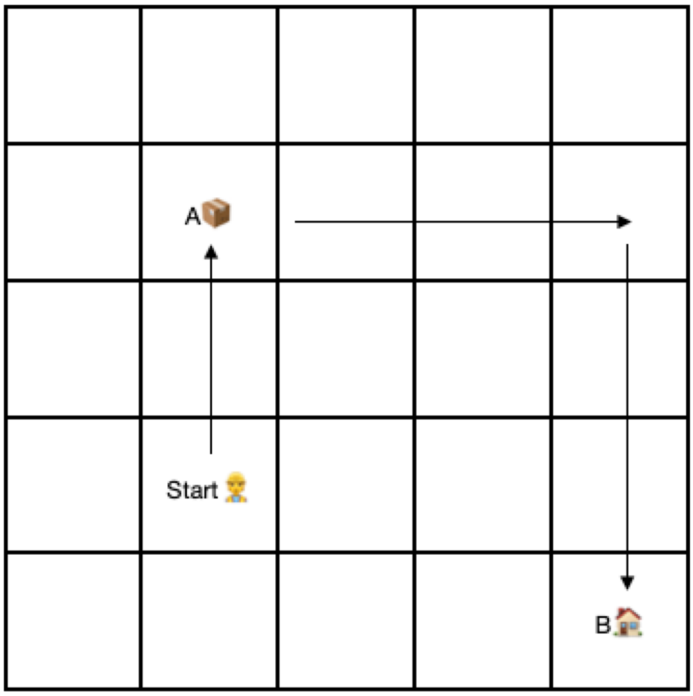
Lab - Week 7

Introducing Deep-Q

The purpose of this exercise is to ensure that you know how to implement (a simple version of) Deep-Q. This also completes our preparation for the individual assignment. You can go one of two pathways: you can either simply take the same task that you have implemented with tabular q-learning in the last couple of weeks and switch it over to deep-q or, alternatively, you can make the task a bit harder as described below.

---

1. Extending the grid world task.



If you are feeling confident and have completed the tabular-q implementation successfully, we recommend you consider this extension right from the the start. If, however, you haven't completed the simple task with tabular-q yet or do not feel particularly confident with it, it is probably better to continue with the simpler task (and only consider the extension once you have finished this).

The agent's task is a minor variation of the Task used in Stage 1. Please refer to the description of Stage 1 if you are unsure about any details of the earlier stage.

- In this stage 2 the target location B is not fixed but randomly selected for every episode. In other words, the target location B is handled in the same way as the start location A was handled in Stage 1. The agent can observe the location of B.

Tasks

1. Consider what the new state space needs to look like.
2. Make the appropriate adjustments in your code from the previous task.

---

2. Switching to Deep-Q

Instead of Tabular Q-Learning, you will now use Deep-Q learning. This is, in part, a consequence of having a larger search space. This is to let you gain experience with using Deep-Q that you will need for your assignment. In order to reduce the training time that your program will need, you you can reduce the grid size to 4x4.

We do, of course, not expect that you are able to program the low-level handling and learning of the Q-network since neural network knowledge is not a prerequisite for the unit.

We are thus providing you with a Python module that you can use at a higher level. This is available on Moodle as a Jupyter notebook **Stage2SkeletonV2.ipynb** in the Assignment section together with these instructions. The core functions in this module are

- `prepare_torch()`: this function performs the entire initialisation that is required for PyTorch and generates the target and prediction networks
- `get_qvals(state)`: this function returns the list of q-values for the state given as the argument from the prediction network
- `get_maxQ(state)`: this function returns the maximal q-value for the state given as the argument from the target network
- `update_target()`: this function copies the state of the prediction network to the target network
- `train_one_step(states, actions, targets, gamma)`: this function performs a single training step. As arguments, it receives three parallel Python lists and the discount factor gamma. The three lists together capture a minibatch for the update. Each triple states[i], actions[i], targets[i] fully describes one transition from the minibatch, with states[i] being the origin state, action[i] the chosen action and targets[i] the TD-target for this transition. Your program will have to compute the TD-targets based on the target state and the reward recorded in the replay buffer and pass them to this network training function.

These functions are documented in the notebook together with the open-source code. You do not need to worry about the source code unless you are interested.

Please be aware that this way of handling the network update is somewhat slower than programming it directly in PyTorch since data is converted several times between PyTorch and plain Python. However, this is unavoidable since we need to ensure that you don't need to handle

PyTorch (or any other deep learning framework) directly yourself.

Guidance on parameter setting (so that you can train in a reasonable amount of time) is:

```
Learning rate = 0.997
Starting epsilon = 1.0
Epsilon decay factor=0.997
Minimum epsilon=0.1
Replay buffer size = 1000
Batch size = 200
Network copy frequency = every 500 steps
```

Tasks

1. Switch you previous implementation from tabular q-learning to deep q-learning using the provided library.
2. Devise a method and metrics to track the performance of the agents as it learns.
3. Training and Testing: Train and test your Q-Learner,.