Environment "Rescue" Documentation

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

Introduction

The document introduces an environment for multi agent study as two agents are trying to rescue people 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

One important robot implementation scenario is that robots are deployed in a completely unknown environment and having some task to finish. In such case, model-free planning is hard to be done because robot lacks prior knowledge of the environment. Knowledge must be gathered and accumulated from observation, and robots have to update their policies as the model is updated with time.

In this environment, I set up a squared area as a ruin after earthquake. There are some people trapped in the ruin that are waiting to be rescued. There are two entries of the area, which are safe points of the environment, and one or more robots start from the safe points, enter the area, find peoples in the area, pick them up and carry them back to the safe points.

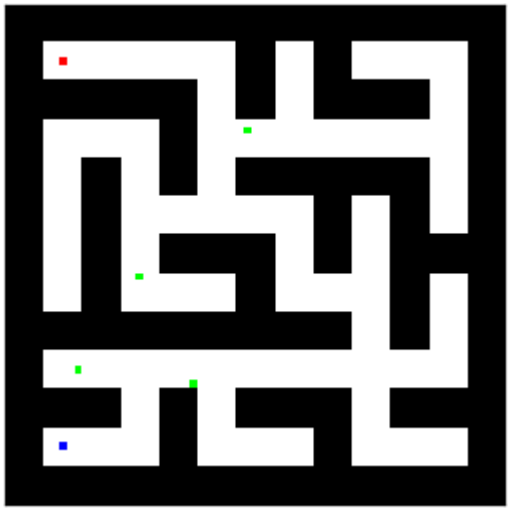

Figure 1

The ruin is as a maze, which can be generated by feeding in a random seed, that each time a

maze with random shape will be generated. There will be robots deployed initially at either up left corner or bottom left corner of the maze. As shown in figure 1, in this case, two agents are set and marked with red and blue dots. There are some number of humans in the maze at random places marked with green dots. The task of robots is trying to find the humans in the maze and carry them back to safe points.

Coordinate System

The coordinate system is as

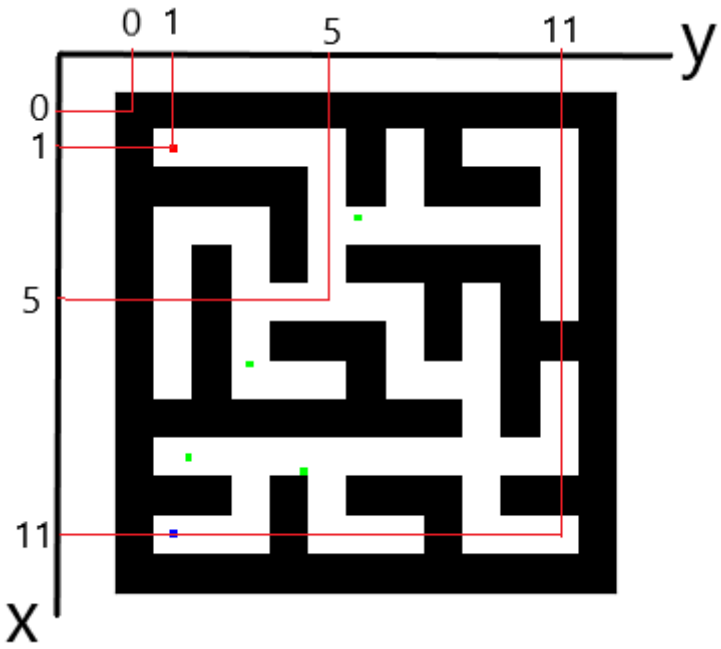

Figure 2

The maze is composed by blocks, robots and humans are placed in the white blocks in the maze (free spaces). In this case, the maze is initialized for 13 blocks along x and y axes. One robot is placed at (1, 1) and another robot will be placed at (11, 1).

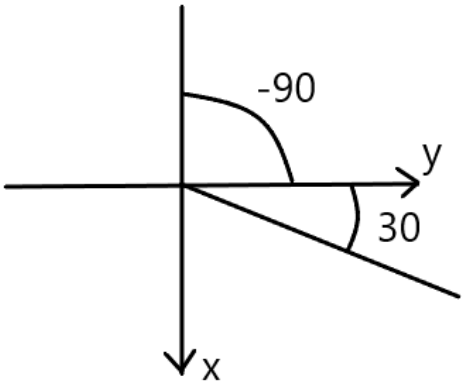

Figure 3

Angle in this environment is defined as the degree with respect to positive direction of y axis. For example, 30 degrees and -90 degrees are shown in Figure 3.

State Space

Each robot has the following attributes:

- [x, y] positions
- Orientation
- If it is carrying human (with humans' id), -1 if not carrying human. Or human's ID its carrying
- Id, integer from 0

Each agent will be given an ID which is an integer from 0 upwards. If more than one robot are initialized, robots with even ID will be set at (1, 1) and odd ID will be set at (maze size – 2, 1), maze size is the number of blocks on the edge of environment. X, y positions are continuous (double). Orientation of the robots are defined as in coordinate system part.

Each human has attributes:

- [x, y] positions
- Carried by which robot (the robot's id)

Action Space

For each agent, the action of each step is of four dimensions.

- Movement along x dimension (double), range [-0.5, 0.5]
- Movement along y dimension, range [-0.5, 0.5]
- Rotation (clockwise is positive), range [-10, 10]
- Pick/drop human (integer). 0, doing nothing. 1, pick human. 2, drop human.

The four dimensions form an action vector which is a list of four elements action = [d_x, d_y, omega, if_pick]. For all robots, there will be a list of lists above action_list = [action1, action2, …].

Observation Space

Each robot can only observe 90 degrees range around its orientation. However this vision will be occluded by walls (black blocks). The largest distance will be about 3 blocks. The observation is like:

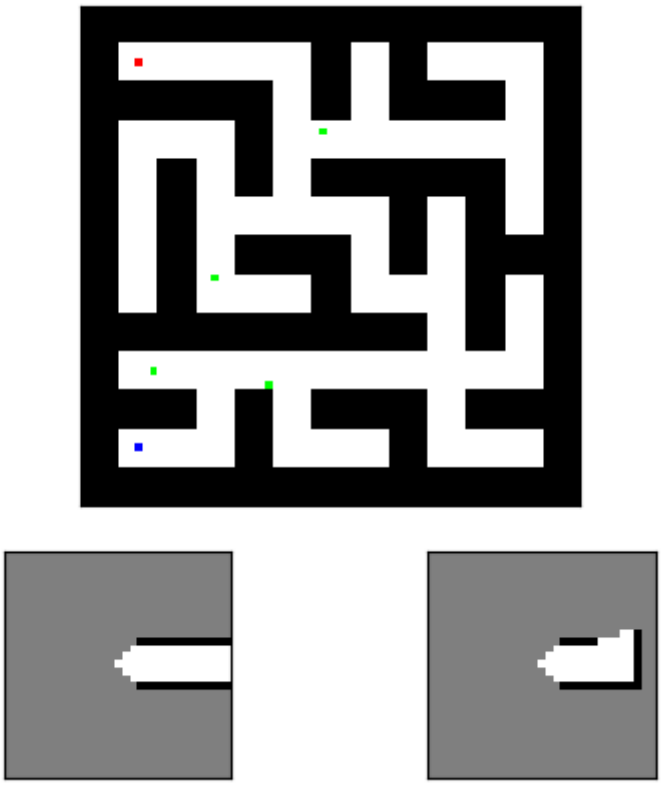

Figure 4

So one example observation is shown in figure 4. Left is the observation of robot 0, at (1, 1). Right is the observation of robot 1, at (11, 1). The unseen area is shown in light grey color. The observation is a numpy array of size (31, 31, 3), valued from 0 to 1.

Class Organization

The environment is programmed in python 2/python 3 and has very simple interfaces. The file contains the environment is "env_rescue.py" and a class " EnvRescue" is defined.

The class has the following interfaces:

__init__(self, map_size, N_agent, N_human, seed)
step(self, action_list)
get_agt_obs(self, agt_id)
get_obs(self)
get_global_obs(self)
reset(self)
get_agent_pos(self, agt_id)
get_agent_ori(self, agt_id)
get_human_pos(self, human_id)
get_carry_human_id(self, agt_id)

get_picked_by(self, human_id)
is_pos_free(self, pos)
is_human_safe(self, human_id)
get_rescued_human_num(self)
is_episode_finish(self)

initialize the object, important variables are
map_size: size of map
N_agent: number of robots
N_human: number of humans

__init__(self, map_size, N_agent, N_human, seed)
Initialize function. Map_size is an integer and should be an odd number no smaller than 3. N_agent and N_human are also two numbers of robots and humans. Seed is an integer which controls the shape of maze. With different seed, the maze shape will be different, but the initial positions of robots are the same. The seed only controls the shape of maze but not the initial positions of humans. In each episode, the humans shall be initialized at different positions.

step(self, action_list)
Step function, should be called at each iteration. Input is a list of action list of each robots consists of 4 action dimensions. The function has no return.

get_agt_obs(self, agt_id)
Returns the observation (image) of given robot id of size (31, 31, 3) as

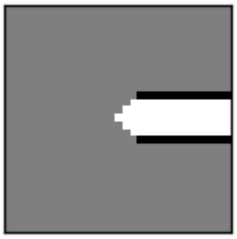

Figure 5

get_obs(self)
returns a list contains observations from all robots.

get_global_obs(self)
return the full observation of the global view as

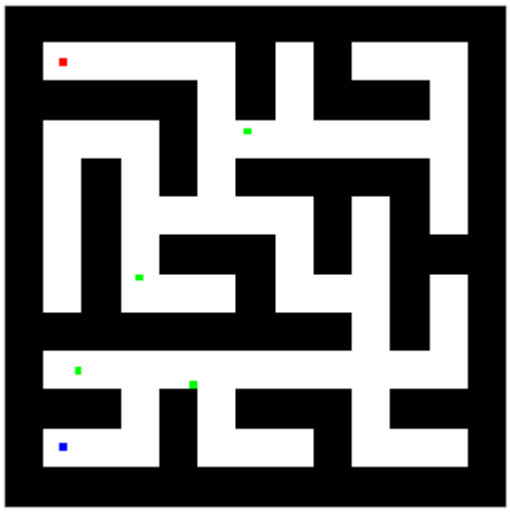

Figure 6

reset(self)
Reset the robots to corresponding safe points and humans at random places.

get_agent_pos(self, agt_id)
return a list of x, y position of given agent id, as [2.63, 5.48]

get_agent_ori(self, agt_id)
return a double value of agent orientation.

get_human_pos(self, human_id)
return a list of x, y position of given human id, as [2.63, 5.48]

get_carry_human_id(self, agt_id)
if robot is carrying a human, return the human's ID it is carrying. Otherwise return -1.

get_picked_by(self, human_id)
return the Robot ID the human is picked by. Otherwise return -1

is_pos_free(self, pos)
given a position as [x, y], return if it is free space (True) or wall (False)

is_human_safe(self, human_id)
given a human's ID, if it is in the block of either one safe point, return true. Otherwise return false

get_rescued_human_num(self)
return the number how many humans are in safe points.

is_episode_finish(self)
return True only when all the humans are in the safe points and all the robots are in the safe

points as well.

Example

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

```
Import time
from env_rescue import EnvRescue
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import random

env = EnvRescue(13, 2, 4, random.randint(0, 10000))
max_MC_iter = 10000
for MC_iter in range(max_MC_iter):
    print('iter', MC_iter)
    if MC_iter%20 ==0:
        fig = plt.figure(figsize=(5, 5))
        gs = GridSpec(3, 2)
        ax1 = fig.add_subplot(gs[0:2, 0:2])
        plt.xticks([])
        plt.yticks([])
        ax2 = fig.add_subplot(gs[2, 0:1])
        plt.xticks([])
        plt.yticks([])
        ax3 = fig.add_subplot(gs[2, 1:2])
        plt.xticks([])
        plt.yticks([])
        ax1.imshow(env.get_global_obs())
        ax2.imshow(env.get_agt_obs(0))
        ax3.imshow(env.get_agt_obs(1))
        plt.show()

    action1 = [random.random()-0.5, random.random()-0.5, 20*(random.random()-0.5), 0]
    action2 = [random.random()-0.5, random.random()-0.5, 20*(random.random()-0.5), 0]
    action_list = [action1, action2]
    print('agent 0 is at',env.get_agent_pos(0))
    print('agent 1 is at',env.get_agent_pos(1))
    env.step(action_list)
```

References

The map generation part uses the Kruskal algorithm from
https://github.com/SilverSoldier/Maze-Python