

Environment "Find the Goals" Documentation

Author: Shuo Jiang (jiangshuo_sd@126.com)

License: Northeastern University, Lab for Learning and Planning in Robotics(LLPR)

Introduction

The document introduces a environment for multi agent study as two agents are trying to find their goals in a maze. 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.

Scene Description

The task of the environment is explained as below

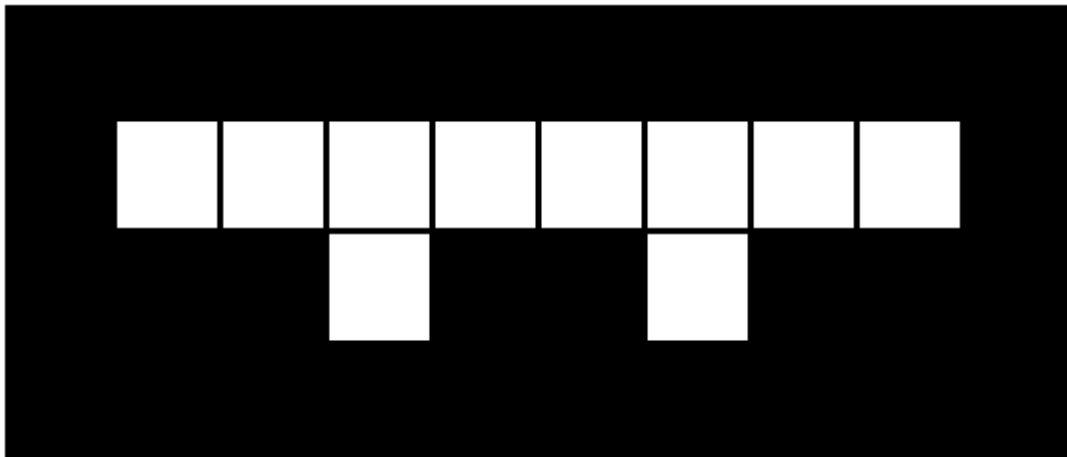


Figure 1

The environment works like a maze, two agents(agent 1 and agent2) will run in the maze and find their targets. The two agents, agent 1 and agent2 will be denoted using color **red** and **blue**, and will be shown as red and blue rectangles in the maze like:

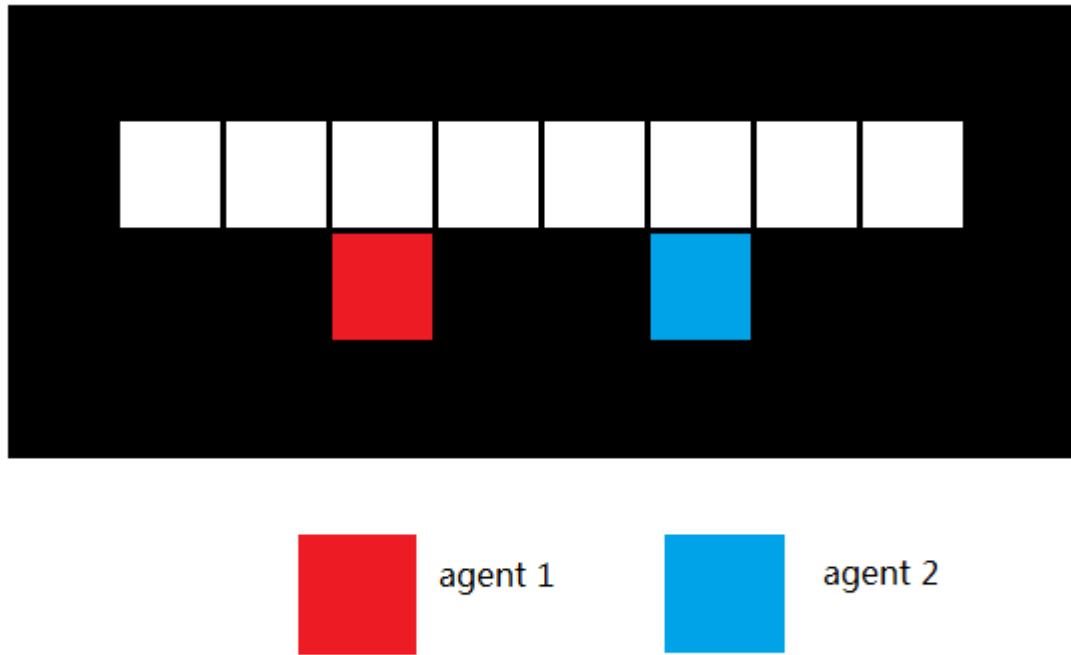


Figure 2

blocks in black are obstacles that agents could not step on and white ones are free spaces. The coordinate system is show as

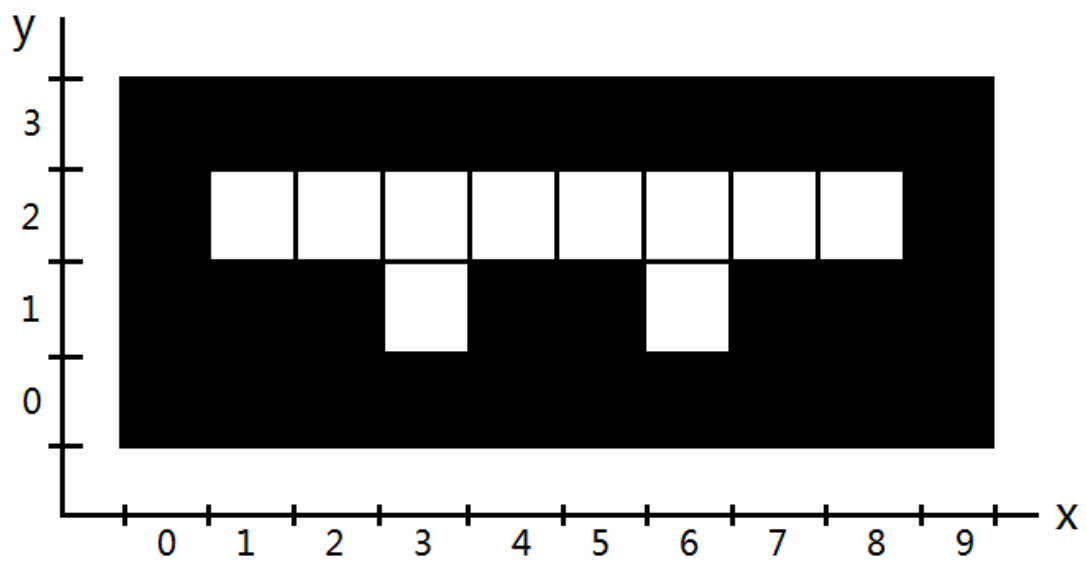


Figure 3

For agent 1, it starts at block $(x, y) = (3, 1)$, agent2 starts at $(6, 1)$ as shown in figure 1. The target of agent 1 is $(8, 2)$ and target of agent 2 is $(1, 2)$. Each time the agent reaches its goal will be immediately transported back to its start position, but the other agent will not be affected, as:

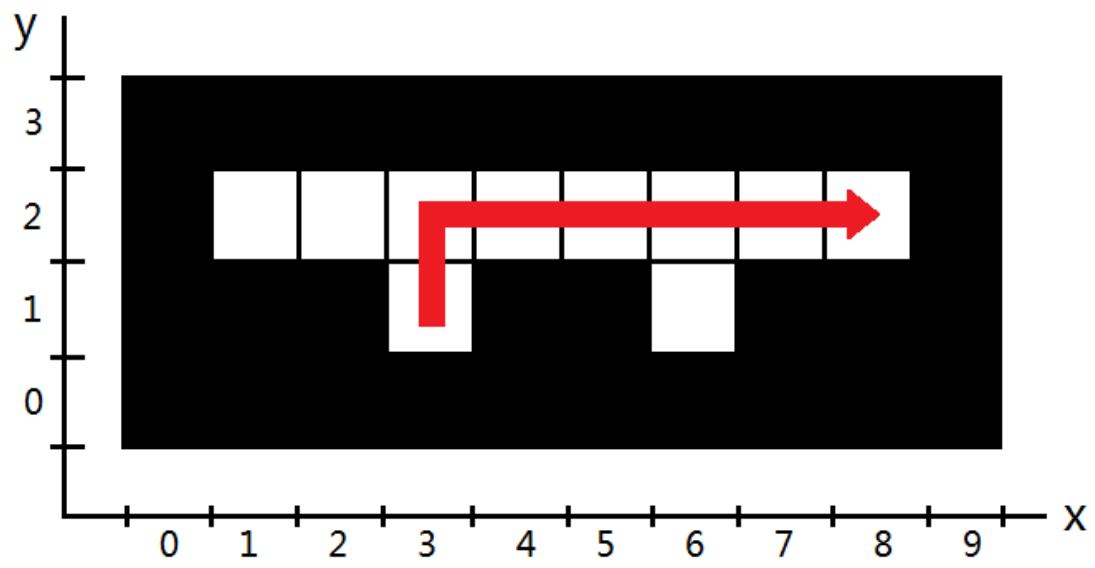


Figure 4

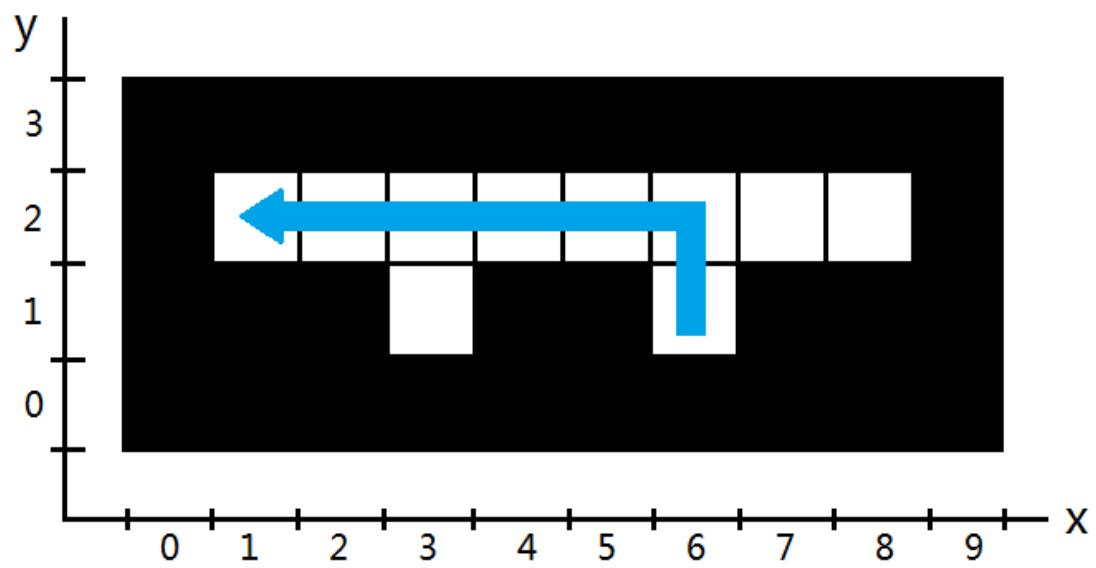


Figure 5

There are 5 possible action for the two agents

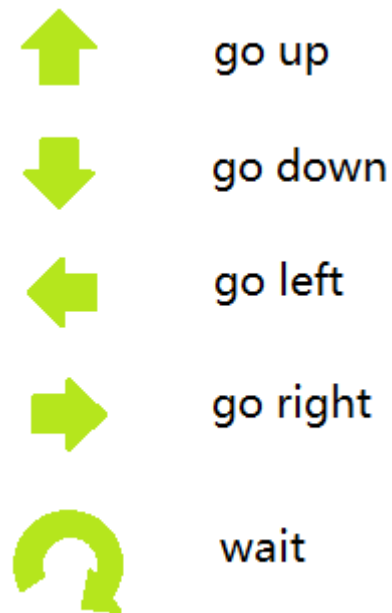


Figure 6

when the agent moves, it will move to the free space of 1 distance near it. However, when there is a block or the other agent is in its way, the action won't work. One can understand this by an example as

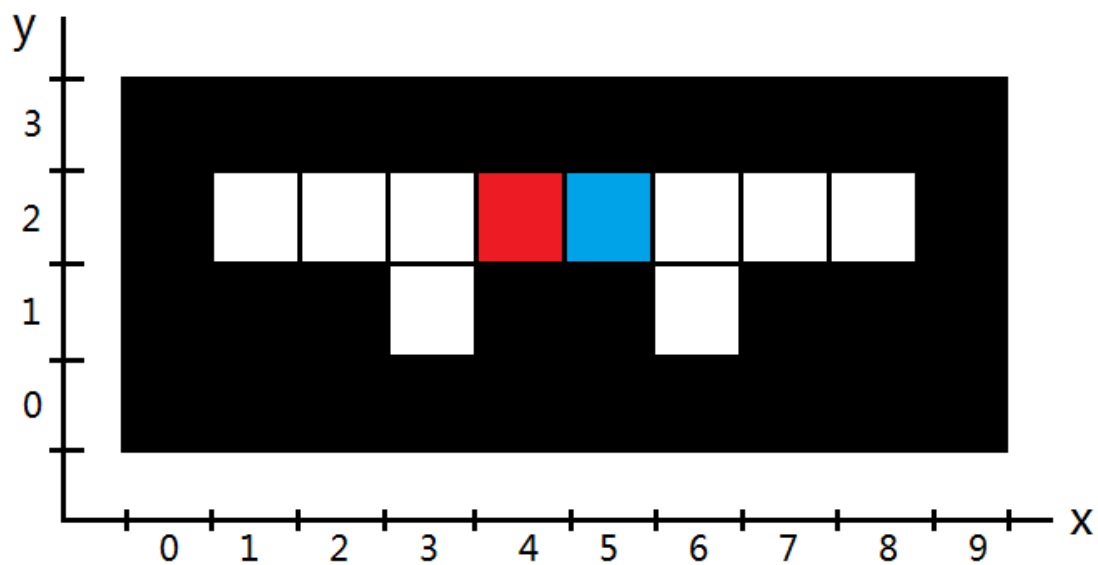


Figure 7

When agent 1 (red) is at (4,2) and agent 2 is at (5,2). As agent 1 wants to move right and agent 2 wants to move left, the result will be no one moving. This interference increases the challenge of the work as the maze is narrow, only when two agents are programmed in a very synchronized and coordinated way, the interference can be avoided.

The reward for each action taken is the same for both agents. Action "go up", "go down", "go left" and "go right" will return a -1 reward and agent reaches its corresponding target will be

rewarded for 100.

Agent 1 and two will only observe their local surroundings as 8 nearest positions

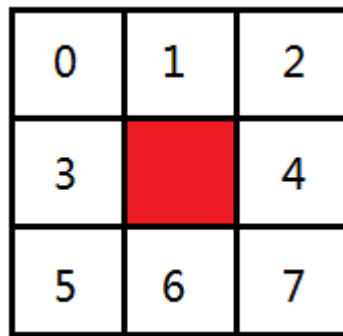


Figure 8

Or using a more global view as

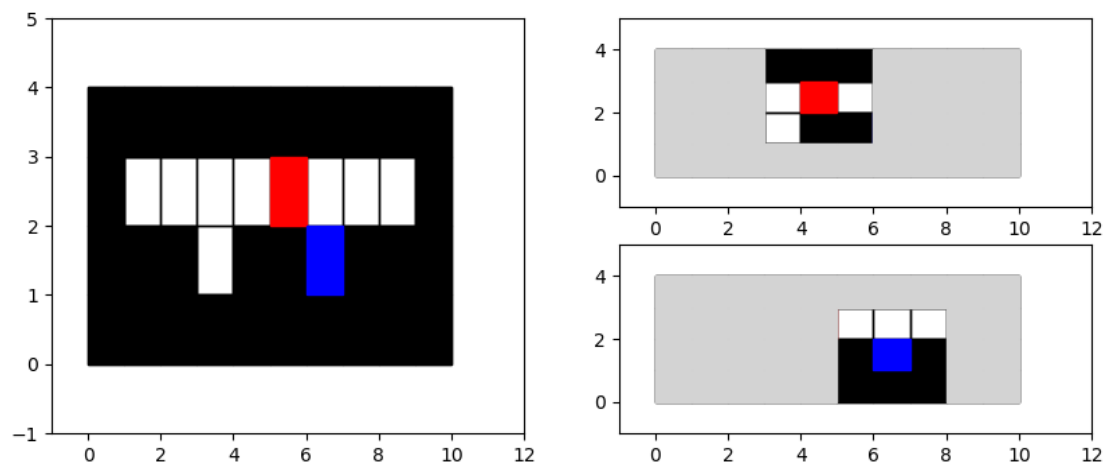


Figure 9

The global positions of agent1 (red) and 2 (blue) is shown as figure 9 left. The vision of agent 1 is shown in figure 9 up right, the light gray area is invisible to agent1. Also the figure 9 down right is the vision of agents 2.

Class Organization

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

The class has the following interfaces:

```
__init__(self):  
get_agt1_obs(self):  
get_agt2_obs(self):
```

```

step(self, action1, action2):
reset(self):
output_info(self):
test(self, action1_list, action2_list, max_iter, interval, if_plot):

```

```

__init__(self):

```

initialize the object, important variables are

```

agent1_pos
agent2_pos

```

These are programmed by a list as agent_pos=[3, 1]

```

get_agt1_obs(self):

```

These function return the current vision of agent 1 by returning a array of size (1, 8), indexed as

0	1	2
3		4
5	6	7

Figure 10

the free space will be valued as 0, block will be valued as 1 and the other agent will be valued as 2.

one example is the view of agent 1 is shown as

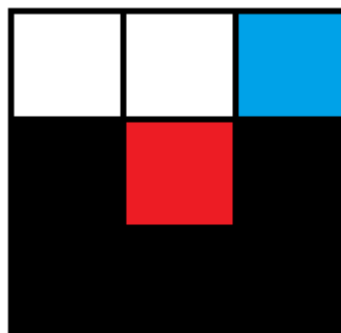


Figure 11

The returned observation vector will be [[0], [0], [2], [1], [1], [1], [1], [1]].

`step(self, action1, action2):`

The function update the environment by feeding the actions for agent1 and agent2. The actions are expressed by integers as






	go up	0
	go down	1
	go left	2
	go right	3
	wait	4

Figure 12

So the size of valid action is 5. The returned values of the function are reward, obs_1 and obs_2, which are the joint reward (sum of reward of agent 1 and agent 2) in the given step, the observation vector of agent 1 and agent 2.

`reset(self):`

This function move the two agent back to their start points as (3, 1) and (6, 1)

`output_info(self):`

print the current positions of agent 1 and agent2 in console

`test(self, action1_list, action2_list, max_iter, interval, if_plot):`

Giving the action array of size (1, max_iter), the function plots an animation of how the environment evolves under the given action list. Max_iter is the maximum length of frame of plotting and interval is the graph update interval in milisecond. if_plot is a bool variable, if it is True, the result will be plotted, else there will be not figure and all information will be printed in console. The printed figure is like:

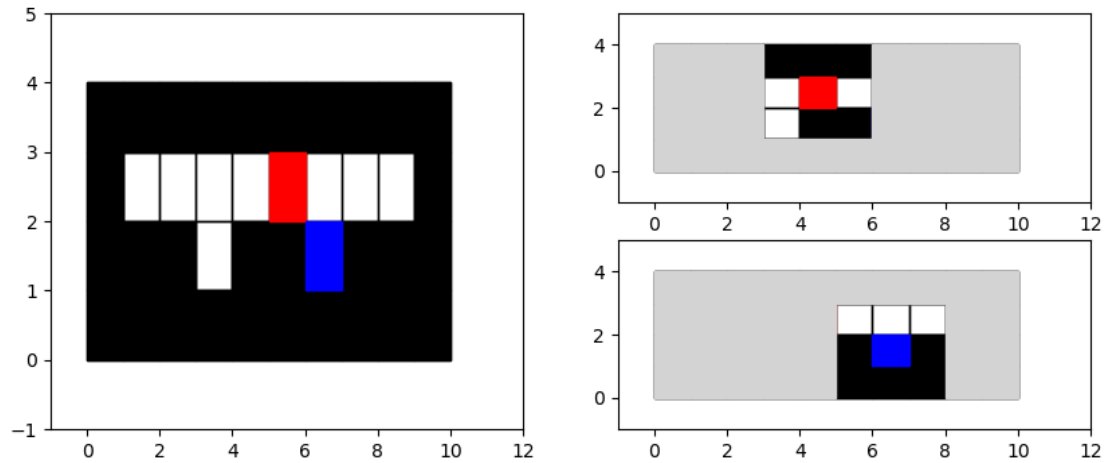


Figure 13

The global positions of agent1 (red) and 2 (blue) is shown as figure 9 left. The vision of agent 1 is shown in figure 9 up right, the light gray area is invisible to agent1. Also the figure 9 down right is the vision of agents 2.

Example

Here is an example using test function, and it is in "test_FindGoals.py"

```
from env_FindGoals import EnvFindGoals
import random
import numpy as np

env = EnvFindGoals()
max_iter = 1000
action1_list = np.random.randint(0, 4, size=[1, max_iter])
action2_list = np.random.randint(0, 4, size=[1, max_iter])
env.test(action1_list, action2_list, max_iter, 200, True)
```

In your implementation, you can forget the test() function and use step() function only.

```
from env_FindGoals import EnvFindGoals
import random
import numpy as np

env = EnvFindGoals()
max_iter = 100
for i in range(max_iter):
    print("iter= ", i)
```



```
env.step(random.randint(0, 4), random.randint(0, 4))
print("agent 1 is at ", env.agent1_pos)
print("agent 1 observation= ", env.get_agt1_obs())
print("agent 2 is at ", env.agent2_pos)
print("agent 2 observation= ", env.get_agt2_obs())
```