Task 3

R O A D N O T T A K E N

The white walkers have successfully broken through the wall, thanks to Viserion. Now they want to reach Winterfell as soon as possible. Being the night king, you are expected to come out with a short and smooth path (they find it difficult to change directions) for the walkers. Thankfully, you know about **RRT*-Connect.**

But there is another problem. Daenerys of the House Targaryen, the First of Her Name, The Unburnt, Queen of the Andals, the Rhoynar and the First Men, Queen of Meereen, Khaleesi of the Great Grass Sea, Protector of the Realm, Lady Regent of the Seven Kingdoms, Breaker of Chains and Mother of Dragons is moving in circles on Drogon. In order to reach Winterfell, you must avoid her and replan your path accordingly.

(The analogy of this to the task is of no importance, just like the final season.)

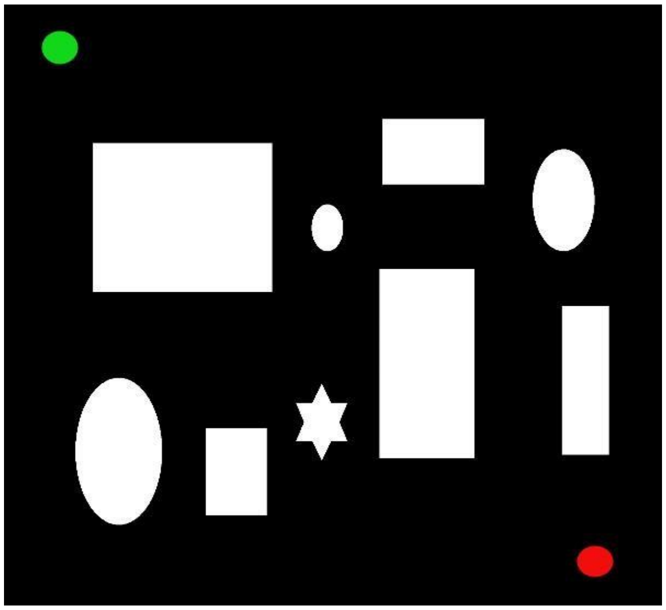
Description

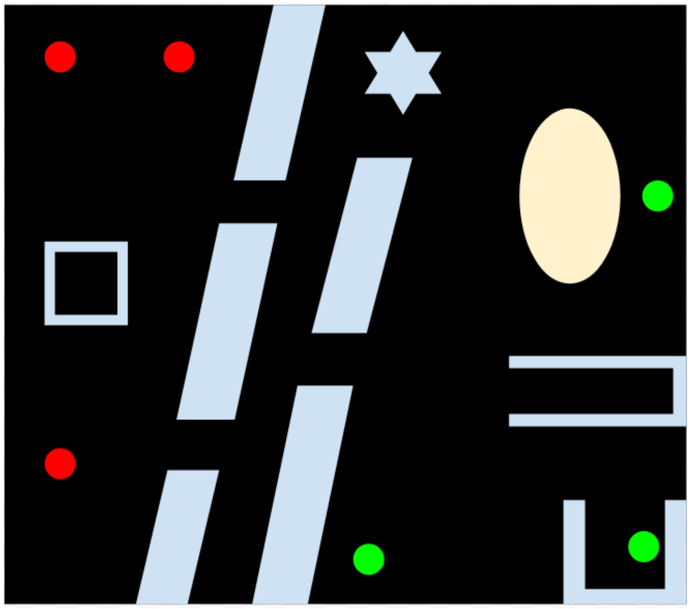
Part-I :

There are various path planning algorithms like Dijkstra, A*, RRT, RRT* with each having its own pros and cons. RRT*-Connect is a variant of RRT* (Rapidly-exploring Random Trees) that generates the trees from both source and destination which progressively grow towards each other through the use of a simple greedy heuristic thus leading to a faster result.

Use the RRT*-Connect algorithm to generate a non-deterministic path from the **source** to the **destination**.

Your task, should you choose to accept it, is to generate a valid path using RRT*-Connect that is smooth and short (not necessarily the shortest). The generated path can be shown on OpenCV GUI. Use the image given below.





Generate a "valid" path on the first image, and then generalize on the second image. In the second image, we've introduced three **sources** and three **destinations.**

You should be able to generate a valid path between **any combination of the source and destination. Note: In the second image, the destinations are green here and sources are red (color coding is opposite to the first image)**

You may/may-not need to perform image operations like transformations(scale up, down etc.)

Part-II :

You're required to move the turtle in turtlesim(ROS), on the path that you've achieved.

A simple way to learn the basics of ROS is to use the turtlesim simulator that is part of the ROS installation. The simulation consists of a graphical window that shows a turtle-shaped robot. The turtle can be moved around on the screen by ROS commands, which can be based on input feed or programmed.

Once you have the path generated in Part-1, **discretize the path according to your needs**, select a specified number of points (your choice) from the path that leads you to the destination from the source**.** Scale these points down to match the dimensions of the ROS Turtlesim simulator. The problem statement requires you to move the turtle along the path via these points, in order to reach the destination. You need to move the turtle using PID. Use feedback to display the position of the turtle on the image.

Part-III :

From the 1st part of the task, you have a path from the start to the endpoint. Now, spawn another turtle that moves in a circle of radius 1/4 of the simulator window length with the center of the circle the same as the center of the window.

Start moving your initial turtle on the path generated and avoid the other turtle by replanning the path in case there is going to be a collision.

Note:

The code should be well structured. It should not be repetitive or redundant. Achieving the goal of the task is indeed important but definitely not at cost of your code design and readability.

Task 4

AGENT PENDULUM

By definition, in reinforcement learning, an agent takes action in the given environment either in a continuous or discrete manner to maximize some notion of reward that is coded into it. In this task, you will be required to train an agent - a pendulum in this case - using an algorithm called Proximal Policy Optimisation or PPO in the OpenAI Gym environment. Your task is to implement the algorithm referring to the OpenAI documentation while making some changes upon the sample code provided.

Setup

Make sure that you have Python 3.6/3.7 and git installed. If you don't, first download them.

Download and install OpenAI Gym:

```
$ git clone https://github.com/openai/gym.git
$ cd gym
# install gym in developer mode
$ pip install -e .
```

Download and install OpenAI SpinningUp:

```
# IF you're in the gym directory, go one level up
$ cd ..

# ONLY IF you're on Ubuntu
$ sudo apt-get update && sudo apt-get install libopenmpi-dev
```

# ONLY IF you're on Mac

```
$ brew install openmpi

$ git clone https://github.com/openai/spinningup.git

$ cd spinningup

$ pip install -e .
```

I. Verify installation by training a simple PPO agent

```
$ python -m spinup.run ppo --hid "[16,16]" --env Pendulum-v0 --exp_name
installtest --gamma 0.999

# THIS WILL TRAIN FOR A QUITE A FEW MINUTES (~30min on my Mac)
# BUT please stop it after max 3min. This is just to verify everything works

# Look at the (partially) learned policy in action

$ python -m spinup.run test_policy data/installtest/installtest_s0

# Plot the reward over time

$ python -m spinup.run plot data/installtest/installtest_s0
```

II. Understand the environment

- While you're doing the next few steps, train a PPO agent on this environment, like done above, but give it 10-15min, and change installtest to pendulum or something else you like
- Create a new Python script
- Implement the basic gym loop:

```python
import gym
env = gym.make("Pendulum-v0")

while True:
    obs = env.reset()
    done = False
    while not done:
        # in practice the action comes from your policy
        action = env.action_space.sample()
```

```
obs, rew, done, misc = env.step(action)
# optional
env.render()
```

You should inspect the observations dimensions and action dimensions by printing env.observation_space and env.action_space.

- Look at the environment definition and understand what the observations mean: `gym/gym/envs/classic_control/pendulum.py`
  - in particular, look at the `step` function, line 32 and where the observations come from (`get_obs` function, line 57)
- What do the different components in the observations stand for? What's `self.state[1]/thetadot` in the environment file, what does it mean?
- Is this discrete action ("turn left/right") or continuous action ("turn left/right at [0-1]x speed")?

III. Modify the reward function

- At this point, look at the trained PPO policy
- If it didn't work, start it again but with more training epochs:

  ```
  $ python -m spinup.run ppo --hid "[32,32]" --env Pendulum-v0 --exp_name pend2 --gamma 0.999 --epochs 200
  ```
- The reward calculation ("costs", line 42), has different components. What do they stand for and how are they weighted?

Task-5 : Nao Robosoccer

In the following task, you need to write a one-on-one strategy for a Two Humanoid Nao-bot, one of which is required to take the ball from its own goalpost to the opponent goal post dodging the opponent in between. For this, you will need to go through UT Austinvilla codebase, the README and understand how the joint torques are passed to the Nao-bot and where the code for the basic strategy exists.

Setup and Explanation:

The following should happen when the code is executed. One bot(attacker) should appear in the simulator at their own goal position. When the match starts the bot should approach the ball and dribble it towards the opponent goal post. While doing so it will encounter the opponent bot(defender). The opponent bot should defend the ball, and stop the attacker from scoring a goal. You need to think of an appropriate strategy that will enable the attacker to avoid the defender while keeping the ball in its possession and score a goal under the given circumstances, and the defender to transfer the possession to itself.

You need to write separate codes for the bots. They can be generated by following the steps below :

- Create two copies of Utaustinvilla/master and open start.sh file in an editor then change the number of players in the script to 1 (from 11) and finally build it in a similar way you did for the original copy. (`cmake .` then `make`)
- In one of the copies of utaustinvillamaster rename the folder to an attacker and other to the defender. Open the start.sh file in this folder and change the team names.
- In these different copies, you are required to write your strategy. Every time you edit your strategy you have to recompile your solution using `make -j4` or `j8` command.

NOTE:
It may so happen the bots would appear to move too slow in your laptop i.e due to low server speed(<40%). You can check your server speed on the top left corner of Roboviz. This is as your Graphics Card Drivers are not installed on Ubuntu. If this happens you need to install appropriate Graphics Drivers for your laptop.