RL_Assign2_19024_KhadgaJyoth

April 19, 2022

0.1 Alli Khadga Jyoth 19024 - DSE

[1]:
```
from gym import Env
import gym
import pygame
from gym.spaces import Discrete, Box,Dict
import numpy as np
import random
```

[2]:
```
class WarehouseAgent(gym.Env):
    def __init__(self):
        """
        Initializing the environment
        """
        self.metadata = {"render_modes": ["human", "rgb_array"], "render_fps":␣
↪4}

        self.window_size = 512
        self.GRID_DIM = [6,7]
        self.size = self.GRID_DIM[0]
        self.agent_position = [1,2]

        self.box_location = [4,3]
        self.goal_location = [3,1]
        """directions: UP, DOWN, LEFT, RIGHT."""
        self._action_to_direction = {
            0: np.array([-1, 0]),
            1: np.array([1, 0]),
            2: np.array([0, -1]),
            3: np.array([0, 1]),
        }
        self._ACTIONLOOKUP = {
            0: 'push up',
            1: 'push down',
            2: 'push left',
            3: 'push right',
            4: 'move up',
            5: 'move down',
```

```
            6: 'move left',
            7: 'move right',
            }

        self.action_space = Discrete(len(self._ACTIONLOOKUP.keys()))
        self.state_space = Discrete((self.GRID_DIM[0]*self.GRID_DIM[1]) **3)
        self.observation_space = Dict(
            {
                "agent": Box(np.array([0,0]), np.array([self.GRID_DIM[0]-1,self.
 ↪GRID_DIM[1] - 1]), shape=(2,), dtype=int),
                'box' : Box( np.array([0,0]), np.array([self.GRID_DIM[0]-1,self.
 ↪GRID_DIM[1] - 1]), shape=(2,), dtype=int),
                "target": Box( np.array([0,0]), np.array([self.
 ↪GRID_DIM[0]-1,self.GRID_DIM[1] - 1]), shape=(2,), dtype=int),
            }
        )
        self._agent_location = np.array(self.agent_position)
        self._box_location = np.array(self.box_location)
        self._target_location = np.array(self.goal_location)


        self.window = None
        self.clock = None


    def step(self, action):        #change = CHANGE_COORDINATES[(action - 1) %␣
 ↪4]
        self._prev_agent_location = None
        self._prev_box_location=None# new_position = self.player_position +
        moved_box = False
#         if action ==0:
#             moved_player = False
        if action <4:
            moved_player, moved_box = self._push(action)
        else:
        # Map the action (element of {0,1,2,3}) to the direction we walk in
#         direction = self._action_to_direction[action]
            moved_player = self._move(action)

        # An episode is done iff the agent has reached the target
        done = np.array_equal(self._box_location, self._target_location)
        reward = 0 if done else -1  # Binary sparse rewards
        observation = self._get_obs()
        info = self._get_info()

        return observation, reward, done, info
#     def render(self):
```

```
#         """Function to get the simulation of the warehouse agent system
#         """
#         pass
    def render(self, mode="human"):
            if self.window is None and mode == "human":
                pygame.init()
                pygame.display.init()
                self.window = pygame.display.set_mode((self.window_size, self.
 ↪window_size))
            if self.clock is None and mode == "human":
                self.clock = pygame.time.Clock()

            canvas = pygame.Surface((self.window_size, self.window_size))
            canvas.fill((255, 255, 255))
            pix_square_size = (
                self.window_size / self.GRID_DIM[0]
            )  # The size of a single grid square in pixels

            # First we draw the target
            pygame.draw.rect(
                canvas,
                (255, 0, 0),
                pygame.Rect(
                    pix_square_size * self._target_location,
                    (pix_square_size, pix_square_size),
                ),
            )
            # Draw the box
            pygame.draw.rect(
                canvas,
                (0, 255, 0),
                pygame.Rect(
                    pix_square_size * self._box_location,
                    (pix_square_size, pix_square_size),
                ),
            )
            # Now we draw the agent
            pygame.draw.circle(
                canvas,
                (0, 0, 255),
                (self._agent_location + 0.5) * pix_square_size,
                pix_square_size / 3,
            )

            # Finally, add some gridlines
            for x in range(self.size + 1):
                pygame.draw.line(
```

```
                    canvas,
                    0,
                    (0, pix_square_size * x),
                    (self.window_size, pix_square_size * x),
                    width=3,
                )
                pygame.draw.line(
                    canvas,
                    0,
                    (pix_square_size * x, 0),
                    (pix_square_size * x, self.window_size),
                    width=3,
                )

            if mode == "human":
                # The following line copies our drawings from `canvas` to the␣
↪visible window
                self.window.blit(canvas, canvas.get_rect())
                pygame.event.pump()
                pygame.display.update()

                # We need to ensure that human-rendering occurs at the␣
↪predefined framerate.
                # The following line will automatically add a delay to keep the␣
↪framerate stable.
                self.clock.tick(self.metadata["render_fps"])
            else:  # rgb_array
                return np.transpose(
                    np.array(pygame.surfarray.pixels3d(canvas)), axes=(1, 0, 2)
                )
    def close(self):
        if self.window is not None:
            pygame.display.quit()
            pygame.quit()

    def reset(self, seed=None, return_info=False, options=None):
        # We need the following line to seed self.np_random
#         super().reset(seed=seed)
        self._agent_location = np.array(self.agent_position)
        self._box_location = np.array(self.box_location)
        self._target_location = np.array(self.goal_location)

        observation = self._get_obs()
        info = self._get_info()
        return (observation, info) if return_info else observation
```

```
    def _get_obs(self):
        return {"agent": self._agent_location, 'box':self.
 ↪_box_location,"target": self._target_location}
    def _get_info(self):
        return {"distance": np.linalg.norm(self._box_location - self.
 ↪_target_location, ord=1)}
    def _push(self,action):
        direction = self._action_to_direction[(action ) % 4]
#         new_position = self.player_position + direction
        if np.array_equal(self._agent_location + direction,self._box_location):
            self._prev_box_location = self._box_location
            self._box_location = np.clip(
                self._box_location + direction, [0,0],[self.GRID_DIM[0]-1,self.
 ↪GRID_DIM[1]-1]
            )
            if np.array_equal(self._prev_box_location,self._box_location):
                self._agent_location = self._agent_location
                return False,False
            else:
                self._agent_location = np.clip(
                    self._agent_location + direction, [0,0],[self.
 ↪GRID_DIM[0]-1,self.GRID_DIM[1]-1]
                )
            return True,True
        else: return False,False
    def _move(self,action):
        direction = self._action_to_direction[(action ) % 4]
        # We use `np.clip` to make sure we don't leave the grid
        self._prev_agent_location = self._agent_location
        self._agent_location = np.clip(
            self._agent_location + direction, [0,0],[self.GRID_DIM[0]-1,self.
 ↪GRID_DIM[1]-1]
        )
        if np.array_equal(self._agent_location,self._box_location):
            self._agent_location = self._prev_agent_location
            return False
        if np.array_equal(self._prev_agent_location ,self._agent_location):
            return False
        else: return True
    def actionSpaceSample(self):
        return np.random.choice(self.action_space)
```

_ACTIONLOOKUP = { > 0: 'push up', > 1: 'push down', > 2: 'push left', > 3: 'push right', > 4: 'move up', > 5: 'move down', > 6: 'move left', > 7: 'move right', }

[3]:
```
env = WarehouseAgent()
```

[4]:
```
env._get_obs()
```

[4]:
```
{'agent': array([1, 2]), 'box': array([4, 3]), 'target': array([3, 1])}
```

[5]:
```
env.step(5)
```

[5]:
```
({'agent': array([2, 2]), 'box': array([4, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 3.0})
```

[6]:
```
env.step(5)
```

[6]:
```
({'agent': array([3, 2]), 'box': array([4, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 3.0})
```

[7]:
```
env.step(5)
```

[7]:
```
({'agent': array([4, 2]), 'box': array([4, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 3.0})
```

[8]:
```
env.step(5)
```

[8]:
```
({'agent': array([5, 2]), 'box': array([4, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 3.0})
```

[9]:
```
env.step(7)
```

[9]:
```
({'agent': array([5, 3]), 'box': array([4, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 3.0})
```

[10]:
```
env.step(0)
```

[10]:
```
({'agent': array([4, 3]), 'box': array([3, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 2.0})
```

[11]:
```
env.step(7)
```

[11]:
```
({'agent': array([4, 4]), 'box': array([3, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 2.0})
```

[12]:
```
env.step(4)
```

[12]:
```
({'agent': array([3, 4]), 'box': array([3, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 2.0})
```

[13]:
```
env.step(6)
```

[13]:
```
({'agent': array([3, 4]), 'box': array([3, 3]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 2.0})
```

[14]:
```
env.step(2)
```

[14]:
```
({'agent': array([3, 3]), 'box': array([3, 2]), 'target': array([3, 1])},
 -1,
 False,
 {'distance': 1.0})
```

[15]:
```
env.step(2)
```

[15]:
```
({'agent': array([3, 2]), 'box': array([3, 1]), 'target': array([3, 1])},
 0,
 True,
 {'distance': 0.0})
```

The box is at its target location

[ ]:

[ ]: