**Intelligent traffic light controller with deep reinforcement learning.**

**Deep Learning project by Yaniv Hacker, March 27th 2022.**

**INTRO:**

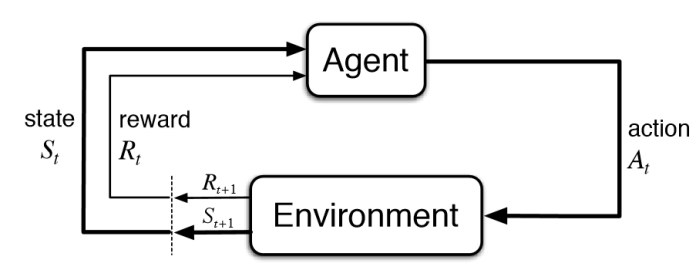
In a world where population keeps on growing, the traffic issue keeps getting worse.

More and more cars are being held in their way to work for hours each day, wasting time, money and a lot of mental health by waiting in this non-stopping “circle of life”.  
My project brings an improvement to this painful topic – I used Reinforcement Learning to study the behavior of each traffic light in intersections, to try and manage the lights to minimize waiting for cars crossing. I solved this problem not only for a single intersection, but for multiple intersections, where cars go from one intersection to another repeatedly.

Why do we even need this system? Isn’t the fixed strategy that we used mostly in real world sufficient enough to control the traffic all across the world? The answer to this question is NO.   
A fixed strategy gives a fixed amount of time such as 30 seconds to each side of the road, whereas the traffic flow might be different. For instance, in the mornings, the roads leading to Tel Aviv are far busier than the roads leaving Tel Aviv, while in the evenings, it’s the other way around.  
In conclusion, we can agree that finding a smarter, more efficient way to operate traffic lights will significantly reduce the amount of waiting time for cars crossing, and will save millions and even billions of people’s daily time, money and frustration.  
  
**Project setting:**  
To test my algorithms, I used the Simulation of Urban MObility (SUMO) simulator platform, on a linux-based (MacOS) operating system, where I made single agent simulations, and multi agent (2 and 9 agents) simulations, which we can see further in this document.  
The data of cars flow was generated in the SUMO simulator, and was transferred to python with the Traci library.  
No preprocessing was made on the data.

**Single agent case:**   
In this setting we first create a traffic simulation which only contains a single intersection. The traffic is coming from all sides of the road, and we named each side of the road as North, East, West and South. Each road has three options to cross the intersection – Going straight, turning left or right. For traffic generation we use Simulation of Urban Mobility (SUMO), as shown in the picture below.

**WHAT IS Reinforcement Learning?**



Reinforcement Learning is a branch of Machine Learning with a specific structure and flow, as described in the chart. An RL agent performs an action in an environment, which causes some kind of reaction/result in the system. The result that appeared yields a reward – whether the consequences were good or bad. These results are being transferred back to the agent, which adapts itself corresponding to the reward it got from his last action.

RL algorithms can be divided into two types: model-free RL algorithms and model-based RL algorithms. In model-free RL algorithms, we do not have an exact model of the environment, which means that we do not know what will happen in the next time step after we performs an action. On the other hand, in model-based RL algorithms, agent must have to learn the model of the environment which may not be available in most of the real-world problems.

In this work I used model-free RL algorithms to control the traffic lights. Model-free algorithms further can be classified as **value-based (Q-learning and QL with Neural Networks = DQN) and policy gradient (A2C, PPO)** algorithms.   
In this work we will work with both value based and policy gradient algorithms, and examine their success on our problem.

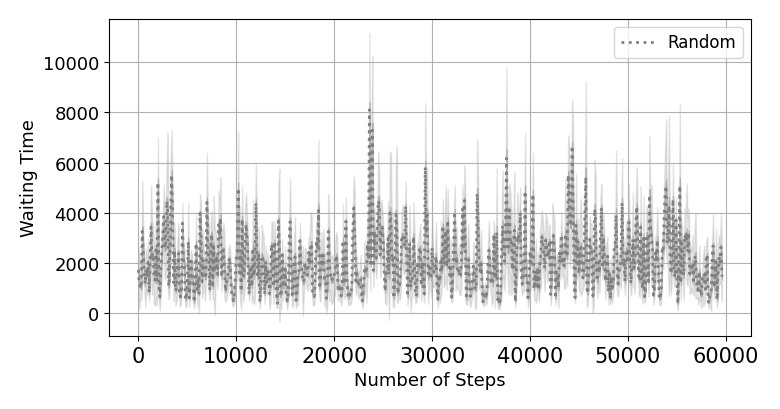
Since we didn’t learn these in class, I’ll elaborate:

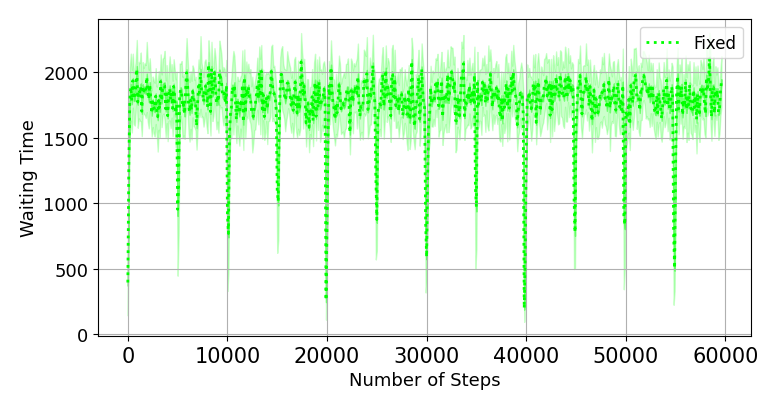
A2C – The Advantage Actor Critic algorithm is somewhat of a merge between the value-based and policy-gradient methods. Each iteration, it computes two values – the Critic function, which is the same function the DQN goes by, to minimize the error, and the Actor function, which takes the Critic’s result in consideration and updates the policy the agent goes by.

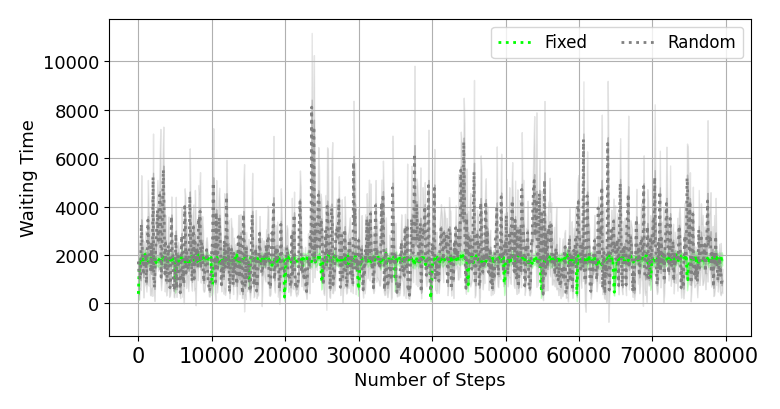
PPO – The Proximal Policy Optimization algorithm does what on-policy algorithms do, which is to continuously update the policy, so that the agents using the policy will be guided to making the optimal decisions for the entire system, to maximize our reward OVERALL (opposed to maximizing it in the short term, like DQN)

**Experimental results – Single Agent (Intersection) problem:**

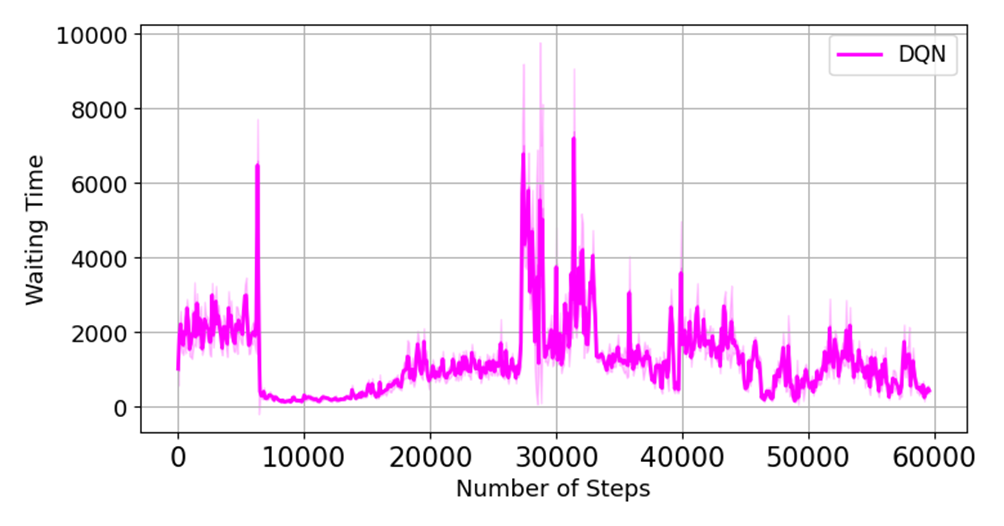
We first run Random agent, which makes random actions, which aren’t a result of his previous actions & rewards,and fixed agent, which gives each traffic light a fixed “green time”, and compare their performance.

This figure clearly shows that random agent does not learn anything (as expected), and the waiting time doesn’t improve overtime.



Fixed agent has much better performance than random agent in terms of avg. waiting time, but still – as it’s fixed, it doesn’t improve overtime, and doesn’t “care” about the real flow of the traffic in the intersection.

Now, we move on to trying to manage the lights to be responsive of the actual traffic in the intersection, using DQN:

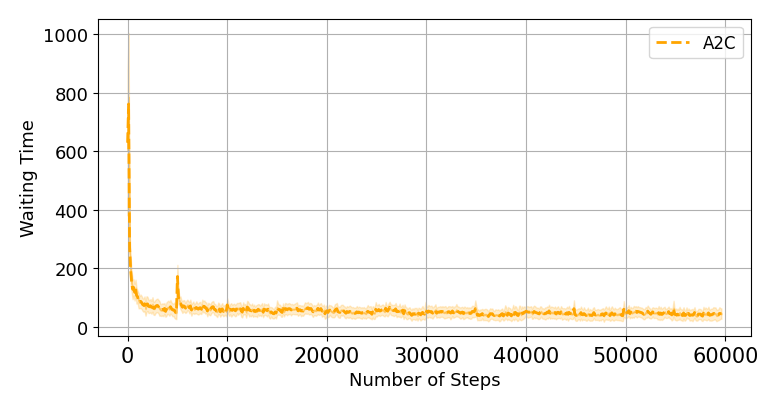
****

**DQN seems to converge and improve its results overtime, but the final waiting times aren’t outstanding.  
WHY?**

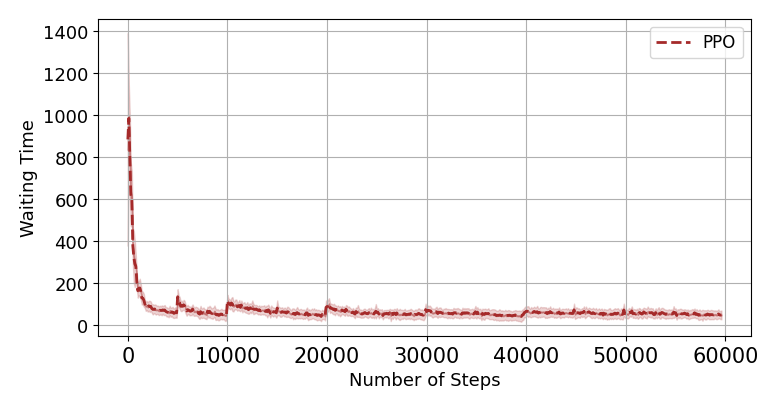
The DQN algorithm is an off-policy algorithm, which tries to maximize the reward it gets in each iteration separately.  
This approach gets us pretty good results, but not optimal.  
Optimal results can be achieved with algorithms that learn a POLICY to go by (like a manual for the agents), that way we achieve a MACRO view instead of a MICRO view in the DQN algorithm.

After realizing on-policy algorithms might perform better in this problem, let’s take a look on their result, in the next page.

Let’s check the A2C performance:



As we can see in the chart, A2C algorithm manages to successfully deal with the single intersection traffic light problem, and controls the lights in a way that minimizes almost completely the cars’ waiting time.

Let’s also check PPO performance:

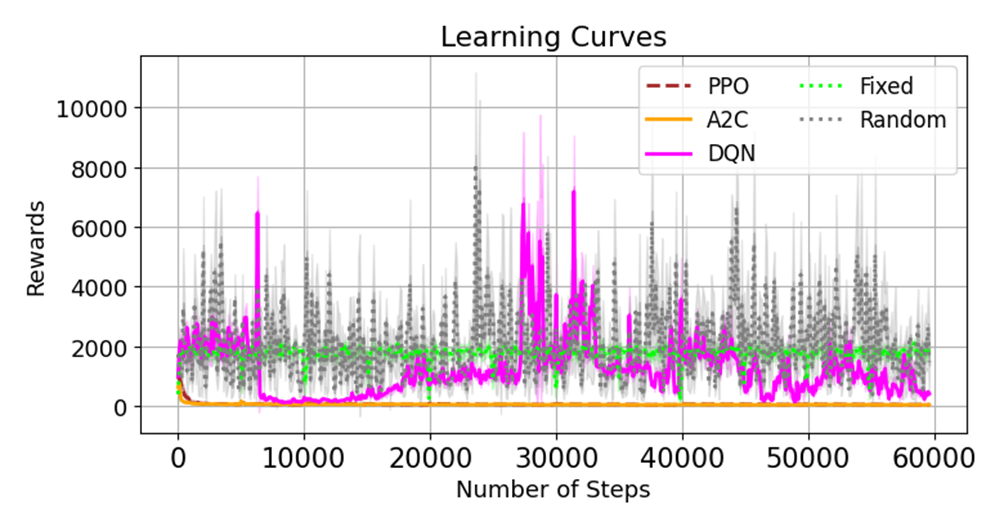
As they are relatively similar, PPO algorithm also performs well in the single agent problem.

**Single agent conclusion:**  
Random Agent – Waiting times are high, and don’t improve overtime.

Fixed Agent – Waiting times are medium, and don’t improve overtime.

DQN – Converges successfully overtime and is better than fixed, but still performs worse than on-policy algorithms.

A2C, PPO – Waiting time decreases massively and end up minimized – solves and deals well with one intersection.



**Lets Move On To Multi Agent - The Real World!**

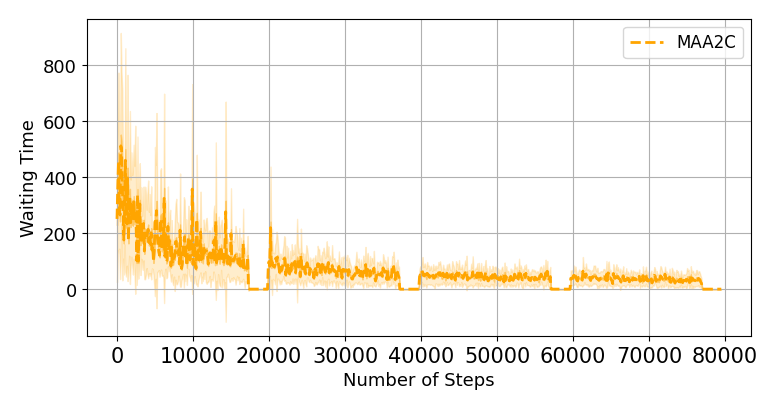
**Multi-Agent:**

After finding a good solution and minimizing waiting time in the Single Agent problem (just one intersection), we move on to the multi-agent problem – The real world. In real life, there are a lot of intersections, where one road you take in an intersection leads you to another in the next one.  
We’ll attempt (Spoiler - and succeed 😊) to deal with this problem as well.

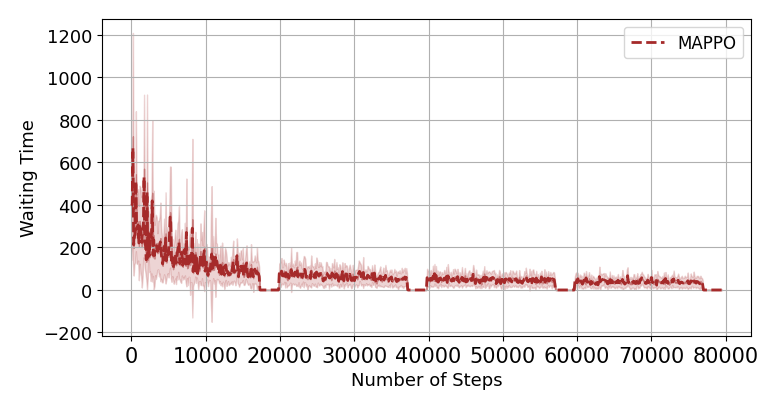


In the picture above, we can see two intersections (therefore we have two agents), where each agent is controlling its own intersection.

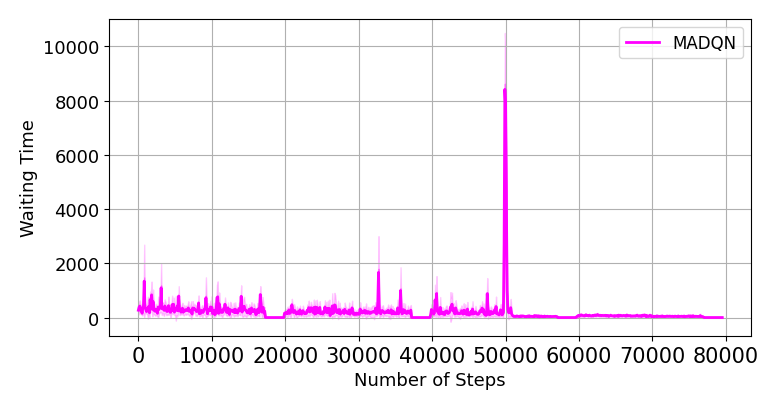
Here are the results:

A2C:

PPO:



DQN:



**What can we see?**

DQN again successfully converges after a decent amount of learning, but is again worse than A2C and PPO (look at the y axis’ scale).

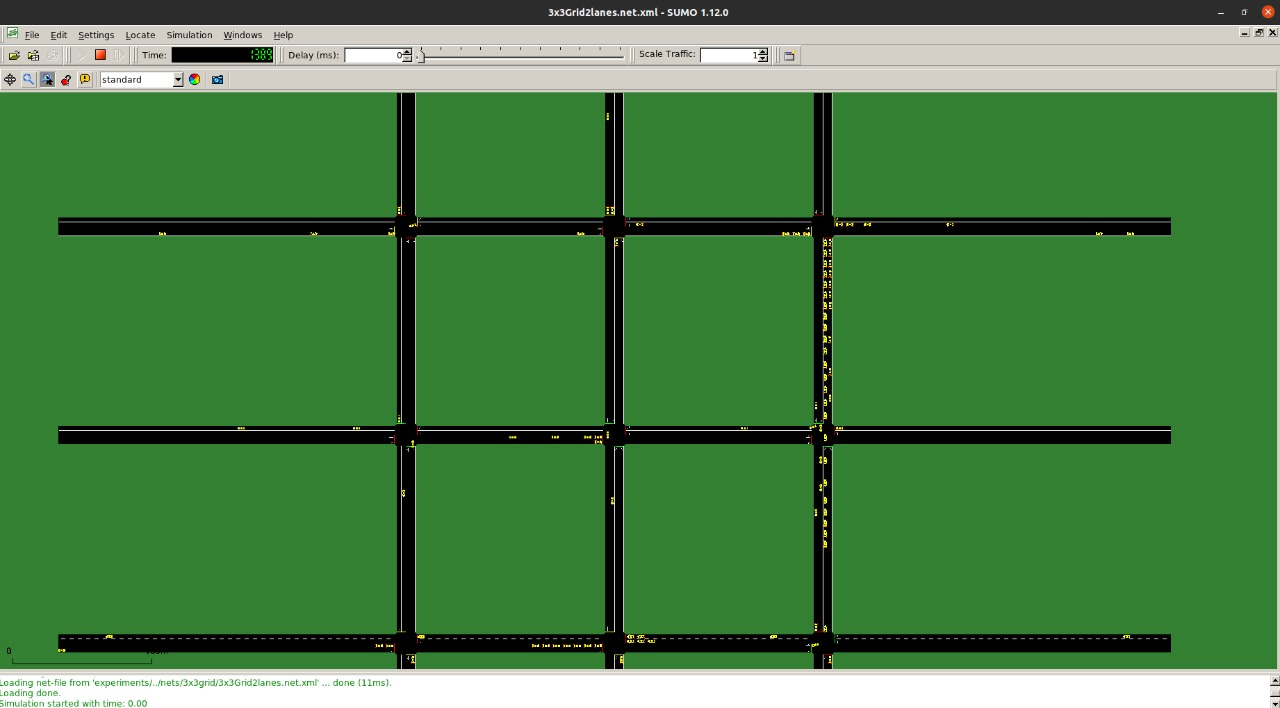
As for A2C and PPO, not only we succeed at solving this problem, **the waiting time for all cars in all intersections COMBINED is the same as it was in the single agent problem.**

**How is this possible?**

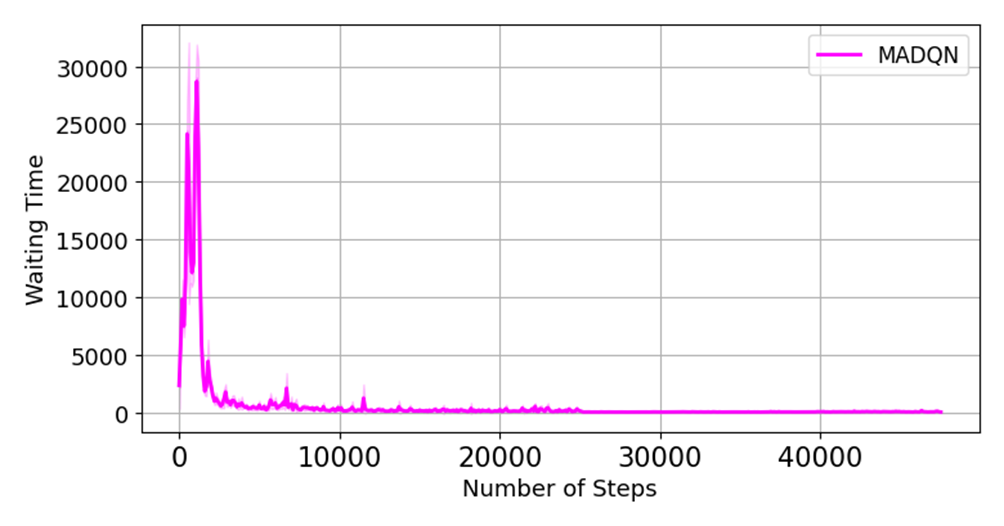
The cause of this is the fact that in single agent, the agent has no idea where the traffic will come from.  
In Multi Agent we have a big advantage – the agents communicate, and tell each other – “Cars are coming from your \_\_ side, be aware”.  
This way the Multi Agent problem handles far more cars and intersections, but can maintain similarly low total waiting times.

**After seeing the great result with 1 & 2 intersections, we can look at the real world, where there’s a net of intersections, all connected to each other, where cars flow in every direction.**

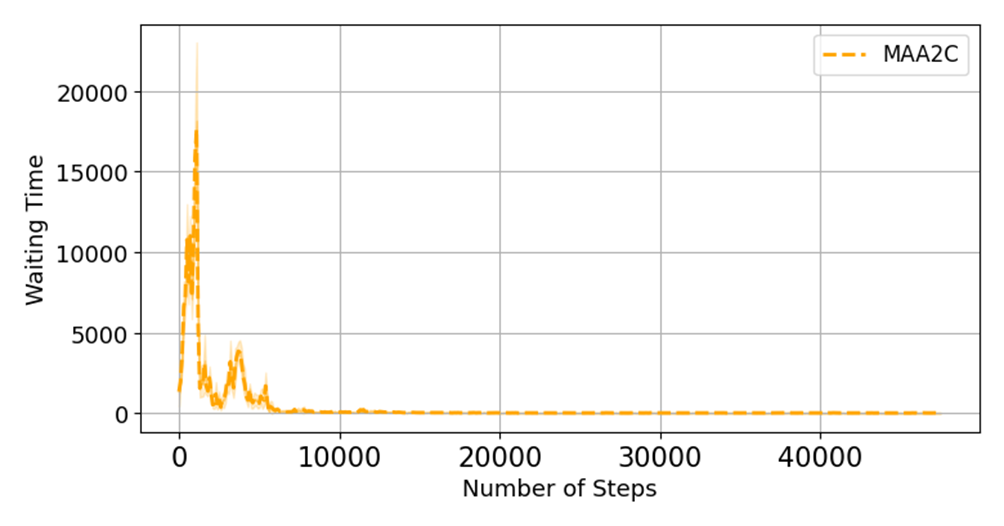
**Third experiments:**

****

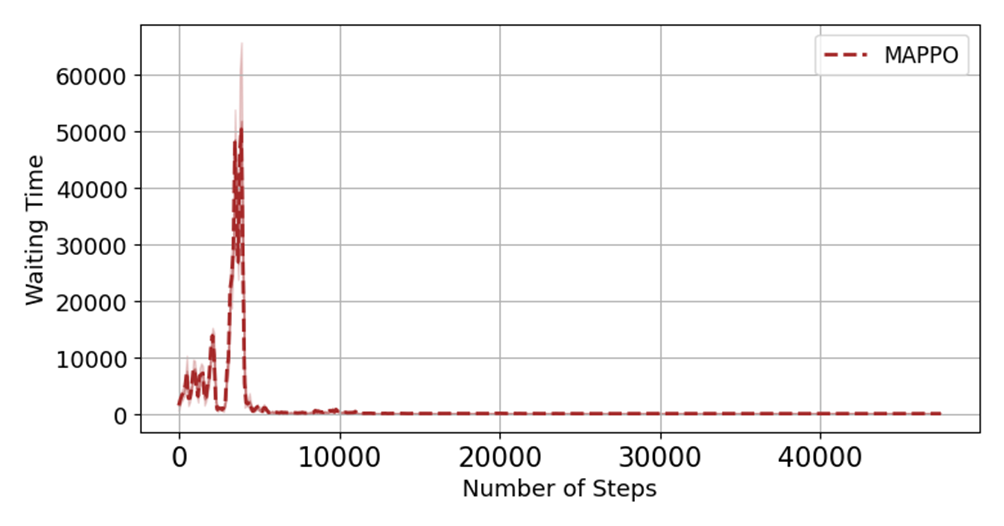
**Results:**

**DQN:**

A2C:

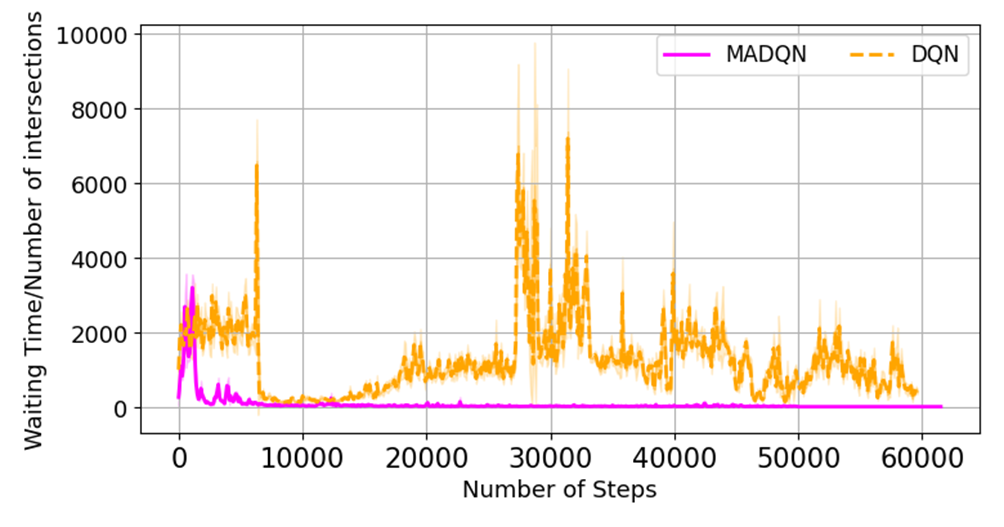


**PPO:**

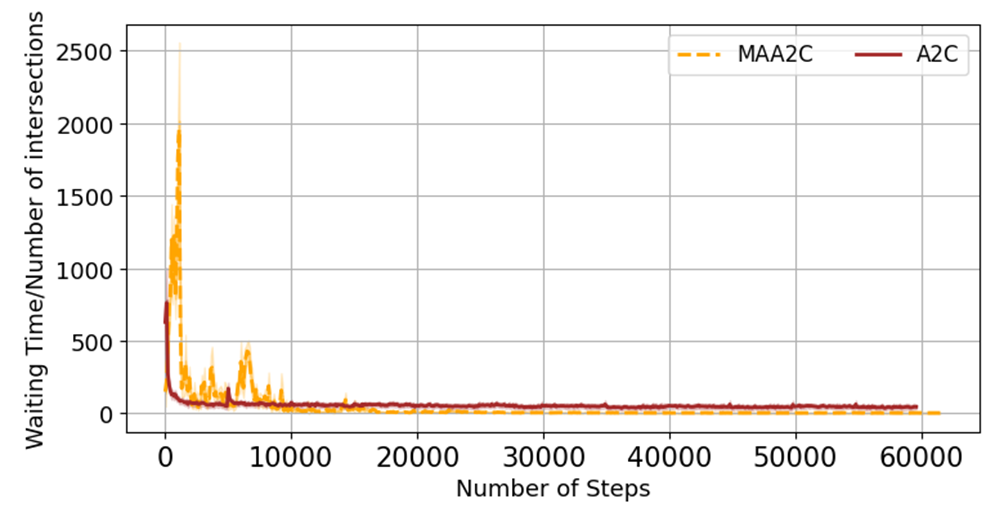
****

**Now, after doing the 9 intersection experiment, we can show the performance of the Single Agent approach VS Multi Agent approach.  
The plots below describe the total waiting time per intersection, as a function of time:**

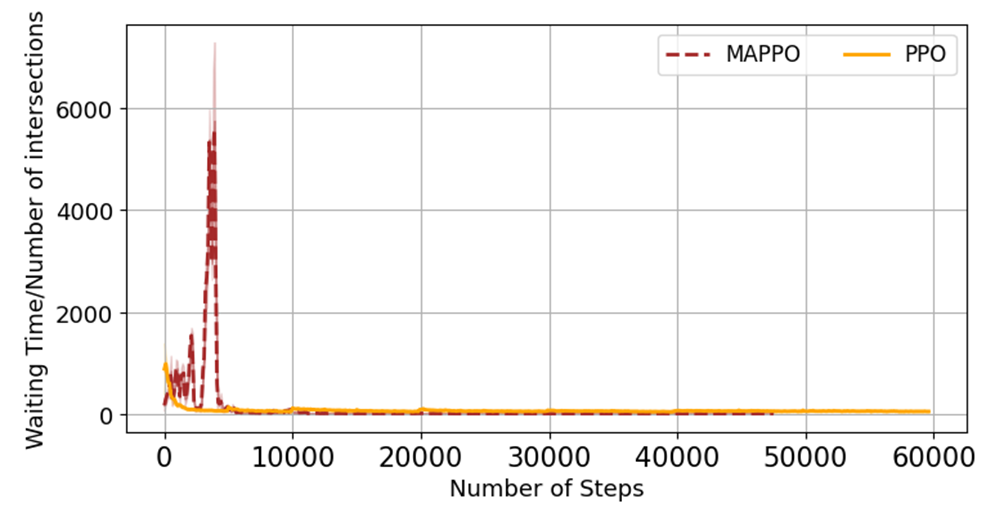
**DQN:**

****

**A2C:**

****

**PPO:**

****

**We can see that in each of the algorithms, the waiting time per intersection in the Multi Agent approach is lower, and if we would scale the plots to a higher resolution we would see a significant difference.**

**Model Comparison Conclusion:**

**We’ve seen that PPO and A2C work better than DQN on this case of problem.  
Why?**

A – A2C tends to succeed in scenarios where information needs to be transferred between agents in an environment.

B – This problem fits the on-policy model better, since it’s very behavior-based, and doesn’t converge well when actions are always taken in a greedy manner (like DQN).

C – Having a policy that is learned and can guide the agents what to do next helps us here, since it provides a lot of information to the agents, who can now make their decisions based on the entire environment instead of just making sure they do the best thing for their own “small world” in the specific moment in time. Also, a policy to guide the agents provide us with a macro view that is capable of capturing the whole scene, instead of each agent optimizing itself just by the next move.

Conclusion – Having a policy that learns from a macro view overtime works far better in this problem than having agents who each optimize their own benefit in every single INDEPENDENT action.

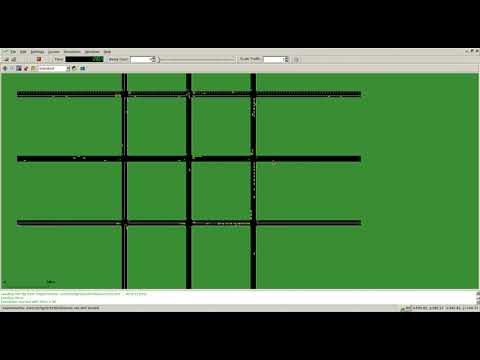
**Realistic implementation of this project’s system:**  
As I believe the idea and initiative is important and technically possible, and seeing the great results I got, I wanted to add a few points to make this solution real-world applicable:  
  
1. The traffic reports will be reported by traffic cameras (which nowadays are pretty much everywhere), and with a Waze integration.

2. The system will be connected to traffic lights, will study the behavior of the intersections they operate in and will manage the lights to minimize cars waiting.

3. As this is a very important topic in our small country, the Ralbad (Rashut Lebetihut Badrahim) will be happy to assist with providing information, funds and camera findings in case they like the solution.

4. Note: in order for this to be a real-world solution, cybersecurity actions will need to take place and be integrated with this Machine Learning system, since in today’s world, having someone breach this system can be very harmful.

Attachment:

[](https://www.youtube.com/embed/ZbA7sYQ-lxY?feature=oembed)A video of the SUMO simulator, after optimizing the traffic using my Reinforcement Learning algorithms.  
\* If it doesn’t work, copy the link and watch in YouTube 😊

GitHub link to project: <https://github.com/YanivHacker/RLTrafficManager>