Environment "Opposite" Documentation

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

Introduction

The document introduces an environment for several agents navigate to their goals in a grid world. 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

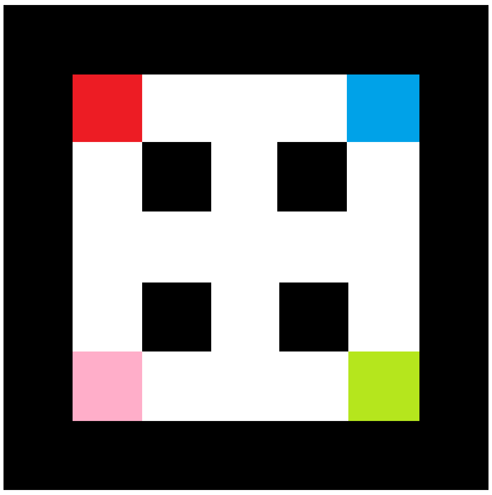


Figure 1

There are four agents in the maze (red, blue, green, pink), they have to find a way to the goal position at the diagonal opposite position. For example the goal of red agent is the position of green agent is standing. Agents can only go to white cells and agents cannot overlap. Only when four agents are all standing on their goal positions, the episode finishes. The coordinate system is shown as

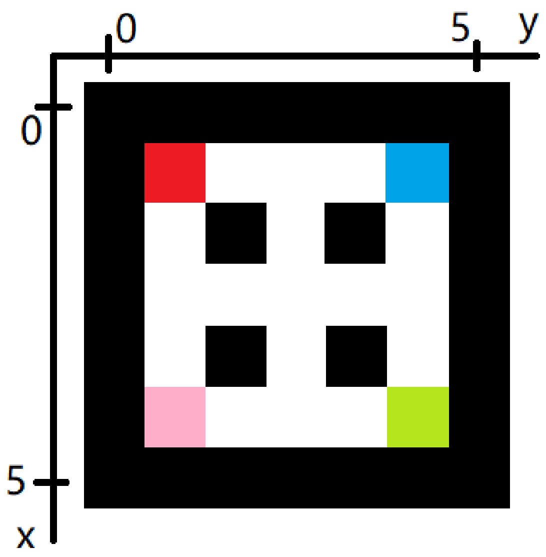


Figure 2

There are 5 possible action for the agent

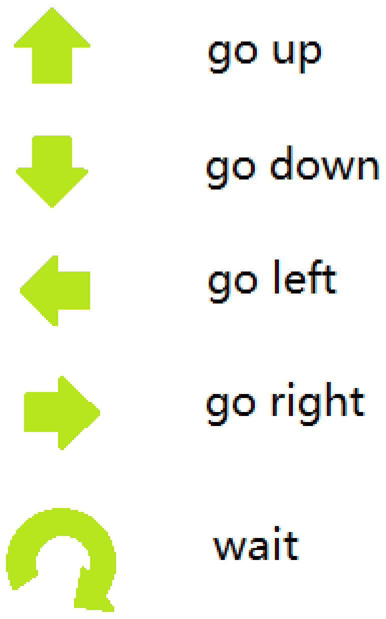


Figure 3

When the agent moves, it will move to the free space of 1 distance near it. However, when there is a block, the action won't work.

Agent who reaches its corresponding goal position will be rewarded +5. Reward are shared that one can imagine that when all four agents are standing on their goal positions, there will be a reward of +20.

The observation is a numpy array of size (1, 8), recording the normalized x, y position for each agent.

Class Organization

The environment is programmed in python 3.6 and has very simple interfaces. The file contains the environment is "env_OppositeV2.py" and a class " Env OppositeV2" is defined.

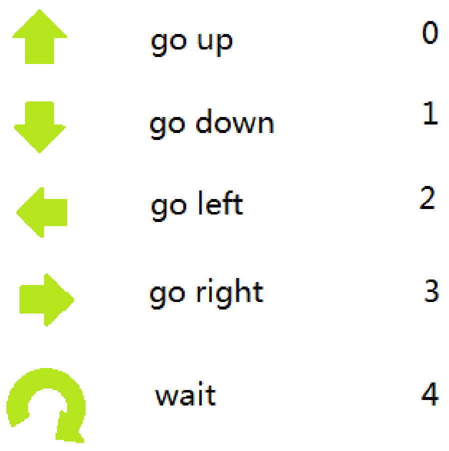
The class has the following interfaces:

__init__(self, size)
get_global_obs(self)
reward, done = step(self, action_list)
reset(self)
render(self)

__init__(self, size)
Initialize the scenario with given size, the size can be any odd integer no smaller than 5

get_global_obs(self)
Return a (size, size, 3) numpy array as image.

step(self, action_list):
The function updates the environment by feeding the actions as a list for agents. We use number the actions as integers



⬆	go up	0
⬇	go down	1
⬅	go left	2
➡	go right	3
↻	wait	4

Figure. 4

And action_list can be a list as [3, 1, 2, 0]. The function returns a reward and bool variable if the episode is finished.

reset(self):

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_Opposite.py". The action is random chosen, click and run.

```
from env_OppositeV2 import EnvOppositeV2
import random

if __name__ == '__main__':
    env = EnvOppositeV2(7)
    max_iter = 100000
    for i in range(max_iter):
        env.render()
        action_list = [random.randint(0, 4), random.randint(0, 4), random.randint(0, 4),
random.randint(0, 4)]
        reward, done = env.step(action_list)
        print("iter= ", i, 'reward', reward)
        if done:
            print('find goal, reward')
            env.reset()
```