Traffic Control: COMP 4010 (Group 8)

Annie Zhang
Carleton University
101187995
anniezhang3@cmail.carleton.ca

Connor McDougall
Carleton University
101179300
connormcdougall@cmail.carleton.ca

Larina Aribi
Carleton University
101185799
larinaaribi@cmail.carleton.ca

Sean Le
Carleton University
101184583
seanle@cmail.carleton.ca

1 Introduction

1.1 Focus

This project will deal with controlling the traffic lights at a four-way intersection by determining the most optimal time intervals based on incoming traffic. As this is a problem that has been optimized from a large number of different angles, we hope to bring our own take on it by combining other approaches to hopefully create a result that is more practical and realistic.

1.2 Rationale

As a city expands and transportation networks improve, traffic and the burdens it creates only intensify. Increased urbanization leads to more people commuting to and from their jobs leading to longer commute times for everyone across the city. Left unaddressed this unmanaged traffic can cause an increased chance of accidents, a higher environmental impact and a decreased quality of life for commuters. Traffic control is a critical issue that has traditionally been tackled by urban planners to optimize a city's flow of vehicles and other modes of transportation. We believe that reinforcement learning has the potential to help reduce congestion, and by doing so decrease the average traffic waiting time for everyday workers.

In the context of reinforcement learning, traffic control is a very dynamic problem with numerous variables to account for. One can address the different road conditions per season while also considering the number of lanes as well as type of lanes. The agent might need to consider turning lanes where a turning light would need to be implemented. The quality of driver could also become a scenario that the agent would need to train for. How should lights behave when an accident occurs? In considering all these variables, the action space has the potential to become very large but also lends itself to interesting behavior for the agent.

2 Approaches

2.1 Previous work

Using a Deep Reinforcement Learning Agent for Traffic Signal Control In researching previous work on the issue, we observed different implementations for the lights. In Genders et al. (2016) design, the light configurations follow a similar pattern to our own implementation, counting both the north/south and east/west combination for actions. In their possible actions they do consider the possibility of an advanced left turn for each axis. The all red configuration as well as flashing

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

a yellow light on each axis are actions that are not chosen explicitly, but rather as a transition that is required to occur between actions. For the rewards, the authors used the change in cumulative vehicle delay between actions, meaning it can be adjusted either positively or negatively depending on the actions. By opting for this, the agent can potentially be punished for the action selection, reinforcing positive outcomes. The implementation uses Q-Learning algorithm with a decreasing epsilon-greedy exploration policy, while also using a deep convolutional neural network to develop the optimal action-selection policy to approximate the action-value function. In regards to the internal representation, they employed matrices in order to represent the different lanes, though they did consider the possibility of storing speed, turning signal and traffic light phase.

IntelliLight Wei et al.'s (2018) approach models their traffic simulation off of large scale traffic data taken from 1,704 surveillance cameras across Jinan, China over a period of a month. This differs from other research forays, which assume vehicles come at a constant rate. This approach is not true for properly simulating traffic patterns, as traffic is typically dependent on the time of day. IntelliLight's approach also emphasizes the understanding of the learned policies over the reward outcomes that drive other models. Part of the reasoning for this approach is the aim to align it closer to a realistic traffic scenario, as opposed to a solution that would mathematically lead to the same throughput but cause confusion among drivers, or strike as unrealistic for a real-life scenario. In this approach, because the basis of the learning is based on real traffic data, rewards alone cannot explicitly show if an agent has learned how to adapt based on different traffic patterns. IntelliLight also makes use of a phase-gated model, which considers the traffic light phase as a key-factor within the decision-making process. This makes use of subnetworks for the current active directions of traffic, while noting the traffic patterns and queue-lengths of the stopped directions.

Flow The approach of Wu et al. (2017) to traffic control issue models traffic as a finite-horizon Markov Decision Process using the SUMO traffic micro-simulator with rllab. Although we initially considered using SUMO to build out our environment, we decided to create the simulation ourselves as a greater learning opportunity. They employed neural network-based policies, namely Multilayer Perceptron (MLP) and Gated Recurrent Unit (GRU), which we hadn't learned about in class. Their learning algorithm was Trust Region Policy Optimization (TPRO), a policy gradient algorithm with baselines = 0.999, and step size 0.01.

Application of Deep Reinforcement Learning in Traffic Signal Control: An Overview and Impact of Open Traffic Data Greguric et al.highlight their findings of using Deep Reinforcement Learning (DRL) for traffic control over the traditional MDP approach. The paper cites the lack of adaptability that comes with the dynamic nature of traffic conditions. The greedy policies that come with traditional implementation may only focus on the immediate conditions rather than the long-term optimization of a transportation network that one would desire. In opting for DRL, the researchers developed an agent that can dynamically react to the situation, using deep neural networks for approximation over the traditional Q-table. Although we didn't employ DRL for our approach, it is interesting to see the possibilities that could be accomplished with a stronger method.

2.2 Our approach

State & Action Space In setting up the environment, we had to select our states and actions. The state space we chose is a list of size 12, containing only 0s and 1s. This maps to the first 3 spots of every oncoming lane, and the 0s and 1s indicate if road was empty or occupied. This meant our state space was 2^{12} by 12. Our action space was much simpler, being that the agent has two choices: switch the light, or stay.

Reward calculation For the implementation of our rewards structure, the team opted to calculate the rewards based on 3 buffers. The north/south buffer, the east/west buffer, and a free buffer. The directional buffers store the cars driving in their respective directions while the free buffer stores the cars that have passed through the intersection. Upon the light turning green, the reward calculation is defined by finding the total wait time in the north/south and east/west buffer. One of the totals will positively affect the reward, and one will negatively affect the reward, depending on the light's current orientation. The free buffer stores the list of cars that have passed through the intersection in only the past frame, and the reward is increased by the total wait time of those cars. The 3 buffer

calculations are scaled by floats for more in depth adjustment. If the light is yellow or red, the rewards are negative only to discourage switching constantly.

Listing 1: Reward Calculation

```
if cur_action == LightAction.V_GREEN:
    tot = (self.pos_scale * (ns_cars * ns_wait) - self.
        neg_scale*(ew_cars * ew_wait))
    tot += self.buffer_scale * self.buffer_reward(buffer)
elif cur_action == LightAction.H_GREEN:
    tot = (self.pos_scale * (ew_cars * ew_wait) - self.
        neg_scale*(ns_cars * ns_wait))
    tot += self.buffer_scale * self.buffer_reward(buffer)
elif cur_action == LightAction.H_YELLOW or cur_action ==
    LightAction.V_YELLOW:
    tot = 0.001 * self.neg_scale * (-(ew_cars * ew_wait) - (
        ns_wait * ns_cars))
else:
    tot = 0.01 * self.neg_scale * (-(ew_cars * ew_wait) - (
        ns_wait * ns_cars))
return tot, (ns_wait + ew_wait)/2
```

Q-Learning In researching the different approaches other papers have employed, our team opted to implement Q-Learning for our project due to it being model-free, lending itself to its adaptability to the dynamic nature of road traffic. Road traffic is difficult to model explicitly because many variables may arise, which may be infeasible to account for such as car accidents and road closures. In implementing Q-Learning, the agent is encouraged to explore different strategies by virtue of updating based on the maximum future reward. Q-Learning also works well in a scenario that requires discrete decision making which in the context of traffic control the agent would be deciding between which directions should have the right of way.

SARSA SARSA is a slight variation on the Q-Learning algorithm, with the primary difference being it is an on-policy algorithm. Using SARSA, the agent learns the state-action pairs based on the actions taken during exploration, as opposed to assuming the best possible action for future states. In abiding by this setup, SARSA's learning process is more conservative and stable which can be more suitable for a traffic control system. Q-Learning can in some cases select an action that may be optimal at the moment but does not necessarily reflect a stable traffic system that would be viable in the real world. SARSA's more conservative and cautious learning process lends itself to a more stable traffic flow. However, the downside to this is that convergence is slower, meaning that more episodes and iterations have to be executed to accurately determine effectiveness.

3 Empirical studies

3.1 Q-Learning

Table 1: Q-Learning experiment with 0.7 Step Size

Hyper Parameters		
Name	Description	Value
ϵ	Epsilon	0.01
step_size	Step Size	0.7
pos_scale	Positive Scale	1
neg_scale	Negative Scale	0.5
buffer_scale	Buffer Scaling	3

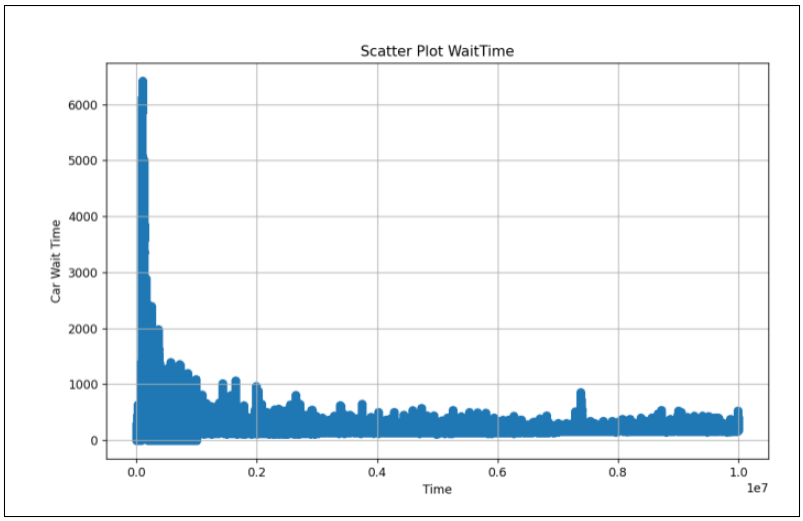

Figure 1: Q-Learning experiment Large Step Size.

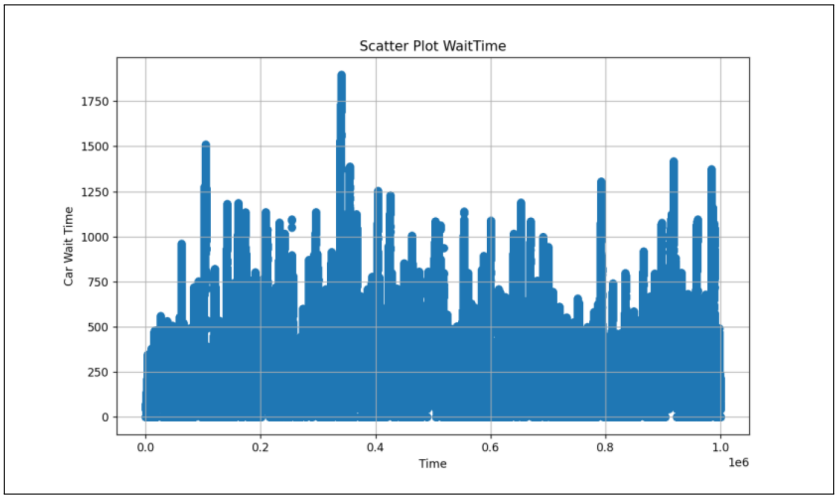

Figure 2: Q-Learning experiment Low Step Size.

Table 2: Q-Learning experiment with 0.01 Step Size

Hyper Parameters		
Name	Description	Value
step_size	Step Size	0.01
pos_scale	Positive Scale	1
neg_scale	Negative Scale	0.5
buffer_scale	Buffer Scaling	3

Based on the two figures, a larger step size worked better than a small step size in our case. The faster learning rate performed better. Perhaps if we had let the Agent run longer, it would have surpassed the larger step size.

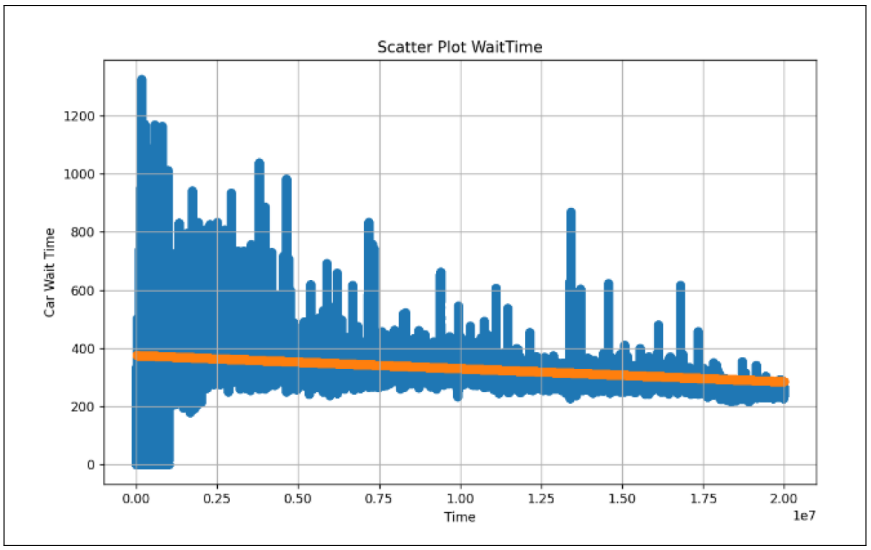

Figure 3: Q-Learning experiment High Positive Scaling.

Table 3: Q-Learning experiment with higher positive scaling

Hyper Parameters		
Name	Description	Value
pos_scale	Positive Scale	2
neg_scale	Negative Scale	1.5
buffer_scale	Buffer Scaling	3

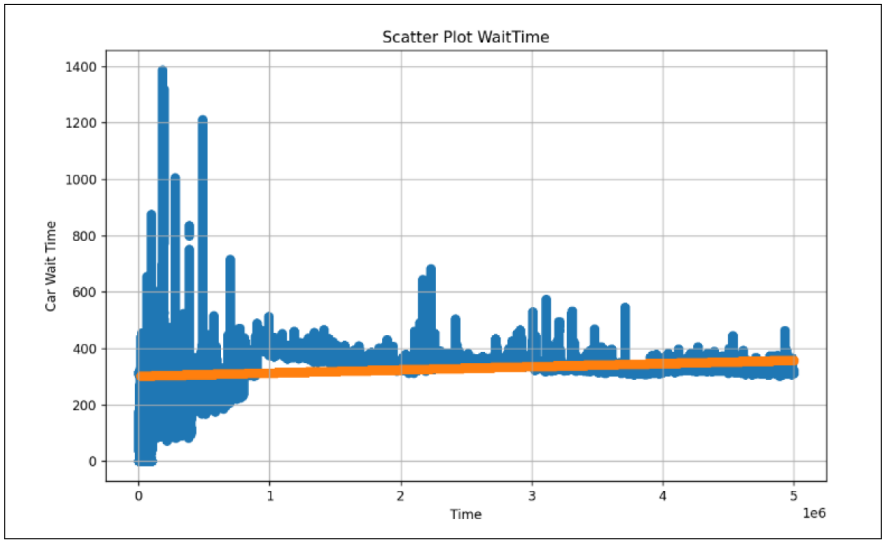

Figure 4: Q-Learning experiment High Negative Scaling.

Table 4: Q-Learning experiment with higher Negative scaling

Hyper Parameters		
Name	Description	Value
pos_scale	Positive Scale	1.5
neg_scale	Negative Scale	2
buffer_scale	Buffer Scaling	3

We found that a higher positive scaling was better than having positive scaling be equal to or less than negative scaling. Although altogether quite similar, the positive scaling had better relative convergence, if slightly more volatility on average.

Given our experiments, we found that the most impactful variables for QLearning were a high buffer reward and a high step size. Positive rewards should also be slightly larger than the negative rewards, but this factor was not overly impactful on results.

3.2 SARSA

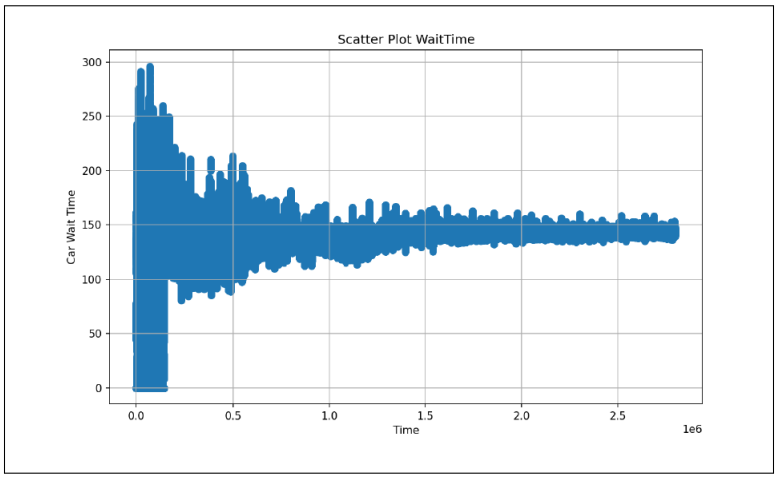

Figure 5: SARSA Base experiment

Table 5: SARSA Base experiment

Hyper Parameters		
Name	Description	Value
step_size	Step Size	0.1
ϵ	Epsilon	0.1
γ	Gamma	0.9

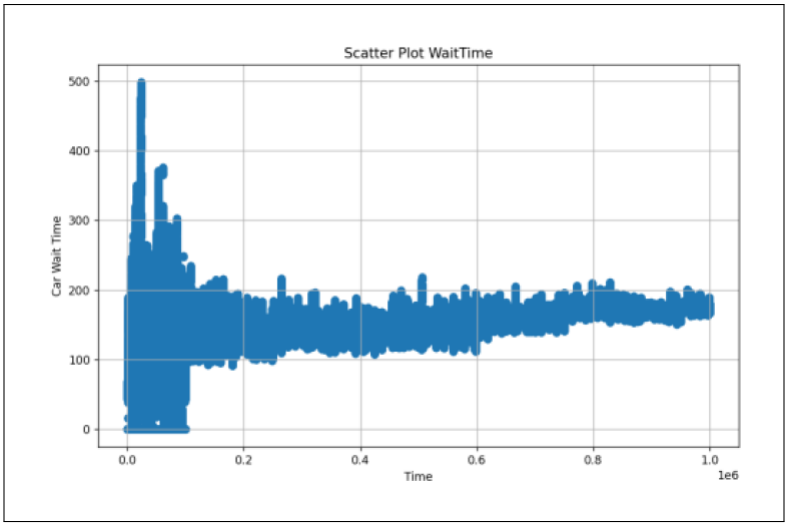

Figure 6: SARSA experiment Large Step Size.

Table 6: SARSA experiment with Large Step Size

Hyper Parameters		
Name	Description	Value
step_size	Step Size	0.9
ϵ	Epsilon	0.05
γ	Gamma	0.9

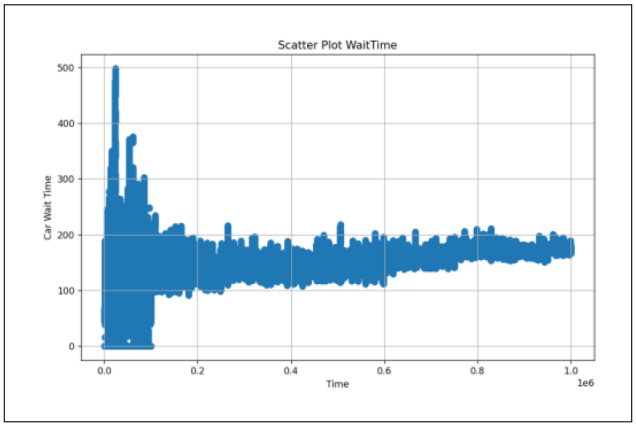

Figure 7: SARSA experiment Small Step Size.

Table 7: SARSA experiment with Small Step Size

Hyper Parameters		
Name	Description	Value
step_size	Step Size	0.05
ϵ	Epsilon	0.1
γ	Gamma	0.9

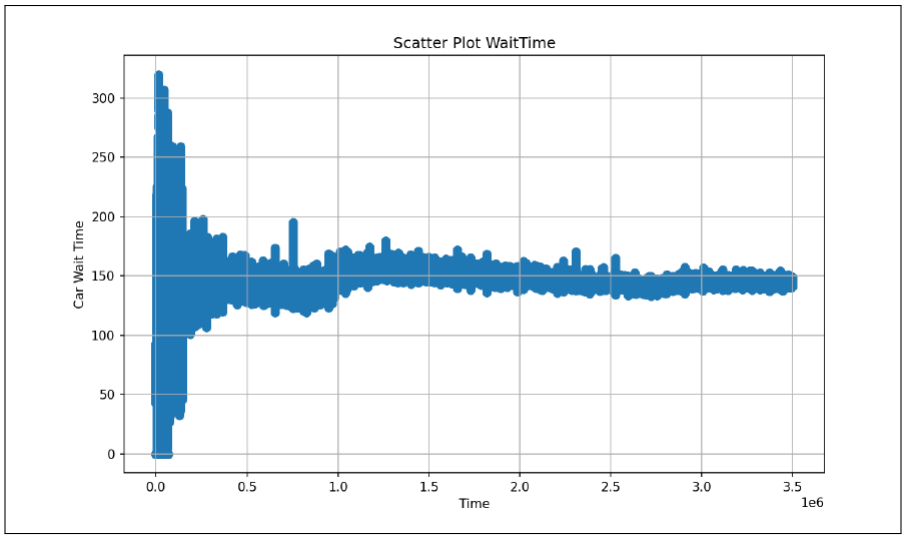

Figure 8: SARSA Small Epsilon.

Table 8: SARSA experiment with small epsilon value

Hyper Parameters		
Name	Description	Value
step_size	Step Size	0.1
ϵ	Epsilon	0.01
γ	Gamma	0.9

3.3 Findings

During every run with the SARSA algorithm we can get it to converge around 150 frames of wait time. As expected, with more episodes/iterations we get less variability, meaning if we run our model for longer we could expect to see it taper further. There is no obvious decline in the wait time after a certain amount of time, like we see within our QLearning results, meaning it quickly steadies around 150 and doesn't seem to improve much. High variability with small step sizes makes it infeasible, step size should be at least 0.1 for reasonable learning results, otherwise it'll take too long. Our SARSA results net a lower total wait time compared to our Q-Learning results, however results may differ if we run both with more computation power.

4 Conclusion

4.1 Lessons learnt

The traffic control project was our first time properly tackling an issue with RL in mind, which in turn became a whole learning experience. Traffic control from an initial glance felt like a simple problem to optimize, but presented lessons upon lessons on how one should utilize RL for the task. The process of setting up the environment introduced us to the basic problem of how one could optimize a traditional 4-way intersection with no advanced turning lane. Assuming each driver was going straight and there was no turning already presents a relatively sizable action and state space that, depending on how many cars the agent is capable of seeing, can grow the action space exponentially. When considering other factors that could have been added such as advanced turning lanes paired with advanced turning lights, the possibility of someone performing a U-Turn, or a car accident interrupting the flow of traffic, developing and testing against the most basic cases made us realize how deep the issue can go. In doing so we also realized that one can train an agent to optimally control the flow of traffic at an intersection, but at the same time it may come at the cost of what makes sense for the average driver. Without proper tuning, the agent begins optimizing to a fault,

resulting in light patterns that aren't realistic for physical application. The project taught us many things regarding its implementation, but also generated questions about the consequences of the optimizations that were generated.

4.2 Improvements for the future

If given more time, we would look into adding a more dynamic car spawning to the environment to simulate a proper day of traffic, as opposed to assuming a constant semi-fixed pattern for incoming cars. We would likely look into local traffic data to model our traffic generation, but that in itself would be a sizable undertaking outside of the current scope of the project. We also thought about adding advanced turning lanes and lights to simulate real-life traffic patterns. This would involve transferring between the different buffers and require a bigger change to the reward structure, as we would have to think whether the turning lane would become its own buffer or if it would remain within the traditional north/south and east/west buffer. We would also look into more dynamic driver behaviour as the quality of the driver can severely impact a build-up of a certain lane. In adding this dynamic behaviour for the drivers, the agent would be more prepared for outlier situations that tend to arise. Another implementation that would go hand-in-hand with this would be accounting for weather behaviour. Depending on the region, drivers often must contend with freezing rain or heavy amounts of snow, which in turn changes how drivers behave and could force drivers to come to an abrupt stop or perform certain maneuvers. Given the nature of the problem and how applicable it is to so many different environments across the world, it would be a massive undertaking to realistically account for how dynamic the roads can become.

Another possible area of improvement would be in the setup of the state space. Currently the state is made up of a list containing 0s and 1s pertaining to slot occupancy. This approach is perhaps too shallow a pool of information for adequate learning over time. Additionally, changing the state space could have allowed us a better reward calculation method. The reward calculation we have in place suffers a bit from what we currently store in the state space, and can lead to the agent reward hacking. Given more time it might have been useful to see if adding more stored detail such as the current phase of light to the state space might have improved results over the long term.

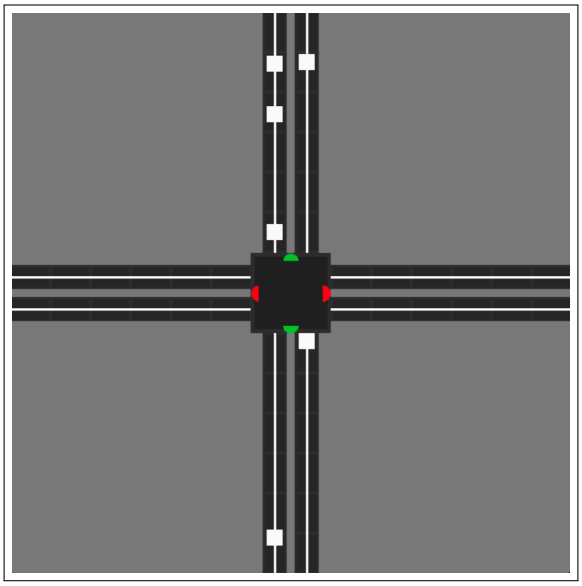

Figure 9: Traffic Control Visualizataion

References

[1] Genders, W., & Razavi, S. (n.d.). Using a Deep Reinforcement Learning Agent for Traffic Signal Control. https://arxiv.org/abs/1611.01142

[2] Gregurić, M., Vujić, M., Alexopoulos, C., & Miletić, M. (2020). Application of deep reinforcement learning in Traffic Signal Control: An overview and impact of open traffic data. Applied Sciences, 10(11), 4011. https://doi.org/10.3390/app10114011

[3] Wei, H., Zheng, G., Yao, H., & Li, Z. (2018). Intellilight. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 4, 2496–2505. https://doi.org/10.1145/3219819.3220096

[4] Wu, C., Kreidieh, A. R., Parvate, K., Vinitsky, E. A., & Bayen, A. M. (n.d.). Flow: Architecture and Benchmarking for Reinforcement Learning in Traffic Control. https://doi.org/10.48550/arXiv.1710.05465