Environment "Cleaner" Documentation

Author: Shuo Jiang (jiang.shuo@husky.neu.edu)

Introduction

The document introduces an environment for multi agent study as two agents (or more) are trying to cooperatively clean the room with complex indoor barriers. The code is implemented in python and provided with simple interfaces. The environment is decoupled with learning method so reader can try any algorithms on.

Scenario Description

The task of the environment is explained as below

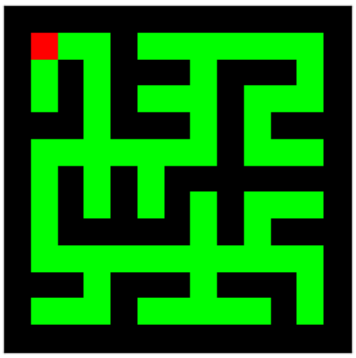

Figure 1

In the environment, agents (marked using color red, the position can overlap) will clean the indoor environment of a room. Where the black blocks are walls that agents cannot go through. The green blocks are dirty floors that agent should clean. At each step, each agent will clean the block it is standing on, and turn it to white (means it is clean). The task can be trying to maximize the area that can clean in given time or try to clean all the floors in minimal time. All the agents will be born on the left top corner of the room. The task is actually a path planning task that agents should plan paths which have minimal overlapping.

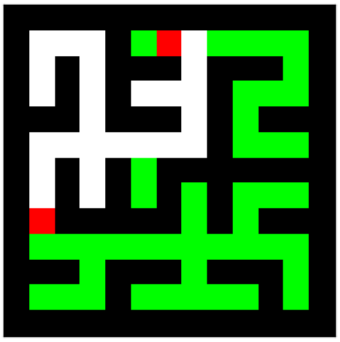

Figure 2. Two agents are cleaning the room

The coordinate system is shown as

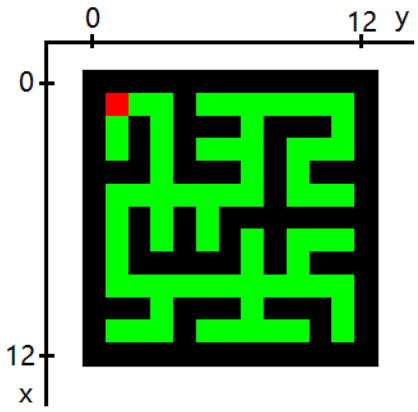

Figure 3

The area is 13*13 large, and agents are born at [1, 1].

There are 4 possible actions for the two agents go up/down/left/right, using an integer as 0/1/2/3. When the agent moves, it will move to the free space of 1 cell near it.

The observation is an image (numpy array) as figure 2, with format (width = 13, height = 13, rgb_channel = 3)

When agent clean a block, the team will get a shared reward 1, otherwise get 0. If clean multiple block in one time step, get multiple rewards

Class Organization

The environment is programmed in python 3.6 and has very simple interfaces. The file contains the environment is "env_Cleaner.py" and a class " EnvCleaner" is defined.

The class has the following interfaces:

__init__(self, N_agent, map_size, seed)
reset(self)
obs = get_global_obs(self)
step(self, action_list)
render(self)

__init__(self)
initialize the object, important variables are
N_agent: number of agents
map_size: can be change, in example is 13, should be an odd positive integer
seed: control the shape of maze, use the same seed will generate the same room

get_global_obs (self):
This function return a list of current visions of the scenario as an image, with form (width = 13, height = 13, rgb_channel = 3) as shown in Figure 2

step(self, action_list):
The function updates the environment by feeding the actions for each agent. The actions are denoted by integers 0/1/2/3. action_list = [action1, action2, …]. The return is shared reward as an integer

reset(self):
This function relocates the agents to the initial position

render(self)
This function plot scene in animation, call in each step

Example

Here is an example using test function, and it is in "test_Cleaner.py". The actions for agents are random chosen, click and run.

```
from env_Cleaner import EnvCleaner
import random

if __name__ == '__main__':
```

```
env = EnvCleaner(2, 13, 0)
max_iter = 1000
for i in range(max_iter):
    print("iter= ", i)
    env.render()
    action_list = [random.randint(0, 3), random.randint(0, 3)]
    reward = env.step(action_list)
    print('reward', reward)
```