

Assignment 2: Capture the Wumpus

AI2SysProj 2022

Topic: Hidden Markov models

Due on: August 16, 2022

Version from: July 5, 2022

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. To locate the Wumpus, you have installed several microphones in the Wumpus cave. You can test and evaluate your implementation on our server.

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 (Section 23 in the lecture notes).







$\langle 0, 0 \rangle$	$\langle 1, 0 \rangle$	$\langle 2, 0 \rangle$	$\langle 3, 0 \rangle$	$\langle 4, 0 \rangle$	$\langle 5, 0 \rangle$	$\langle 6, 0 \rangle$	$\langle 7, 0 \rangle$	$\langle 8, 0 \rangle$
$\langle 0, 1 \rangle$	$\langle 1, 1 \rangle$	$\langle 2, 1 \rangle$	$\langle 3, 1 \rangle$		$\langle 5, 1 \rangle$		$\langle 7, 1 \rangle$	$\langle 8, 1 \rangle$
$\langle 0, 2 \rangle$		$\langle 2, 2 \rangle$	$\langle 3, 2 \rangle$	$\langle 4, 2 \rangle$	$\langle 5, 2 \rangle$	$\langle 6, 2 \rangle$		$\langle 8, 2 \rangle$
$\langle 0, 3 \rangle$	$\langle 1, 3 \rangle$	$\langle 2, 3 \rangle$	$\langle 3, 3 \rangle$	$\langle 4, 3 \rangle$	$\langle 5, 3 \rangle$	$\langle 6, 3 \rangle$	$\langle 7, 3 \rangle$	$\langle 8, 3 \rangle$
$\langle 0, 4 \rangle$		$\langle 2, 4 \rangle$	$\langle 3, 4 \rangle$	$\langle 4, 4 \rangle$	$\langle 5, 4 \rangle$		$\langle 7, 4 \rangle$	$\langle 8, 4 \rangle$

Figure 1: The Wumpus cave.

2 Detailed problem description

The Wumpus (⊕) lives in a 9×5 cave (Figure 1). Every time step the Wumpus performs an action (move to an adjacent square or stay where it is) and makes some noise in the process. There are 5 microphones, located at $\langle 1, 2 \rangle$, $\langle 1, 4 \rangle$, $\langle 4, 1 \rangle$, $\langle 6, 4 \rangle$ and $\langle 7, 2 \rangle$. Every time step, they report either no noise (🔇), quiet noise (🔊) or loud noise (🔊🔊).

After recording the noise levels for 10 time steps, your portal-maker is available for 6 time steps. You will have to create the portal within that time to catch the Wumpus and you 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.

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 squares with equal probability. Note that the Wumpus cannot leave the 9×5 cave, so if it is e.g. in square $\langle 7, 4 \rangle$, 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. Let us assume that the Wumpus has previously moved from $\langle 6, 2 \rangle$ to $\langle 6, 1 \rangle$. Then it is most likely to next move to $\langle 6, 0 \rangle$. There is also a chance that it moves left ($\frac{1}{10}$) or right ($\frac{1}{10}$). With the remaining probability ($\frac{1}{5}$), it will stay in $\langle 6, 1 \rangle$. After moving to $\langle 6, 0 \rangle$ it has reached the cave boundary, so it has to either stop (probability of $\frac{4}{5}$) or move left or right.

2.2 Noise and microphones

During every time step, the Wumpus makes a certain amount of noise, which the microphones can pick up. 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

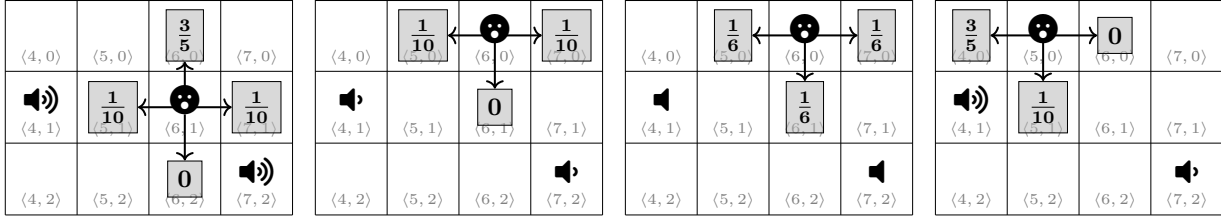


Figure 2: Probabilities of the Wumpus moving in a particular direction.

	$P(\text{🔊})$	$P(\text{🔊})$	$P(\text{🔊🔊})$
$\frac{N}{d^2} \geq 1$	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.

only makes noise on its new square, i.e. if it moves from $\langle 6, 2 \rangle$ to $\langle 6, 1 \rangle$, we assume that the noise is produced in $\langle 6, 1 \rangle$.

If the Wumpus is on a square with a microphone, that microphone reports a loud noise (🔊🔊). 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 🔊 or 🔊🔊 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 $\frac{N}{d^2}$.

For example, let us assume the Wumpus has moved from $\langle 6, 2 \rangle$ to $\langle 6, 1 \rangle$ and made a relatively quite noise $N = \frac{3}{2}$ (Figure 1). Then for the microphone at $\langle 7, 2 \rangle$ we have $d = \sqrt{2}$ and $\frac{N}{d^2} = \frac{3}{4}$. Therefore, the microphone will report 🔊 with a $\frac{9}{10}$ probability and 🔊🔊 otherwise.

3 Catching the Wumpus on the server

You should test and evaluate your agent by competing on the server. The strength of your agent is the maximum number of times it caught the Wumpus 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. 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:

```
{
  "remaining-time": 3,
  "sensor-locations": {"s1": [1, 2], "s2": [1, 4], "s3": [4, 1], "s4": [6, 4], "s5": [7, 2]},
  "sensor-history": [
    {"s1": 0, "s2": 0, "s3": 1, "s4": 0, "s5": 2},
    {"s1": 0, "s2": 1, "s3": 2, "s4": 0, "s5": 2},
    ...
  ]
}
```

`remaining-time` indicates the number of time steps remaining to create a portal (your last opportunity to make a portal is when the remaining time is 0). `sensor-locations` describes the microphone locations (though they never change). `sensor-history` lists the reported noises for each time step. `◀` is represented as 0, `◀▶` as 1, and `◀▶▶` as 2.

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 $\langle [X], [Y] \rangle$.

3.3 Light mode

The server also provides a “light mode”: By switching on the light in the cave, you can directly observe the Wumpus movements. The action requests then also have a `wumpus-locations` field in the description that lists the places where the Wumpus has been seen at each time step.

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. Start with the light mode.
2. 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).
3. 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.
4. For simplicity, it might make sense to find a way to model the problem as a first-order Markov model.
5. 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.
6. 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). Concretely, you will get the following points for the *light mode*:

- 20 points if the strength is ≥ 500 .
- 30 points if the strength is ≥ 650 .

For the normal mode, you will get additionally

- 20 points if the strength is ≥ 150 .
- 30 points if the strength is ≥ 240 .
- 40 points if the strength is ≥ 330 .

- 50 points if the strength is ≥ 420 .

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).

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).

Please try to avoid flooding the server with requests. In particular, respond with an empty list of action requests if there is nothing to do for you. In that case, please wait for one second before sending another request.

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,
- **tournament**: the name of the tournament¹,
- **name**: the agent’s name,
- **pwd**: the agent’s password, and
- **single-request**: indicating whether the server should send only one action request at a time rather than sending multiple action requests in parallel for efficiency (optional boolean property, default: `false`).

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

- **name**: the agent’s name (from the configuration file).
- **pwd**: the agent’s password (from the configuration file).
- **actions**: the actions the agent wants to do as a list. Each action is represented as an object with two fields: `id` is an identifier of the action request (provided in the server

¹The name “tournament” comes from the idea of agents competing in a game. In general, a “tournament” is one specific way of evaluating your agent. For some assignments, there might be multiple “tournaments” that allow you to e.g. evaluate your agent in a simplified setting.

```

// First request:
{'name': 'MyAgent', 'pwd': 'r7iUM8o1NLbFdkI2WmBDldsYHD3wLwUQAKoG.2_xBcE', 'actions': []}

// First response
{'errors': [], 'messages': [], 'action-requests': [
  {'id': '40#1', 'content': ...},
  {'id': '7#3', 'content': ...}]}

// Second request
{'name': 'MyAgent', 'pwd': 'r7iUM8o1NLbFdkI2WmBDldsYHD3wLwUQAKoG.2_xBcE',
  'actions': [{'id': '40#1', 'action': ...},
              {'id': '7#3', 'action': ...}]}

```

Figure 3: Example interaction with 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: **id** is an identifier so that your action can be linked to the request and **content** is the content of the action request (e.g. the state of 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.