

Assignment 2: Catch the Wumpus

AI-2 Systems Project (Summer Semester 2023)

Jan Frederik Schaefer

Friedrich-Alexander-Universität Erlangen-Nürnberg, Department Informatik

Topic: Markov models
Due on: August 4, 2023
Version from: June 21, 2023
Author: Jan Frederik Schaefer

1 Task summary

Your goal is to capture the Wumpus. As the Wumpus has good hearing and hides from people, you decided to use the help of a friend who can remotely create a portal that teleports the Wumpus into your cage. You have different ways of tracking the Wumpus, but unfortunately they are not very precise and you will have to combine the tracking data with your knowledge of how the Wumpus behaves to make an educated guess. You can test and evaluate your implementation on the server¹.


Didactic objectives

1. Gain experience modelling a problem with a non-trivial state space as a hidden Markov model,
2. implement basic algorithms for hidden Markov models,
3. solve various challenges involving probabilistic reasoning.

Prerequisites and useful methods

1. Basics of working with probabilities,
2. (hidden) Markov models and their basic algorithms (as covered in the lecture notes).

2 Detailed problem description

The Wumpus () lives in a 8×5 cave (Figure 1). Depending on the environment, you have different ways of tracking the Wumpus: via a precise location tracker, via a GPS tracker or

¹<https://aisysproj.kwarc.info>






0,0	1,0		3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1		6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	
0,3		2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4		7,4

Figure 1: The Wumpus cave.

via microphones placed in the cave. After tracking the Wumpus for 10 time steps, you can attempt to catch it with the help of a portal-maker. The portal-maker is only available for 5 time steps. You will have to create the portal within that time and have only one attempt. Creating a portal takes 1 time step, i.e. you have to create the portal in the square where you expect the Wumpus next.

In the following subsections, we will see how the Wumpus moves and how the sensors work.

2.1 Wumpology – how does the Wumpus move?

Every time step, the Wumpus either moves to one of the adjacent squares (north, east, south, west) or stays in its current square. The probability of the Wumpus taking a particular action depends on its previous action:

1. If the Wumpus stayed in its square, there is a $\frac{1}{2}$ probability that it will stay there for another time step. Otherwise, it will move to any of the adjacent free squares with equal probability. Note that the Wumpus cannot leave the 8×5 cave, so if it is e.g. in square $\langle 0, 1 \rangle$, it only has 3 options.
2. If the Wumpus moved in one direction, there is a $\frac{3}{5}$ probability that it will move in the same direction again (if that's possible) and a $\frac{1}{10}$ probability that it will move to the left or right respectively (if possible). Otherwise, it will stay in its square.

Figure 2 illustrates action probabilities with an example. We assume that the Wumpus has previously moved one square north. Then it is most likely ($\frac{3}{5}$) to next move north again,

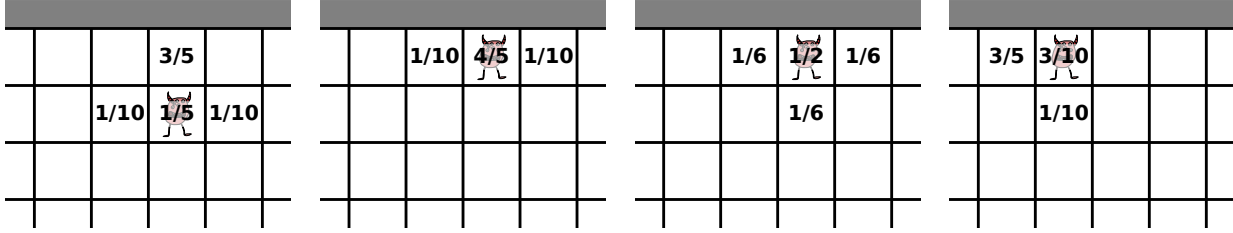


Figure 2: Probabilities of the Wumpus moving in a particular direction.

but there is also a $\frac{1}{10}$ chance that it turns left or right respectively. With the remaining probability ($\frac{1}{5}$), it will stay in its square. If the Wumpus ends up moving north, it reaches a wall, so it has to either stop (probability of $\frac{4}{5}$) or move left or right.

2.2 Sensors

Depending on the environment, you have different sensors to track the Wumpus.

2.2.1 Precise location tracker

The precise location tracker tells the exact location of the Wumpus at any time. It is intended to help you get started.

2.2.2 GPS tracker

The GPS tracker tells you the estimated coordinates of the Wumpus, but it is not very precise. Concretely, it reports the correct coordinates with a probability of $\frac{1}{9}$. It also reports each of the 8 surrounding squares with a $\frac{1}{9}$ probability.

2.2.3 Microphones

During every time step, the Wumpus makes a certain amount of noise, which can be picked up by microphones that were placed in the cave (Figure 1). If the Wumpus moves to a new square, the amount of noise it makes is uniformly randomly distributed over the interval $[1, 10]$. Otherwise, it is much quieter and the noise is uniformly randomly distributed over $[\frac{1}{2}, 2]$. For simplicity, we assume that the Wumpus only makes noise on its new square, i.e. if it moves from $\langle x_1, y_1 \rangle$ to $\langle x_2, y_2 \rangle$, we assume that the noise is produced in $\langle x_2, y_2 \rangle$.

There are three noise levels that the microphones can detect: 0 (silent), 1 (quiet) and 2 (loud). If the Wumpus lands on a square with a microphone, it will pick up a loud noise.

	$P(\text{silent})$	$P(\text{quiet})$	$P(\text{loud})$
$\frac{N}{d^2} \geq 1$	0	0	1
$\frac{N}{d^2} \in [\frac{1}{4}, 1)$	0	$\frac{9}{10}$	$\frac{1}{10}$
$\frac{N}{d^2} < \frac{1}{4}$	$\frac{8}{10}$	$\frac{1}{10}$	$\frac{1}{10}$

Table 1: Noise detection probabilities depend the amount of Wumpus noise N and the distance d to the microphone.

Otherwise, the amount of Wumpus noise that a microphone picks up is $\frac{N}{d^2}$, where N is the amount of noise made by the Wumpus and d is the (Euclidean) distance between the microphone and the Wumpus.

The microphone reports quiet or loud noise if it picked up noises above a certain threshold. Sometimes there are random noises in the cave not caused by the Wumpus. Table 1 shows the probabilities for the reported volume depending on $\frac{N}{d^2}$.

In some environments, the precise amount of Wumpus noise is also reported, which can be helpful for debugging.

3 Catching the Wumpus on the server

You should test and evaluate your agent by competing on the server. The rating of your agent is the ratio of successful captures 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 the current state of your Wumpus capturing attempt. Here is an example state description:

```
{
  "sensors": [
    {
```

```

    "microphones": {
      "2,0": 2, "5,1": 1, "7,2": 1, "6,4": 1, "1,3": 2, "wumpus": 8.626903632435095
    },
    "gps": "2,0"
  },
  {
    "microphones": {
      "2,0": 2, "5,1": 2, "7,2": 1, "6,4": 1, "1,3": 2, "wumpus": 8.098510160219618
    },
    "gps": "2,2"
  },
  ...
],
"remaining-attempts": 5
}

```

`sensors` lists the sensor history until now. `remaining-attempts` indicates how many attempts you have left to make a portal.

3.2 Sending Actions

Your agent should respond to an action request by sending an action as a string:

1. send "WAIT" to indicate that you do not want to make a portal yet,
2. send "PORTAL X,Y" to make a portal at the position $\langle [X], [Y] \rangle$.

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. You do not have to model everything perfectly – you can still get full points if you cut a few corners (e.g. in the sensor model).
2. Using matrices for the transition probabilities might be very difficult. Instead, you can just implement a function that computes the transition probabilities on a case-by-case basis.
3. For simplicity, it might make sense to find a way to model the problem as a first-order Markov model.
4. It is easy to make small mistakes that break your model and are hard to track down. It might be a good idea to check intermediate results where possible.
5. If your agent decides to wait with the portal creation, you can store the filtering results to re-use them later on and save computation time (in case that becomes a problem).

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 3 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 2: Catch the Wumpus*. URL: <https://gitlab.rrze.fau.de/wrv/AISysProj/ss23/a2.2-catch-wumpus/assignment>.

Config	Sensors	Points
ss23-2.2.1-tracker.json	tracker	$\left\{ \begin{array}{l} 0 \text{ if rating} < 0.5 \\ 20 \text{ if rating} \geq 0.5 \\ 30 \text{ if rating} \geq 0.65 \end{array} \right.$
ss23-2.2.2-gps.json	gps	$\left\{ \begin{array}{l} 0 \text{ if rating} < 0.2 \\ 30 \text{ if rating} \geq 0.2 \\ 40 \text{ if rating} \geq 0.3 \\ 55 \text{ if rating} \geq 0.4 \end{array} \right.$
ss23-2.2.3-microphones-simplified.json	microphones, wumpus-noise	$\left\{ \begin{array}{l} 0 \text{ if rating} < 0.1 \\ 30 \text{ if rating} \geq 0.1 \\ 40 \text{ if rating} \geq 0.25 \\ 50 \text{ if rating} \geq 0.4 \\ 65 \text{ if rating} \geq 0.45 \end{array} \right.$
ss23-2.2.4-microphones.json	microphones	$\left\{ \begin{array}{l} 0 \text{ if rating} < 0.15 \\ 35 \text{ if rating} \geq 0.15 \\ 60 \text{ if rating} \geq 0.3 \\ 75 \text{ if rating} \geq 0.45 \end{array} \right.$
ss23-2.2.5-both.json	microphones, gps	$\left\{ \begin{array}{l} 0 \text{ if rating} < 0.3 \\ 40 \text{ if rating} \geq 0.3 \\ 55 \text{ if rating} \geq 0.4 \\ 70 \text{ if rating} \geq 0.45 \\ 77 \text{ if rating} \geq 0.5 \\ 80 \text{ if rating} \geq 0.55 \end{array} \right.$

Figure 3: Number of points that you can get.

```

// First request:
{"agent": "MyAgent", "pwd": "r7iUM8o1NLbFdkl2WmBDldsYHD3wLwUQAKoG_2_xBcE", "actions": []}

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

// Second request
{"agent": "MyAgent", "pwd": "r7iUM8o1NLbFdkl2WmBDldsYHD3wLwUQAKoG_2_xBcE",
  "actions": [{"run": "40#1", "action": ...},
              {"run": "7#3", "action": ...}]}

```

Figure 4: 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 4 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.