a 3 and a 4 will win).

2.3 Actions

Depending on where you are, you can perform one of the following actions:

- **NORTH/EAST/SOUTH/WEST**: Walk one step in the direction (you cannot walk in the direction of a wall, and you also cannot walk away from the Wumpus without fighting – the server would reject such actions). It is dark in the cave, so sometimes the agent walks in the wrong direction. If the other three squares (behind/left/right of you) are all walls, the agent will definitely walk in the requested direction. Otherwise, the probability of walking in the requested direction depends on your navigation skill. If you roll a 2 or higher, you will walk in the requested direction. Otherwise, you will walk to one of the other free squares (behind/left/right of you). Each square is equally likely, but your agent will never walk into a wall.
- **EXIT**: Exit the cave (only possible if you are at the stairs).
- **FIGHT**: If you are on a square with a Wumpus, you have to fight it. To win, you have to use the fighting skill and get a score of at least 3.
- **TELEPORT**: If you are on a square with a teleporter, you can use it to teleport. You will then be teleported to another teleporter. You are equally likely to be teleported to each of the other teleporters. You will never be teleported to the square you are teleporting from (and there is never just one teleporter). Using teleporters is risky, so you will need to use your navigation skill and get a score of at least 2 – otherwise your agent will not survive the teleportation.

2.4 Obstacles

Depending on the environment, your agent can encounter the following deadly obstacles:

- **Pit**: If your agent falls into a pit, it will not survive.
- **Wumpus**: If your agent meets the Wumpus, it will have to fight (see Section 2.3). If your agent wins, the Wumpus will not bother you anymore, but if it looses, your agent dies.

3 Competing on the server

You should test and evaluate your agent by competing on the server. The rating of your agent is the average amount of gold that your agent carried out of the cave in 1000 consecutive runs. Your agent can communicate with the server via HTTP requests. A Python implementation of the protocol is provided in the assignment repository [AR]. The details of the protocol are described in Appendix A, but you will only need them if you want to create your own implementation (e.g. in a different programming language).

3.1 Action requests

The server will send you action requests, which contain an identifier for the request and a JSON object describing your current adventure. Concretely, it has the following fields:

- **"map"**: The map of the cave.
- **"history"**: The actions you have performed so far and what the outcome was. The "outcome" fields are the most relevant ones as they help you re-construct the current state. You can ignore the "expl" fields.
- **"skill-points"**: Your skill point allocation (if you have already allocated them).
- **"free-skill-points"**: The number of skill points you can allocate in this environment (if you haven't allocated them already).

Here is an example (apologies for the poor line breaking):

```
{
  "map": "XXXXXXXXXXXXXXX\nXXXXXXXXXPXXXXX\nXXXXXXXXPSXXXXX\
      nXXXXXXXXXWXXXXX\nXXXXXXXXPGPXXXX\nXXXXXXXXXX XXXX\nXXXXXXXXXX
       XXXX\nXXXTG XXXX\nXXX XPXXX XXXX\nXXXT GWXXXX\
      nXXXXXXXXXXXXXXX\nXXXXXXXXXXXXXXX",
  "history": [
    {"action": {"navigation": 12,"fighting": 8},"outcome": {"navigation": 12,"fighting": 8}},
    {"action": "SOUTH","outcome": {"position": [9,3]}},
    {"action": "FIGHT","outcome": {"killed-wumpus-at": [9,3],"expl": "result: 4"}},
    {"action": "SOUTH","outcome": {"position": [9,4],"collected-gold-at": [9,4],"current-
        amount-of-gold": 2,"expl": ""}},
    {"action": "SOUTH","outcome": {"position": [9,5],"expl": ""}},
    {"action": "SOUTH","outcome": {"position": [9,6],"expl": ""}}
  ],
  "skill-points": {
    "navigation": 12,
    "fighting": 8
  }
}
```

3.2 Sending Actions

Your agent should respond to an action request by sending an action. The first action should be an allocation of skill points in the form of a JSON object (e.g. `{"navigation": 12, "fighting": 8}`).

For the remaining actions, you should send the desired action as a string (Section 2.3 lists the available actions).

4 What to submit

Your solution should be submitted to your team's repository. It should contain:

1. all your code,
2. a README.md file explaining how to run your code to compete on the server (including how to install dependencies),
3. a brief summary of how you solved the problem either as a PDF file (≈ 1 page) or as part of your README.md.

5 A few tips

1. Consider using value iteration.

2. Model the rewards in a way that the utilities roughly correspond to the expected amount of gold that you will carry out of the cave. This way, maximizing the utility corresponds to maximizing the expected amount of gold, i.e. maximizing the rating of your agent. You might have to deviate from this a bit, but it should be a good starting point.
3. It can help with debugging if you can somehow look at the computed utilities/policy.
4. Use Markov Decision Processes for the first environment (even though it can be solved without) to get some experience before moving on to the harder environments.
5. Code optimizations:
    - If you cache the utilities/policy, you do not have to re-compute them for every action.
    - The number of possible states is relatively large – try to avoid making it much larger than necessary.
    - Value iteration (or similar algorithms) might be much faster if you once create an efficient representation of the state space and the transition model, and then only use that representation. For example, you could assign an integer to each state.
    - As a typical run involves many actions and the delays from the interaction with the server may become problematic. To mitigate this, you can have several runs in parallel. The server then sends you multiple action requests within a single HTTP request (and expects that many responses). If you use the example client, you can easily switch to that behaviour.
6. The server interaction is a bit tricky. Please reach out if you face any difficulties or have ideas how it could be improved.

6 Points

You can get up to 80 points for the strength of your agent according to the server (assuming it is reproducible). Assuming you have at least a partial solution, you can additionally get up to 20 points for the quality of the submission (README, explanation, ...). The maximum number of points is therefore 100. If the grading scheme doesn't seem to work well, we might adjust it later on (likely in your favor).

Figure 2 summarizes how many points you can get for each environment according to the rating on the server. When grading, we will only consider the environment in which you would get the most points.

Config	**Obstacles**	**Skill points**	**Points**
ws2324-wquest-1.json		6	$\begin{cases} 0 & \text{if rating } < 2.0 \\ 10 & \text{if rating } \geq 2.0 \\ 20 & \text{if rating } \geq 3.0 \end{cases}$
ws2324-wquest-2.json	P	10	$\begin{cases} 0 & \text{if rating } < 1.5 \\ 30 & \text{if rating } \geq 1.5 \\ 45 & \text{if rating } \geq 1.9 \\ 60 & \text{if rating } \geq 2.3 \end{cases}$
ws2324-wquest-3.json	P	10	$\begin{cases} 0 & \text{if rating } < 1.5 \\ 35 & \text{if rating } \geq 1.5 \\ 50 & \text{if rating } \geq 1.9 \\ 65 & \text{if rating } \geq 2.3 \end{cases}$
ws2324-wquest-4.json	P, W	20	$\begin{cases} 0 & \text{if rating } < 1.5 \\ 40 & \text{if rating } \geq 1.5 \\ 55 & \text{if rating } \geq 1.8 \\ 70 & \text{if rating } \geq 2.0 \\ 75 & \text{if rating } \geq 2.2 \\ 80 & \text{if rating } \geq 2.45 \end{cases}$

Figure 2: Number of points that you can get.

References

[AR] *Repository for Assignment 3: Wumpus Quest.* URL: `https://gitlab.rrze.fau.de/wrv/AISysProj/ws2324/a2.3-wumpus-quest/assignment`.

```
// 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 3: Example interaction with the server.

A 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 will have to send an empty list of actions to get action requests from the server (see Figure 3 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.