Warning

This page is located in archive. Go to the latest version of this course pages. Go the latest version of this page.

Task 2: BlockWorld and Q-learning

Henwas' trouble even greater

You sure remember our poor fella Henwas. Since last time, he grew only older and the memory does not serve him as before. Unfortunately, this translates into him making mistakes during moving the crates.

Once again, he turns to you for help. He wants you to prescribe for him what to do in any situation, should he suddenly forget the plan. In another words, he wants you to give him a *policy*.

Your Job

Download task2_qlearning_v2.zip [/b222/_media/courses/zui/tasks/task2_qlearning_v2.zip]. Once again, you are given a BlockWorld problem. Only now, there is a chance that the moved block is put somewhere else. For example:

```
$ python blockworld.py

state = [[1], [3, 4], [2]]
actions = [(1, 3), (1, 2), (3, 0), (3, 1), (3, 2), (2, 1), (2, 3)]

<from> <to>: 3 2
state = [[3, 1], [4], [2]]
```

Here, the block `3` should have been moved to `2`, but instead ended up on `1`.

In `student.py`, you'll find the following:

```
from blockworld import BlockWorldEnv
import random

class QLearning():
```

```
# don't modify the methods' signatures!
def __init__(self, env: BlockWorldEnv):
        self.env = env

def train(self):
        # Use BlockWorldEnv to simulate the environment with reset() and

        # s = self.env.reset()
        # s_, r, done = self.env.step(a)

        pass

def act(self, s):
        random_action = random.choice( s[0].get_actions() )
        return random_action
```

Implement Q-learning algorithm in `train()`. The `act(s)` method should return a single action to perform in the given state. To simulate the environment, use `reset()` and `step(a)` methods of the given `BlockWorldEnv`.

The `reset()` method randomly chooses a new `state` and `goal` and returns them as a tuple `(state, goal)`. The `step(a)` method performs the action and returns an updated state in form of `(new_state, goal)`, reward and a termination flag. To get available actions, call `state.get_actions()`.

Note that your algorithm has to react to all possible combination of states and goals. You can assume that number of blocks will be less or equal to 4.

Hint: You can use Python's dictionary `get(key, default_value)` method to get a stored value, or the `default_value` if the `key` is not present.

Evaluation

Upload your solution (only `student.py`) into Brute [https://cw.felk.cvut.cz/brute/student/course/1444], where it will be evaluated automatically.

First, `train()` method of your class is called and given 30 seconds to perform Q-learning. Second, it is evaluated on 1000 instances, each with a limit of 50 steps and a time limit of 5 seconds for all of them together.

You will gain 10 points, if you perform comparatively to the reference solution, and 0 points if you perform close or worse than random. For your information, the reference solutions solves 95% of problems with 6 steps on average. Random sampling solves about 35% of problems with about 20 steps on average.

Note: You don't have to measure the time of your `train()` method, the evaluation script will

automatically terminate it when 15 seconds elapse. All what you compute in the alloted time and save within your class is then used in the evaluation. However, the evaluation is considered unsuccessful if you exceed 5 seconds, and zero points are awarded in that case.

Note: You are supposed to implement Q-learning and we will randomly check your solutions.

courses/zui/tasks/task2.txt · Last modified: 2023/04/15 13:12 by janisjar