Traffic Light Optimization using Reinforcement Learning

MSCA Course Project Presentation

Shogo Nakano/Darren Colby

University of Chicago

5/25/2023

Table of Contents

1. Introduction
2. Method
3. Results
4. Conclusion

Motivation

- In the United States, the ever-growing issue of traffic congestion has become a significant concern, affecting millions of daily commuters and causing a substantial negative impact on the economy and environment.

Motivation

- Most congested metro areas in 2022:[1]
  1. **Chicago (155 hours lost)**
  2. Boston (134 hours)
  3. New York (117 hours)
  4. Philadelphia (114 hours)
  5. Miami (105 hours)
- Chicago is the most congested area, we lost **155 hours** per year!!

[1] https://www.thestreet.com/money/30-cities-with-the-worst-traffic-in-the-us

Motivation

- Reinforcement learning for traffic light control: an innovative and promising solution.

Implementation

- **SUMO-rl**: library for simulating traffic situations[1].
- SUMO is open-source software for simulating traffic situations.
- **stable-baseline3**: library for implementing reinforcement learning algorithms.

Summary of the model

- Agent: Traffic Lights
- Environment: two way single intersection
- Algorithm: DQN

Summary of the model

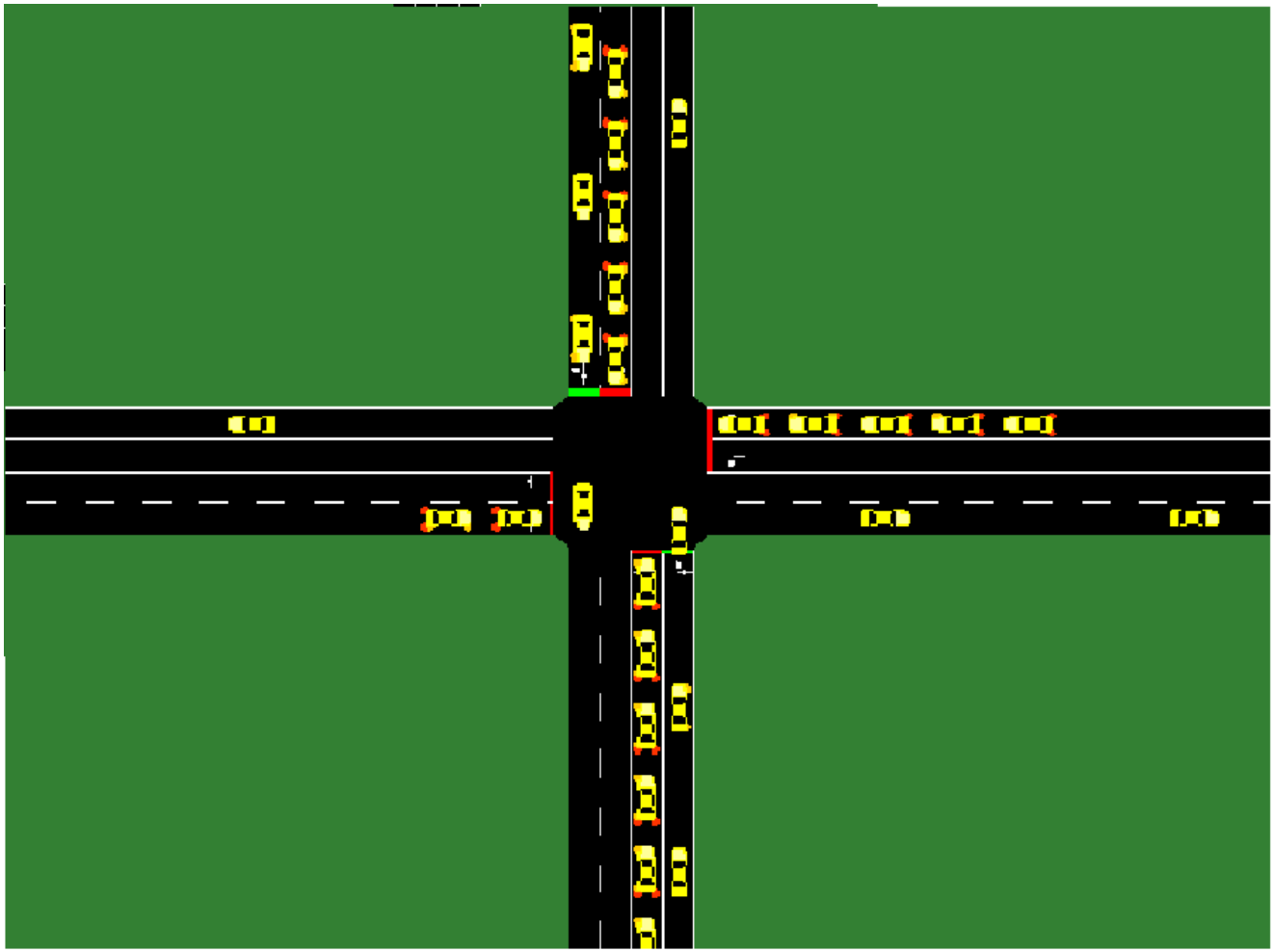


Figure: two way intersection

Summary of the model

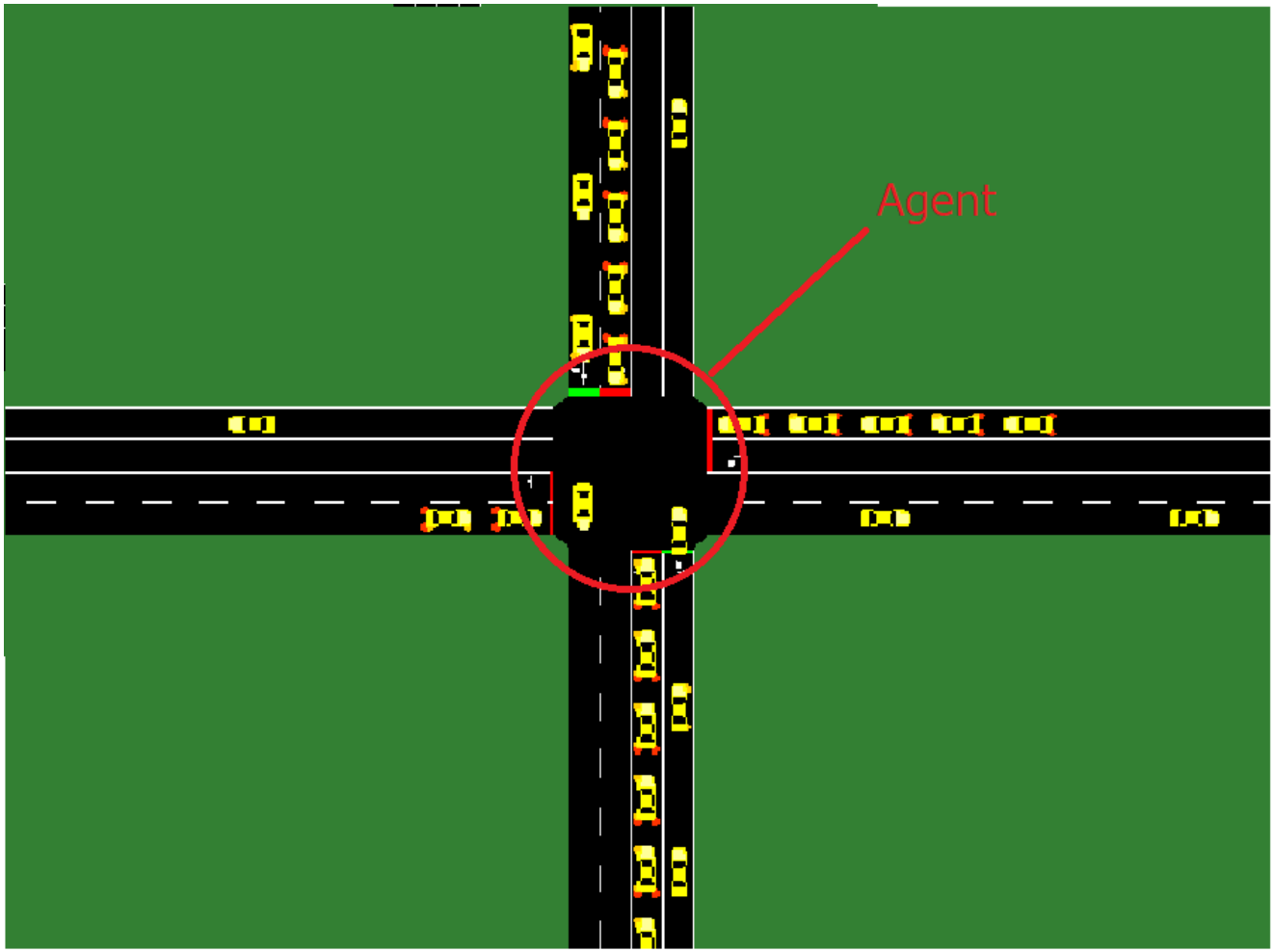


Figure: Agent

Summary of the model

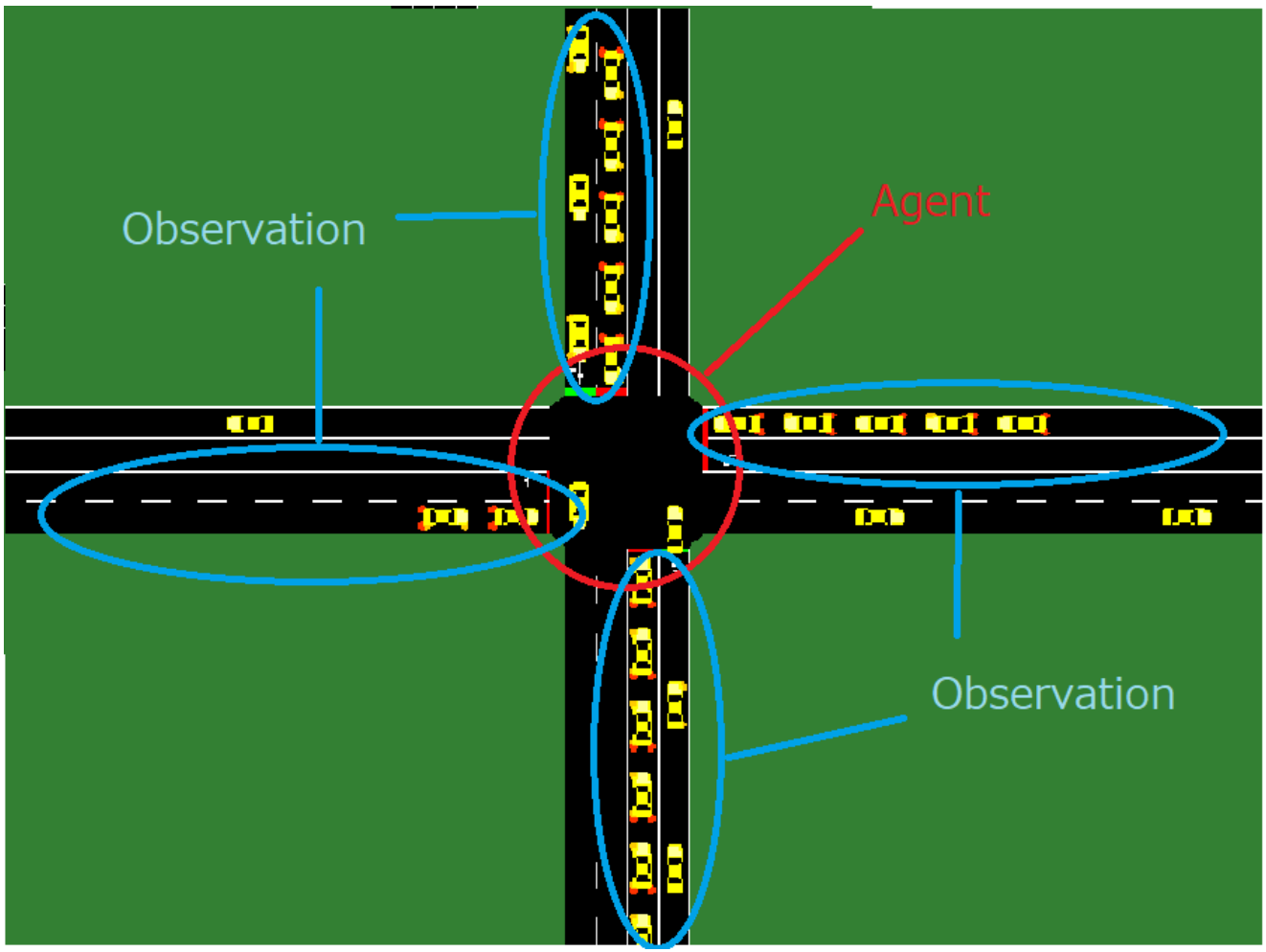


Figure: Observation

Observation

The agent's observation is following four vectors.

Variable	**Description**
phase_one_hot	A one-hot encoded vector indicating the current active green phase.
min_green	A binary variable indicating whether min_green seconds have already passed in the current phase.
lane_i_density	The number of vehicles in incoming lane i divided by the total capacity of the lane.
lane_i_queue	The number of queued (speed below 0.1 m/s) vehicles in incoming lane i divided by the total capacity of the lane.

Action

The action space is discrete[0, 1, 2, 3]. Every 5 seconds, the agent can choose the next green phase configuration.

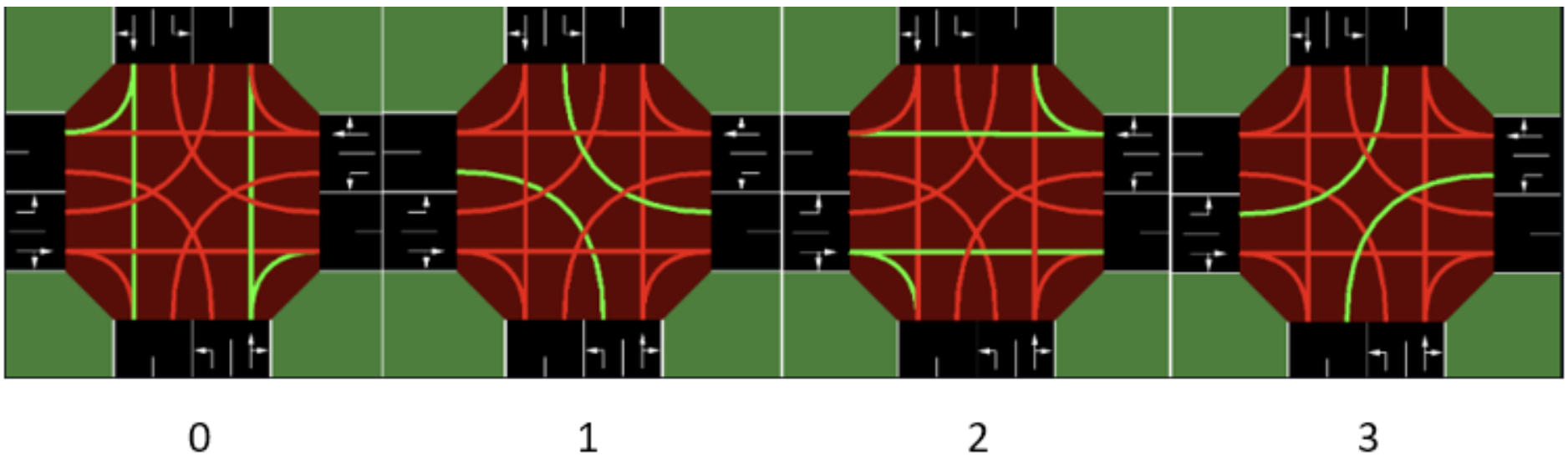


Figure: Action Space

Rewards

The reward function is the change in cumulative vehicle delay:

$$r_t = D_t - D_{t-1} \tag{1}$$

where D_t is the sum of the cumulative waiting time of all incoming vehicles (Other possible measures for the reward function are lanes density or average speed).

In this setting, the agent tries to minimize the cumulative vehicle delay.

Algorithm

- DQN

Deep Q-Learning

Deep Q-Learning is a reinforcement learning method that applies the Q-Learning algorithm to a Deep Neural Network (DNN) to handle continuous state and action spaces[2].

Q-value update equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max_a Q^{target}(s_{t+1}, a) - Q(s_t, a_t)\right)$$

In Deep Q-Learning, the Q-values are approximated by a DNN, and the network parameters are updated using the state, action, reward, and next state as learning data.

Replay Buffer, Target Network, and Gradient Clipping

We use stable-baseline3 for DQN, and it uses a replay buffer, a target network and gradient clipping for stabilize the learning process.

- **Replay Buffer:**
  - A data structure used to store experiences (state, action, reward, next state) during training.
  - Enables learning from past experiences by sampling random mini-batches.
  - Helps to break correlations between consecutive samples, improving learning stability.

Replay Buffer, Target Network, and Gradient Clipping

- **Target Network:**
  - A separate neural network with the same architecture as the main DQN.
  - Used to compute the target Q-values for updating the main DQN.
  - Parameters are periodically updated from the main DQN, providing a more stable learning target.

- **Gradient Clipping:**
  - A technique to limit the magnitude of gradients during backpropagation.
  - Prevents large gradients that can cause unstable updates and negatively impact learning.
  - Helps to stabilize training and improve convergence in Deep Q-Learning.

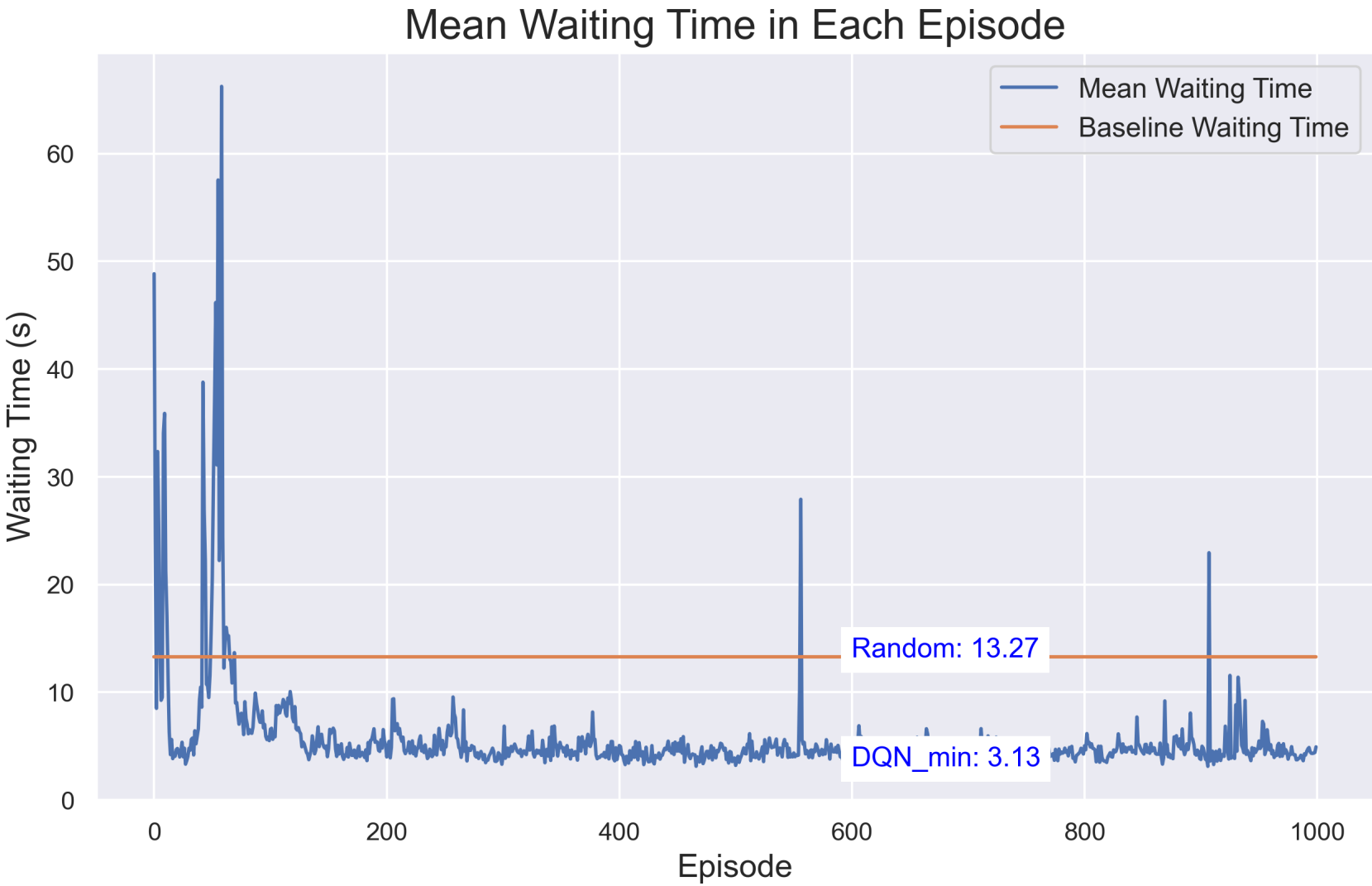
Hyperparameters

Parameter	Value
Episode Number	1000
Episode Length	500 seconds (100 steps)
Learning Rate	0.001
Learning Start	0
Target Update Interval	500
Initial Epsilon	0.05
Final Epsilon	0.01

Table: DQN Hyperparameters

-> Total learning step is 100,000 (1000 episodes * 100 steps)

Learning curve of DQN agent



Summary of Findings

- Baseline result comes from the agent which takes random action
- Performance of the DQN agent became worse once during the initial learning stage (episodes 20-30) -> **Exploratory action**
- Learning was mostly completed by episode 200
- Compared to the baseline, the best DQN agent achieved a waiting time reduction of **about 10 seconds**

Demo

- See the videos.
  - The first video is during the training.
  - The second video is after the training.

Conclusion

- We successfully reduced waiting times at intersections using reinforcement learning for traffic light control
- For practical implementation:
  - Methods for observing the area around traffic lights are necessary
  - Examples: cameras, GPS data

References

[1] Lucas N. Alegre. *SUMO-RL*. https://github.com/LucasAlegre/sumo-rl. 2019.

[2] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].