Environment "Navigation" Documentation

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

Introduction

In this environment, an agent should approach a given goal position in a 2D squared area. 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

In this environment, an agent should approach a given goal position in a squared area. The scenario looks like

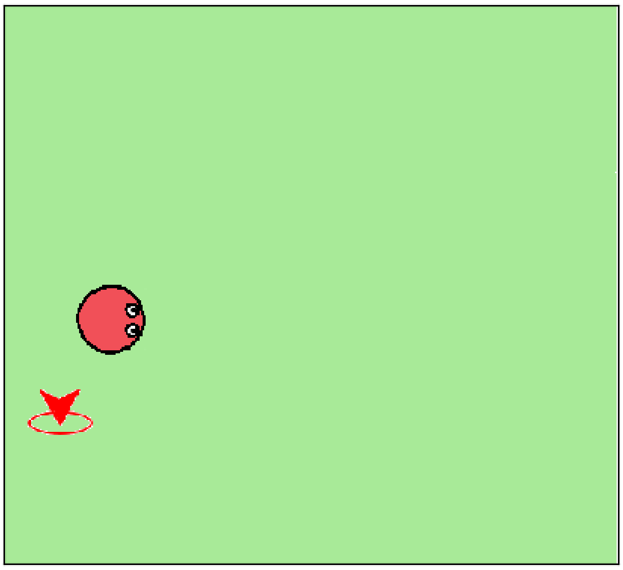
Figure. Scenario

The agent should approach the given target marked with red arrow. Once it is close enough to the goal position, a new goal position will be generated randomly in the area. Each time step, the agent will get a reward depending on the distance between its position and the position of goal, which is

$$reward = \frac{-distance}{500}$$

Coordinate system

The coordinate system is defined as

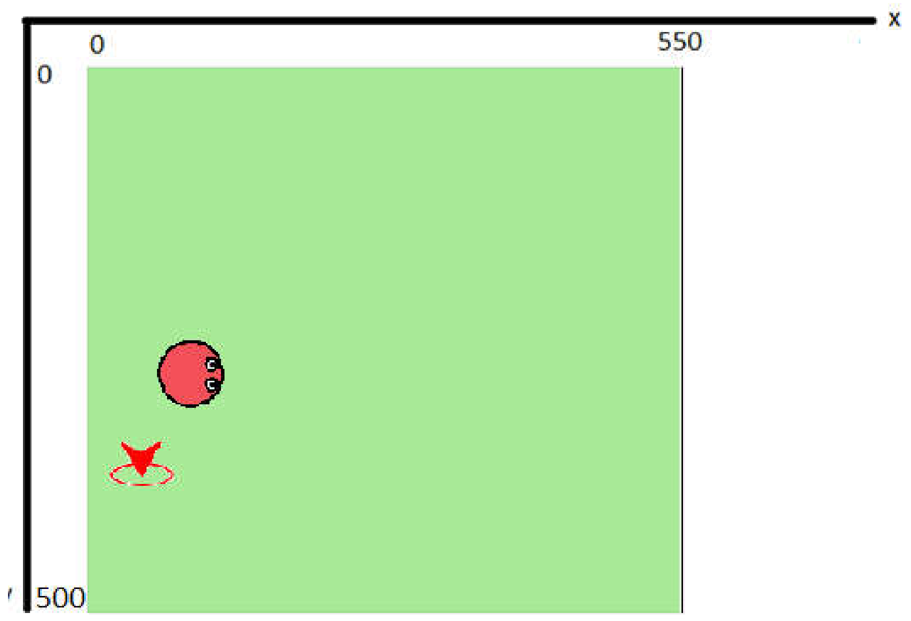

Figure. Coordinate system

So if you want to relocate the players manually, please keep the players in x [0, 550] and y [0, 500] range, otherwise you will not see the players.

Agent

The agent's state is defined as the aggregation of Cartesian position in 2D, velocity, orientation and angular velocity. The position and velocity are defined in the coordinate system illustrated before.

Orientation is defined as the angle between orientation vector and vector [1, 0], from -180 degree to 180 degree.

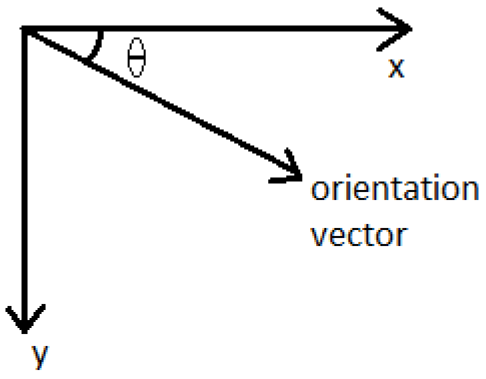

Figure. Orientation of player

The angular velocity is defined in degree. And clockwise rotation stands for positive angular velocity and anti-clockwise as negative.

Action Space

The action is given as a list of 3 elements, as [1, 4, 2]. The first two elements as [1, 4] defines a vector of the velocity of agent in the coordinate system. There two elements should be in range [-20, 20]. The third element is the angular velocity which is also bounded in [-20, 20], but in degree. Positive value of the third element means clockwise rotation, negative means anti-clockwise.

Observation

What can players observe is crucial in implementing learning algorithms. In this environment, we use full observation.

function `get_global_obs()` will return a numpy array in RGB image form of size (500,550,3). It is a fully observable setting.

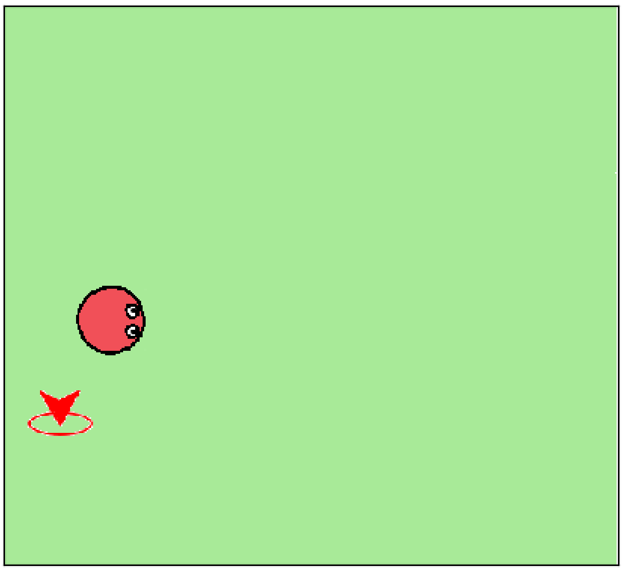

Figure. Full Observation

Class Organization

The class " EnvNavigation" and "Runner" is defined.

There are bunch of functions can be called, but only some of them can be useful to users. I will pick some for introduction.

Class Player

Member variables

```
self.pos = [103., 241.]          # position of agent
self.vel = [0., 0.]             # velocity of agent
self.theta = 0                  # rotation of agent, based on [1, 0],
                                bounded in -180-180
self.omega = 0.                 # angular velocity of agent in degree,
                                positive stands for anti-clockwise
self.max_speed = 30             # max running speed of agent
self.check_radius = 25          # if distance between target and agent is
                                smaller than this value. a new target will
                                be initialized
self.target_pos = [103., 241.]  # position of target
```

Member functions

users do not have to call any member functions in this class.

Class EnvNavigation

Member functions

`get_global_obs()`

# return an image of full observation of size (500,550,3)

`step(action)`

# to be called at each time step. Update the system. Action should be a (1, 3) list of three floats. It returns a reward as a float.

Example

Here is an example using test function, and it is in "test_Navigation.py". The action is randomly chosen for the agent, click and run.

```
from env_Navigation import EnvNavigation
import matplotlib.pyplot as plt
import random

env = EnvNavigation()
fig = plt.figure()
max_MC_iter = 2000
plt.xticks([])
plt.yticks([])
for MC_iter in range(max_MC_iter):
    print(MC_iter)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(env.get_global_obs())

    # add random velocity
    reward = env.step([20*(random.random()-0.5), 20*(random.random()-0.5), 
20*(random.random()-0.5)])
    plt.pause(.04)
    plt.draw()
```