Advanced Topics in Artificial Intelligence

T-720-ATAI-2016

EXERCISE 3 (Partial Observability)

In this exercise you will use agents that cannot observe the environment's full state.

Update TEAL

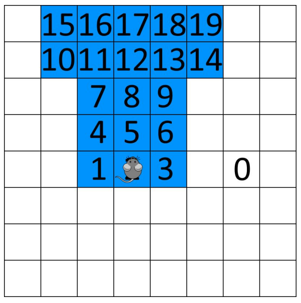
Pull the latest version of TEAL if you're using Git, or download it from https://github.com/ThrosturX/rl-mouse/tree/exercises. Do this before every exercise.

Curse of dimensionality

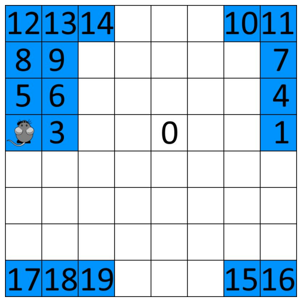
In the previous exercise we saw that the time needed to learn the task increases explosively when the grid size is increased. Even for a tiny world of 20x20 cells, this would take a prohibitively long time. The problem is that the number of observations that the agent can make is almost 20^4=160,000, which means that there are 20^4*3 state-action pairs that need to be tried multiple times. One way to deal with this problem is to reduce the number of possible observations. We could argue that this is also more realistic, because our field of view is limited in reality too.

Cone of view

Implement a solution that uses `MouseAgent` instead of `OmniscientMouseAgent` (you can use `ex3_template.py` as a template). This agent has a cone for a field of view (FOV):



Field of view in blue.



Remember the world is circular.

The agent gets two inputs: one for the location of the cheese and one for the location of the trap. If the cheese/trap is in the FOV, that input will correspond to the number in the picture above. Otherwise it will be 0.

Call `teal.py <solution-file> --grid_size <size> --seed <your-kennitala> --max_epochs 200` to run your experiments.

Quick note: You can get immediate results with CTRL+C (`KeyboardInterrupt`) if you notice that the task is solved but many evaluation epochs remain.

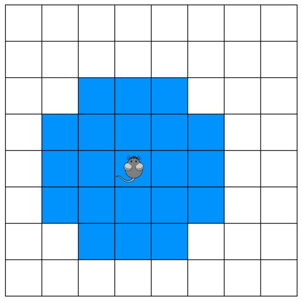
A. Answer the following questions. Support your answers with appropriate arguments and data.

1. How many possible observations are there now?
2. Looking back at the previous exercise, how many actions do you think this agent needs to learn the task?
3. Run the experiment on an 8x8 grid, using epochs of 100, 1000 and 10000 actions. Can the agent solve the task (i.e. get more than 390 reward 90% of the time)? How long does it take the agent to learn to its full potential? How do you explain this result? (*Hint: Remember you can compare solutions by adding `--compare_to <solution>` to your call to TEAL.)*
4. What do you think will happen on a 15x15 grid in terms of performance, ratio, timeouts, deaths and speed of learning?
5. Run the experiment and compare the results to your predictions.
6. Take a look at the ratio of the shortest route to the cheese and the actual number of steps that the agent took. Compare this to the results you got in previous exercises. How do you explain this?
7. Imagine there is no trap (anywhere) and the cheese is at location 18: i.e. the observation is (18, 0). Describe the sequence of optimal actions and subsequent observations to get to the cheese. E.g. if you think the mouse should move forward first, start your answer like this:
    - (18, 0) → forward
    - (13, 0) → *action2*
    - *observation3* → *action3*
    - …
    - (2,0) → found!
8. Do the same for observation (19,0).

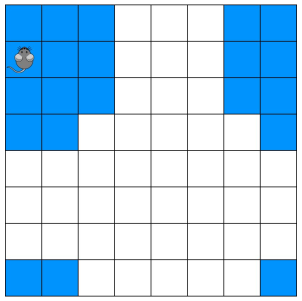
9. Could this be turned into a policy: i.e. a mapping from observations to actions like you had to define in Exercise 1? Why (not)?
10. Can you come up with a solution to this problem while keeping the inputs as they are?

Eyes everywhere

The `RadiusMouseAgent` has a sensory field that is square centered on the agent:



Field of view in blue.



Remember the world is circular.

B. Answer the following questions. Support your answers with appropriate arguments and data.

1. Do you think this solves the problem we ran into with the cone shaped POV? Why (not)?
2. Do you expect this agent to perform better or worse than the one with the cone shaped POV? Why (not)?
3. Copy your solution and change `MouseAgent` to `RadiusMouseAgent` in the new solution file. Compare the performance of the agents, and explain the differences you see and why you think they occur.

Your point of view

The `CustomMouseAgent` has a sensory field that you can specify yourself.

C. Answer the following questions. Support your answers with appropriate arguments and data.

1. Create a new solution file (you can copy one of the others) and use a `CustomMouseAgent` with a FOV that you think will work best. Try to beat the other agents with as few observed cells as possible. Explain your choice of FOV.

Compare the performance of the agents, and explain the differences you see and why you think they occur.

2. Observing less cells on the grid is not the only way in which we can reduce the number of possible observations. Try to come up with the optimal inputs for the agent so that you minimize the number of possible observations while maximizing the amount of information the agent has. It should be possible to come up with inputs that solve the problem with the many timeouts that the other agents get on any grid size.
3. BONUS: If you want, you can implement this new agent and test it out. Take a look in agent.py to see how the other agents are implemented. This is completely optional.

Submit your solutions

Submit your solution file with your own FOV (and possibly your new agent if you did C3) as “ex3_” followed by your name “.py”. Put your answers to the above questions in a file with the same name, but a “.txt” extension. Please write your full name and kennitala into both files (as a comment in the Python file). Submit them in MySchool.