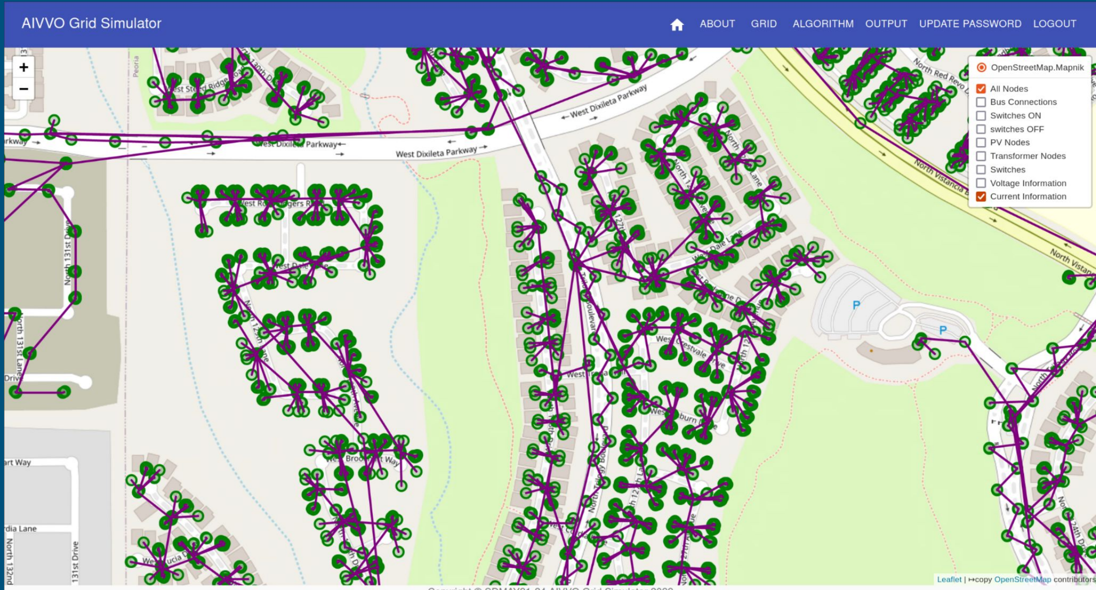
AI-VVO sdmay22-36 Weekly Update #11

4/7/2022 - 4/14/2022

Front-end (This Week)

- Implemented current information as layerControl component
- Added more documentation to frontend files



● OpenStreetMap.Mapnik
☑ All Nodes
□ Bus Connections
□ Switches ON
□ switches OFF
□ PV Nodes
□ Transformer Nodes
□ Switches
□ Voltage Information
☑ Current Information

Front-end (Next Week)

- Finish up documentation on frontend
- Implement tutorial on how to run application for next team
- Implement video recording for code base walkthrough
- Create final poster, report and prepare for presentation

Back-end (This Week)

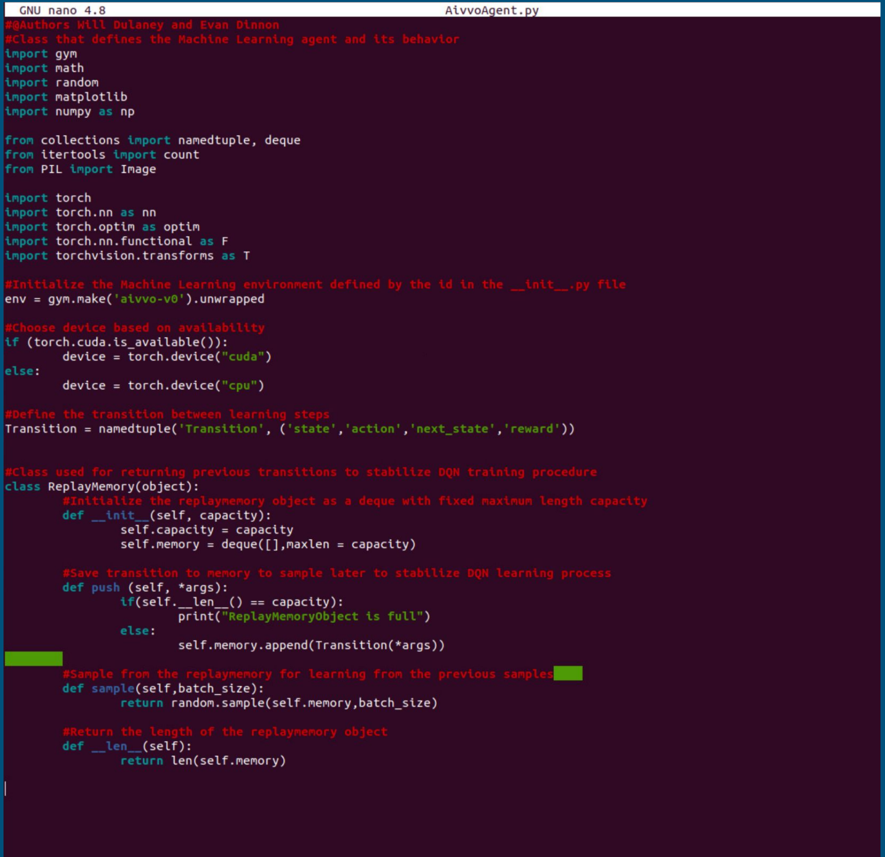
- Added more unit tests to help make sure django functions are working correctly
- Worked towards updating documentation so that next team has less trouble understanding the project.

Back-end (Next Week)

- Finish up documentation on backend
- Create final poster, report and prepare for presentation

Machine Learning (This Week)

- Began writing the RL Agent class used to step through the environment class.
- Started on the deep Q learning algorithm following pytorch tutorials
- Was able to create the environment from class written last week



```
#@Authors Will Dulaney and Evan Dinnon
#Class that defines the Machine Learning agent and its behavior
import gym
import math
import random
import matplotlib
import numpy as np

from collections import namedtuple, deque
from itertools import count
from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as T

#Initialize the Machine Learning environment defined by the id in the __init__.py file
env = gym.make('aivvo-v0').unwrapped

#Choose device based on availability
if (torch.cuda.is_available()):
        device = torch.device("cuda")
else:
        device = torch.device("cpu")

#Define the transition between learning steps
Transition = namedtuple('Transition', ('state','action','next_state','reward'))


#Class used for returning previous transitions to stabilize DQN training procedure
class ReplayMemory(object):
        #Initialize the replaymemory object as a deque with fixed maximum length capacity
        def __init__(self, capacity):
                self.capacity = capacity
                self.memory = deque([],maxlen = capacity)

        #Save transition to memory to sample later to stabilize DQN learning process
        def push (self, *args):
                if(self.__len__() == capacity):
                        print("ReplayMemoryObject is full")
                else:
                        self.memory.append(Transition(*args))

        #Sample from the replaymemory for learning from the previous samples
        def sample(self,batch_size):
                return random.sample(self.memory,batch_size)

        #Return the length of the replaymemory object
        def __len__(self):
                return len(self.memory)
```

Machine Learning (Next Week)

- Finish reinforcement learning algorithm
- Write final report
- Create final poster
- Prepare for final presentation