Environment "Single Agent Runs in Maze" Documentation

Author: Shuo Jiang (jiang.shuo@husky.neu.edu)
License: Northeastern University, Lab for Learning and Planning in Robotics(LLPR)

Introduction

The document introduces an environment for a single agent trying to find its 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.

Scenario Description

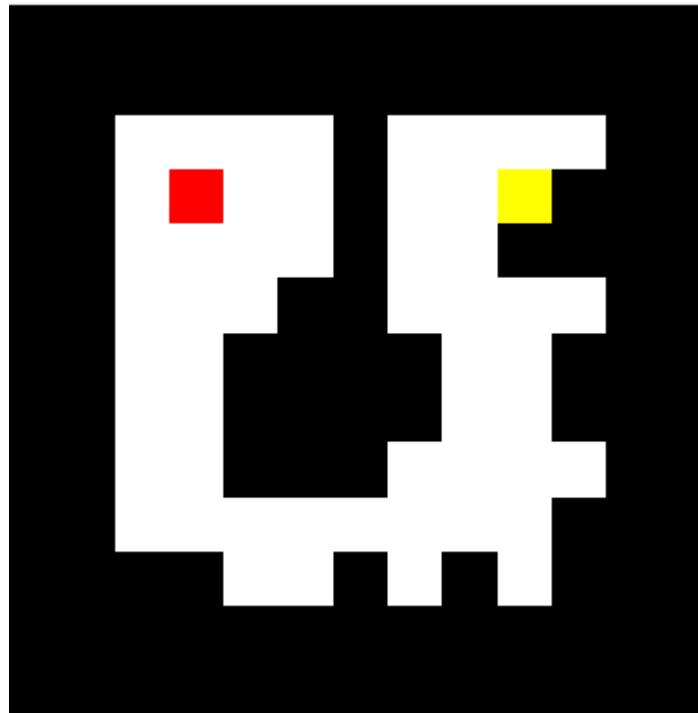The task of the environment is explained as below

Figure 1

The environment works like a maze, agent will run in the maze and find its targets. The agent will be denoted using color red, and will be shown as red rectangles in the maze like in figure 1. The Goal it tries to reach is the yellow block shown in the maze. Blocks in black are obstacles that

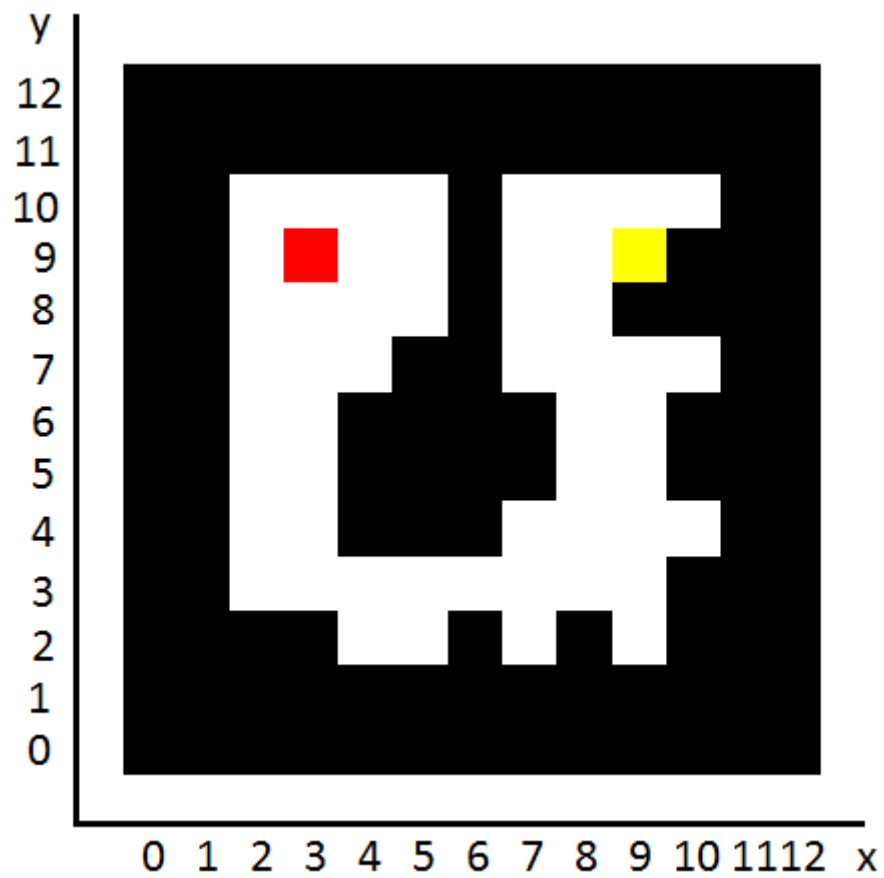agents could not step on and white ones are free spaces. The coordinate system is shown as

Figure 2

Agent starts at block (x, y)=(3,9) and the goal of agent 1 is at (9, 9). Each time the agent reaches its goal (step on it) will be immediately transported back to its start position, and gets reward:
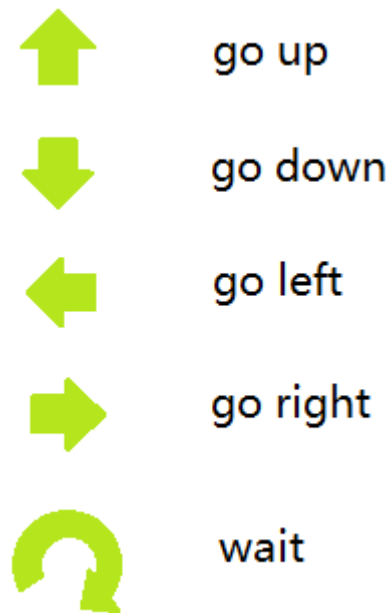
There are 5 possible action for the agent

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.

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 500. However if the agent bumps to the wall (for example, walk upwards but a wall block is at the place), the agent will be rewarded for -20.

Agent will only observe their local surroundings as 15 nearest positions as an image.
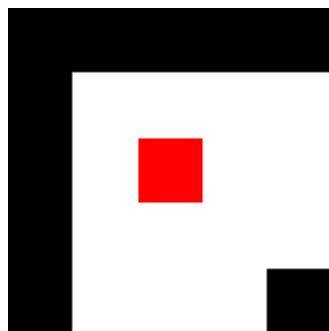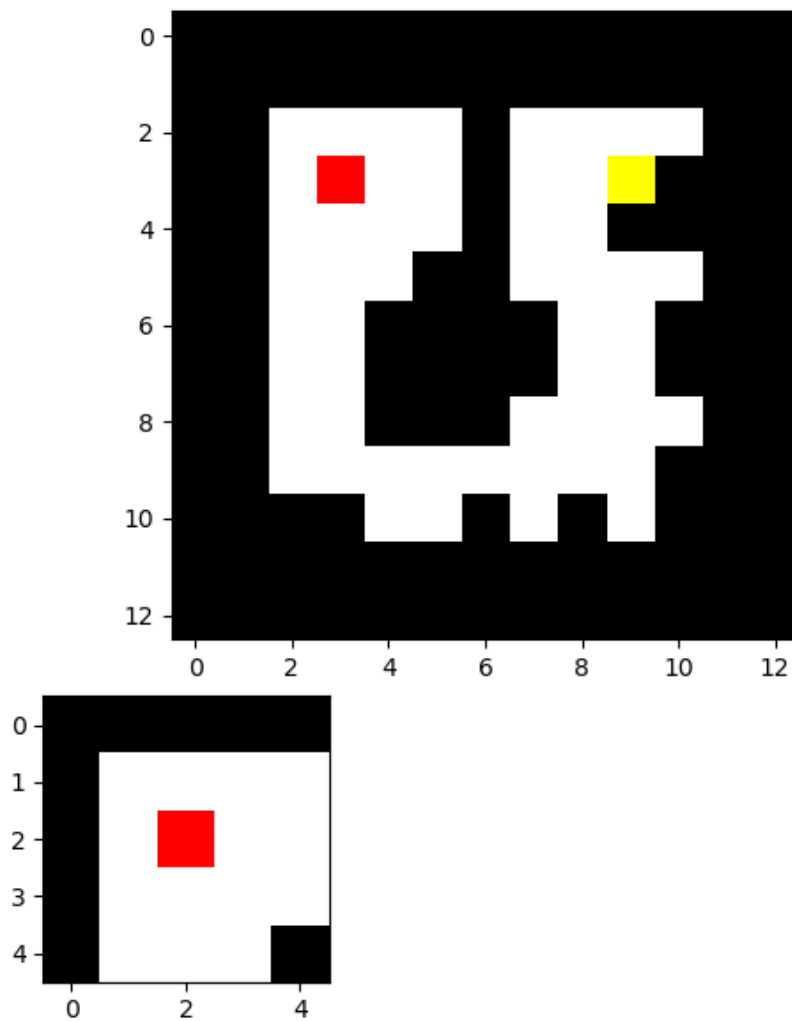

Figure 4

Or using a more global view as

Figure 5

The global positions of agent (red) is shown as figure 5 top. The vision of agent is shown in figure 5 left bottom.

Class Organization

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

The class has the following interfaces:

```python
__init__(self, is_po)
get_agt1_obs(self)
get_global_obs(self)
step(self, action1):
reset(self):
```

```
plot_scene(self):
```

```
__init__(self, is_po)
```
initialize the object, important variables are
```
agent1_pos
```
It can be set by a list as agent_pos=[3, 1]
`is_po` is a bool variable, if it is set as "true", means the environment is partially observable. The agent can only see the environment around it. Otherwise, the agent will see the whole environment.

get_agt1_obs(self):
These function return the current vision of agent 1 by returning a array of size (5,5, rbg channel) or (5,5,3) as a image. agent 1 will be centered as at the center using a red rectangle.

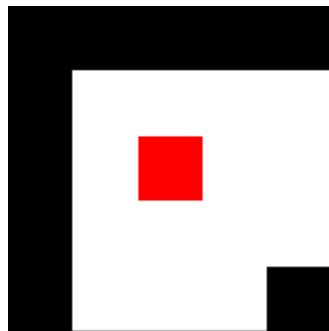one example is the view of agent 1 when is_po is set as "true" is shown as

Figure 6

```
step(self, action1):
```
The function update the environment by feeding the action for agent1. The actions are expressed by integers as

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

Figure 7

So the size of valid action is 5. The returned values of the function are reward, obs_1, which are the reward in the given step, the observation image of agent 1.

reset(self):
This function moves the agent back to start point as (3, 9).

plot_scene(self)
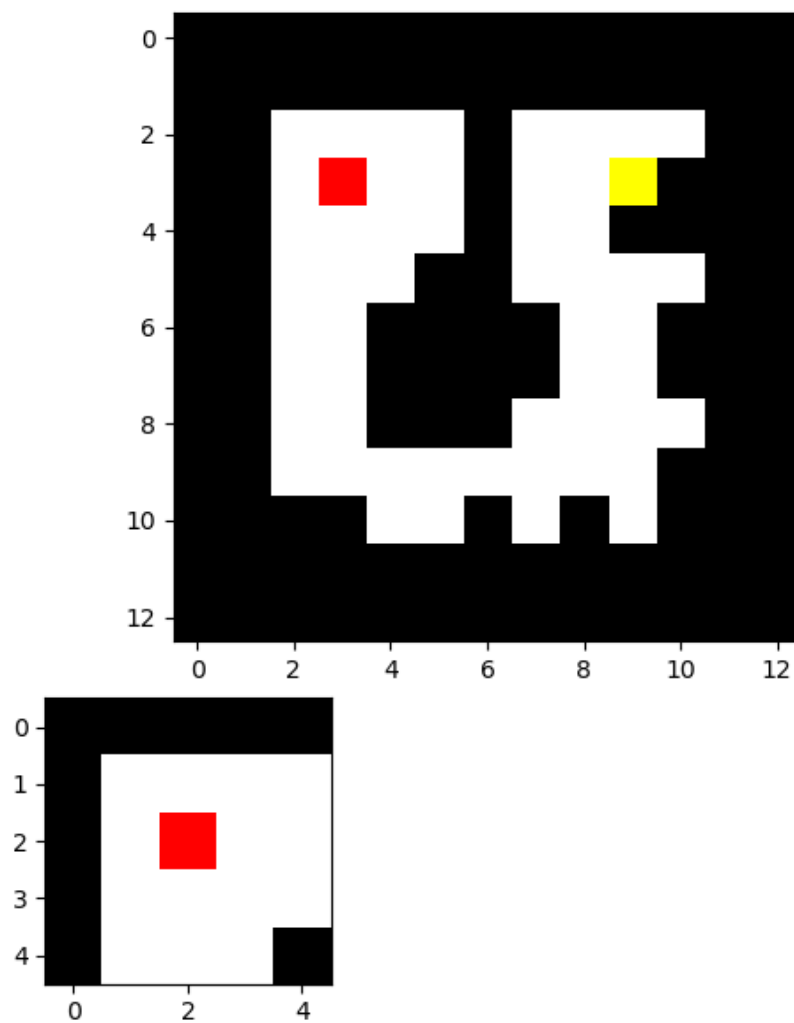This function plots the current state of the whole environment, and uses subplot to show the view agent

Figure 8

The global positions of agent1 (red) is shown as figure 8 top. The vision of agent 1 is shown in figure 13 bottom left.

Example

Here is an example using test function, and it is in "test_SingleMaze.py". The action is random chosen, click and run.

```python
from env_SingleMaze import EvnSingleMaze
import random

env = EvnSingleMaze(True)
max_iter = 10000
for i in range(max_iter):
    a_1 = random.randint(0, 3)
```

```python
print("action", a_1)
env.plot_scene()
reward_1, obs_1 = env.step(a_1)
if reward_1 > 0:
    print("iter= ", i)
    print("find")
    print("agent 1 at", env.agt1_pos)
```