Assignment 3: Wumpus Quest

AI-2 Systems Project (Summer Semester 2023)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic:	Markov Decision Processes
Due on:	September 1, 2023
Version from:	August 11, 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`

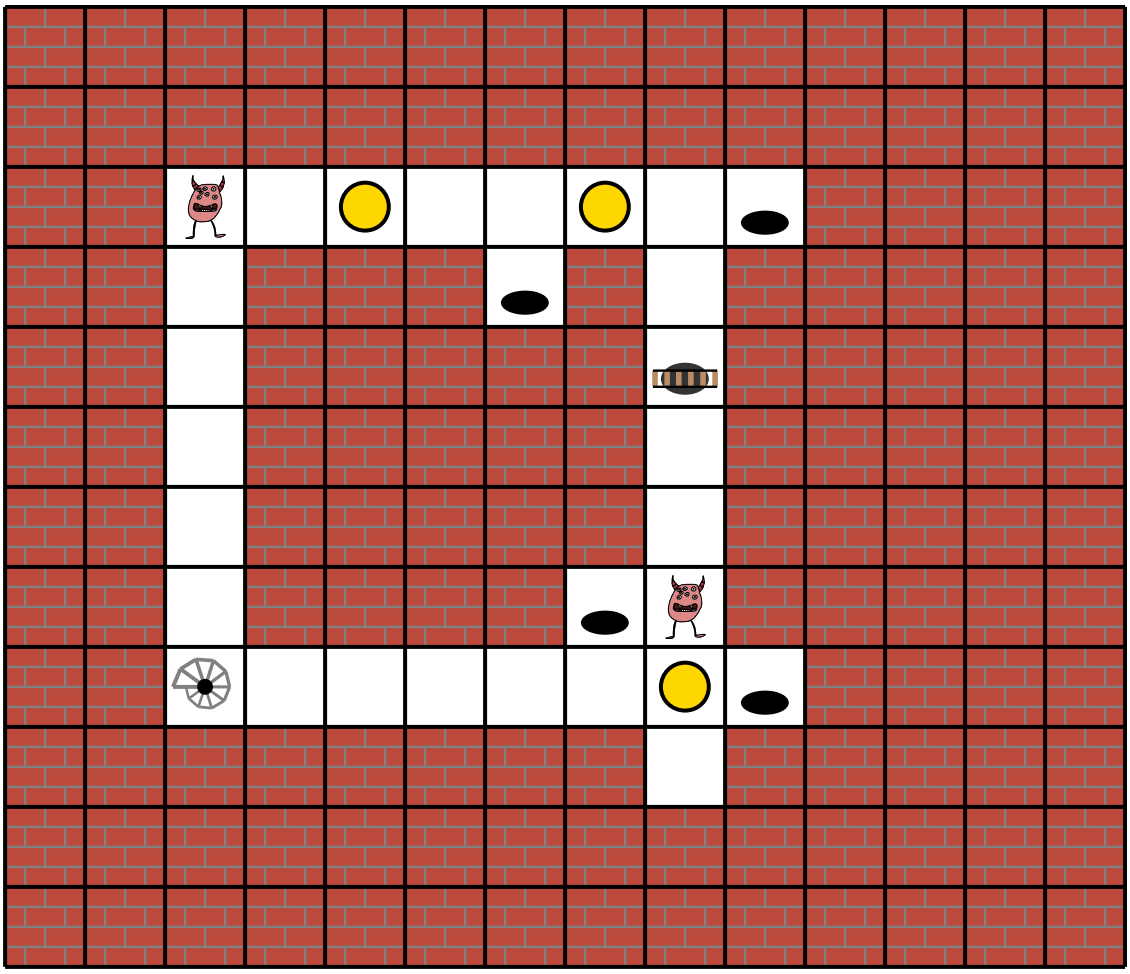

Figure 1: The Wumpus cave.

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
XXXXXXXXXXXXXX
XXW␣G␣␣G␣PXXXX
XX␣XXXPX␣XXXXX
XX␣XXXXXBXXXXX
XX␣XXXXX␣XXXXX
XX␣XXXXX␣XXXXX
XX␣XXXXPWXXXXX
XXS␣␣␣␣␣GPXXXX
XXXXXXXX␣XXXXX
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.
- **B**: A bridge.

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: agility and fighting. For example, if you can distribute 8 skill points, you could allocate 3 points for agility and the remaining 5 points for fighting. If you need to use a skill, your agent will role as many dice as it has allocated to the relevant skill. Your agent then gets a **score**, which is the sum of the three best results. If the score is sufficiently high, your agent succeeds – otherwise it fails.

For example, if your agent has to fight the Wumpus, it will need to use the fighting skill. With 5 allocated skill points, it would role 5 dice. Let us say the result is ⚅⚅⚀⚄⚂. Then the three best results are ⚅, ⚅ and ⚄, which gives us a sum of 16. To win against the Wumpus, the sum has to be at least 13, so, in this case, your agent would win against the Wumpus.

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 there is no wall to the left, there is a 10 % chance that it will walk there instead. Similarly, there is a 10 % chance that it will walk to the right square if possible. With the remaining probability, it will walk in the requested direction. So, if there are walls to the left and right, the agent will definitely walk into the requested direction. Whenever the agent walks to a square with gold, it will automatically collect it.
- **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 13.

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.

- **Bridge**: If your agent crosses a bridge, it will have to use its agility skill and get a score of at least 12. If successful, it can continue its adventure. Otherwise, it will fall into the pit below the bridge and not survive.

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.
- **"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": "XXXXXXXXXXXXXX\nXXXXXXXXXXXXXX\nXXW G G PXXXX\nXX XXXPX
      XXXXX\nXX XXXXXBXXXXX\nXX XXXXX XXXXX\nXX XXXXX XXXXX\nXX
      XXXXPWXXXXX\nXXS GPXXXX\nXXXXXXXX XXXXX\nXXXXXXXXXXXXXX\
      nXXXXXXXXXXXXXX",
  "history": [
    {"action": {"agility": 0,"fighting": 12},"outcome": {"agility": 0,"fighting": 12}},
    {"action": "NORTH","outcome": {"position": [2,7]}},
    {"action": "NORTH","outcome": {"position": [2,6]}},
    {"action": "NORTH","outcome": {"position": [2,5]}},
    {"action": "NORTH","outcome": {"position": [2,4]}},
```

```
    {"action": "NORTH","outcome": {"position": [2,3]}},
    {"action": "NORTH","outcome": {"position": [2,2]}},
    {"action": "FIGHT","outcome": {"killed-wumpus-at": [2,2],"expl": "dice: 553415214222"
        }},
    {"action": "EAST","outcome": {"position": [3,2]}},
    {"action": "EAST","outcome": {"position": [4,2],"collected-gold-at": [4,2],"current-
        amount-of-gold": 3}}
  ],
  "skill-points": {
    "agility": 0,
    "fighting": 12
  }
}
```


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. `{"agility": 0, "fighting": 12}`).

For the 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 ($\approx$ 1 page) or as part of your `README.md`.

5 A few tips

1. Consider using value iteration.

2. It can help with debugging if you can somehow look at the computed utilities/policy.
3. 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.
4. Code optimizations:
    - If you cache the utitilies/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 involes 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.
5. 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.

References

[AR] *Repository for Assignment 3: Wumpus Quest.* URL: `https://gitlab.rrze.fau.de/wrv/AISysProj/ss23/a2.3-wumpus-quest/assignment`.

Config	Obstacles	Skill points	Points
ss23-wquest-1.json		6	0 if rating < 2.0 15 if rating ≥ 2.0 25 if rating ≥ 3.0
ss23-wquest-2.json	B	6	0 if rating < 1.3 35 if rating ≥ 1.3 45 if rating ≥ 1.6
ss23-wquest-3.json	B, P	8	0 if rating < 1.3 45 if rating ≥ 1.3 70 if rating ≥ 1.8
ss23-wquest-4.json	B, P, W	12	0 if rating < 1.4 65 if rating ≥ 1.4 73 if rating ≥ 1.7 80 if rating ≥ 1.95

Figure 2: Number of points that you can get.

```
// 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.