

**CS609 Introduction to Reinforcement Learning**

**Group Project Final Report**

Topic: F1 Race Pit Stop Decision Optimization

**Group 8**​

Chia Dehan

Colin Jiang Kelin

Lim Wei Jie

Ng Juan Yong

November 2023

[Executive Summary 3](#_Toc150713454)

[1. Introduction 4](#_Toc150713455)

[1.1. The Environment 4](#_Toc150713456)

[1.1.1. The State 4](#_Toc150713457)

[1.1.2. Actions 4](#_Toc150713458)

[1.1.3. Transitions 4](#_Toc150713459)

[1.1.4. Rewards 4](#_Toc150713460)

[1.2. Approaches 4](#_Toc150713461)

[2. Algorithms 5](#_Toc150713462)

[2.1. MCTS 5](#_Toc150713463)

[2.1.1. Methodology 5](#_Toc150713464)

[2.1.2. Pseudo Code 7](#_Toc150713465)

[2.2. Other models 8](#_Toc150713466)

[2.2.1. Rule-Based Agent 8](#_Toc150713467)

[2.2.2. Vanilla DQN 8](#_Toc150713468)

[2.2.3. TD(Lambda) Agent 8](#_Toc150713469)

[2.2.4. QRDQN, TRPO, ARS, RecurrentPPO 8](#_Toc150713470)

[3. Results and Evaluation 9](#_Toc150713471)

[3.1. Evaluation method 9](#_Toc150713472)

[3.2. Results and Final Agent Selection for Task 1 10](#_Toc150713473)

[3.3. Model behaviours 11](#_Toc150713474)

[4. Task 2 12](#_Toc150713475)

[4.1. Modifications 12](#_Toc150713476)

[4.1.1. Modifications to Environment 12](#_Toc150713477)

[4.1.2. Modifications to Action Space 12](#_Toc150713478)

[4.1.3. Modifications to Rewards 12](#_Toc150713479)

[4.2. Models 12](#_Toc150713480)

[4.2.1. Split Models 12](#_Toc150713481)

[4.2.2. Stack Models 13](#_Toc150713482)

[4.2.3. DQN Model 13](#_Toc150713483)

[4.3. Trainings and Results 13](#_Toc150713484)

[4.3.1. Training 13](#_Toc150713485)

[4.3.2. Results and Discussions 13](#_Toc150713486)

[References 19](#_Toc150713487)

[Appendix A: Rollout Policies and their Details 19](#_Toc150713488)

[Appendix B: TD(Lambda) HPT results 19](#_Toc150713489)

# Executive Summary

This assignment describes an environment simulating an F1 race with weather changes, track radius, and tyre conditions. Our objective is to use Reinforcement Learning methods to determine the optimal pit stop and tyre change decisions for our Racer Agent.

We started off with the benchmark case using Rules- based approach, where thresholds are used to determine when a pitstop should be made, and what tyres to swap to. Next, we tried classic RL algorithms, namely TD-lambda and Deep Q Networks. While TD-Lambda updates value estimates based on temporal differences, DQN uses neural networks as a function approximator to estimate the action values. We also tried using gymnasium environment wrappers to employ various stable baseline 3 algorithms such as QRDQN, TRPO, ARS, and Recurrent PPO. Finally, to gain an edge beyond learnings from this course, we adapted a novel method found from the paper “Online Planning for F1 Race Strategy Identification” by

Piccinotti et al. called MCTS, Rigorously Flexible Q-Learning Open Loop Planning (RF-QL-OLP).

The different models are evaluated against a set of fixed radius and weather configurations. From there, we found the top 3 models to be Rules- based, MCTS, and DQN. Rules- based approach came out top in most of the evaluation set, and produces decent results in sets where it did not rank first in. MCTS did well in sets where DQN and Rules- based struggled in, but experiences sharp drops in performance in many others. DQN was very similar to Rules- based but was always lacking slightly in rewards.

In task 2, we attempted to make modifications to the environment to make it more representative of a real race. Here, we expanded the action choices of the agents to allow the racers to drive “recklessly” (increased velocity). In exchange, this would increase their rate of accidents and risk getting disqualified. To incentivize the agents to take this risk, we included multiple agents in each episode and changed the rewards to be based on race rankings rather than on time alone. Intuitively, racers would take risk in order to rank higher and reap exponentially better rewards.

The different models are then evaluated against the same opponents. From the evaluations, we gained insight into how models make decisions on driving strategies and discovered that adopting a probabilistic approach gives the best performance. Combining our findings, we propose a rule-based probabilistic approach which outperforms all other models.

# 1. Introduction

Racing is a competitive sport where the outcome of a race depends on both intrinsic and extrinsic factors. Intrinsic factors include condition of the car, the driver’s skill or aggressiveness, and the team's tyre and pit-stop strategies. Extrinsic factors, on the other hand, include weather, probability of weather changes, track curvature and tyre wear. The project group is looking into using Reinforcement Learning algorithms to decide when to pit-stop and what tyre compound to use.

## 1.1. The Environment

The environment controls the weather transition for the state. It also, given the weather, determines the deterioration of the car tyres and velocity for that transition.

### 1.1.1. The State

The state is this environment is defined differently for different algorithms. It comprises of:

* Tyre conditions [float]
* Weather conditions [categorical]
* Track Radius [float or categorical]
* Laps Cleared [float or categorical]

### 1.1.2. Actions

The actions represent the pit stop decision and what type of tyre to swap to.

1. Ultra Soft Tyres
2. Soft Tyres
3. Intermediate Tyres
4. Full Wet Tyres
5. No Pit Stop

### 1.1.3. Transitions

Each transition represents 1/8 lap clear for the car. In other words, in each lap, there are 8 transitions. In each transition, the weather can change according to a set probability. The tyre deterioration, velocity, and rewards are hence determined by the weather. Pit stop decisions can only be made at every 6/8 lap, at which point an action is locked in. The tyre change (or pit stop) will only happen in the subsequent 8/8 lap. Each pit stop will incur a time of 23 seconds, and tyre conditions will be completely restored.

### 1.1.4. Rewards

Rewards are based on time taken to clear all 162 laps. The less time taken, the better the result.

## 1.2. Approaches

We tried many algorithms, which we will go into details in the following sections. In particular, we will spend more time (and pages) exploring the MCTS algorithm which is a new concept that we will introduce.

* Rules-based
* TD Lambda
* DQN
* QRDQN
* TRPO
* ARS
* Recurrent PPO
* MCTS, Rigorously Flexible Q-Learning Open Loop Planning (RF-QL-OLP) **(MCTS for short)**

# 2. Algorithms

This section briefly touches on the various algorithms that we have tried. However, more time will be spent on MCTS which is a new concept that we will introduce.

## 2.1. MCTS

In this section, we explore the use of ***Monte Carlo Tree Search algorithms*** to aid in reducing race time and improve a generalized pitstop and tyre strategy. Below, the project group has implemented a modified version of ***Q-Learning Open Loop Planning,*** a method recently proposed by Piccinotti et al. in 2021 that combines Monte Carlo sampling and Temporal Difference updates to figure out whether to pit-stop on each lap and which tyre compound to use.

Piccinotti et al. originally tested their method using a simulator based on (Heilmeier et al. 2020a), which they modified to make it work with planning algorithms. However, as their implementation delved into sampling from specific F1 tracks and the strategies used by real-life F1 teams and recorded by ESPN, a motorsport opinion website, we, instead, attempt to build new strategies by sampling from simulated trajectories with guidance from a rigorous, yet flexibly loose, set of rule-based rollout-policy.

Our modified version of the algorithm, which we name ***Rigorously Flexible*** ***Q-Learning Open Loop Planning (RF-QL-OLP)***, is shown to be able to ***reduce the race time*** over all stipulated conditions in task 1 of the project requirements ***by a range between 600 to 1100 seconds*** from the single action baseline.

### 2.1.1. Methodology

The diagram below shows how we employ crucial parts of the algorithm proposed by Piccinotti et al. and modify certain parts which do not aid in our project.

A diagram of a root

Description automatically generated

Diagram 1: Step-by-Step depiction of our group’s ***Rigorously Flexible*** ***Q-Learning Open Loop Planning***

***(RF-QL-OLP) algorithm***

In the third step, we discuss the rollout policy utilized. The rollout policy is an important mechanism in filtering important trajectories from which we will calculate the rewards and the subsequent Q-values. The following paragraphs aim to justify the rollout policy.

A circle with text on it

Description automatically generatedA math equations with black text

Description automatically generated with medium confidence

Diagram 2: Setting Threshold to Determine when to Pit with Lap Time Proxy

First, we have to set a threshold to indicate when it is time to pit for a car. To do this, we measure the expected time a car takes to complete a lap around a certain radius by taking the snapshot velocity at ¾ of each lap. Our findings from empirical testing shows that if we set the threshold too slow, pitstops occur too fast and too frequently, especially for tracks with smaller radiuses. On the other hand, if we set the threshold too high, pitstops for tracks with larger radiuses occur much later, which causes slower race times for larger radiuses. After tuning, we find that a lap time proxy of more than 1.5 times the pitstop time is optimal.

A screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generated

A white background with text and black text

Description automatically generated

Diagram 3: Setting Epsilon Threshold to Determine which tyres to change to

Next, in our rollout policy, we insert a secondary epsilon filtering threshold with some flexibility to account for tyre-to-weather durability and track radius factors. We first set the initial epsilon threshold to 0.5, which indicates random choice of tyres if threshold is passed. Then, we go through the following steps above in diagram 3.

In step 1, we intuitively and empirically test which tyres that are favourable to which weather. Whether a tyre is favourable to the weather is determined solely by its durability in the specific weather condition. For instance, we expect Fullwet tyres to be the most durable in 100% Wet weather conditions, thus this reduces the need to change tyres more frequently in wet conditions. The dictionary fixed in step 1 will then be used in step 2 to determine whether the epsilon threshold should be increased or decreased. An unfavourable tyre-to-weather variable shows that we have to change our tyres immediately while a favourable tyre-to-weather will prompt our rollout policy to not change the tyres. We also add in linearly adjusted factor to our epsilon threshold in step 2 to account for track radius length. The idea behind this is that a longer track radius will require a higher probability of changing tyre in the current ¾ mark of the lap compared to a shorter track radius. This is because for longer tracks, the next decision making point may be too long later which negatively affects our total race time. In step 3, we apply the flexibly transformed epsilon threshold to determine whether to do nothing or to do tyre selection.

In order to decide which tyre to select, we focus on the durability factor of the tyre in all race conditions, which can be measured by number of laps till which the tyres reach close to 0 for a specific constant weather and secondly, the linear rate of deceleration over time. Logically, a less durable tyre in 100% wet conditions will last much shorter and its linear rate of deceleration will be much higher. Thus, one can view the number of laps till which the tyres reach close to 0 as a positive factor while the rate of deceleration as a negative balancing factor. When we weigh both factors equally, they negate each other out and we are able to get the best tyre for each specific weather condition, radius and tyre type by sorting it in descending order.

The final tyres selected are matched with this list of tyres by durability. Thus, we aim to fit on the most durable tyres when the indicative conditions are right.

### 2.1.2. Pseudo Code

A white text with black text

Description automatically generatedA white text with black text

Description automatically generated A math equations and formulas

Description automatically generated with medium confidence

Algorithm 2: Rigorously Flexible Q-Learning Open Loop Planning (RF-QLOLP)

At each lap, we create a root node. We then expand the root node to its unexplored actions and subsequent states. Thereafter, we do a rollout to terminal state based on the previously discussed rollout policy. The backup function computes the Q value which represents the expected returns based on the rollout strategy.

We then take the expected Q-value after backpropagating and add the current rewards to the Q-value so to have a better representation of both long-term and short-term rewards. We then take the maximum of these 2 types of returns and choose the action based on it. The way we weight current rewards with the backpropagated Q-values is as follows and we do so as empirical testing shows that this is the most optimal weightage:



We take the maximum of long-term Q-value and the short-term immediate reward of taking the next action. The weightage again takes into consideration of track radius. A smaller track radius would want fewer pitstops taken, thus we weigh Q-values, a longer-term view, more than immediate rewards.

A number and numbers on a white background

Description automatically generated

We do a further clipping of the weightage of future q-values. We find that as track radius get larger, the decay in the importance of future q-values is exponential but smooths eventually to an asymptote. Thus, we try to replicate the exponential decay to an asymptote with a clipping function as shown above.

At the end, we take the action with the largest combined return function.

## 2.2. Other models

### 2.2.1. Rule-Based Agent

In the described scenario, a rule-based agent navigates through decisions based on the conditions of a race, specifically the condition of the tires, the weather, and the number of laps cleared.

There are two if statement checks. Firstly, the agent evaluates the condition of the tires. If the tire condition is below a certain threshold, ‘A’, a decision is made to refresh the current tyre. Secondly, the agent assesses the weather conditions. If the weather is dry, the agent opts for ultrasoft tyres. However, if it's wet, the agent's decision varies based on the degree of wetness. If the wetness percentage is below threshold 'B', soft tyres are chosen. If the wetness percentage is below a different threshold 'C' (presumably higher than 'B'), choice intermediate tyres is made. For wetness percentage beyond threshold 'C', fullwet tyres are selected.

Lastly, the agent checks the number of laps cleared. If the laps cleared exceed a certain threshold 'D', choice to not change tyre type is made.

These thresholds are optimized over grid search. Furthermore, to accommodate for the varying radius, a set of thresholds was determined for each radius bin (intervals of 50). In other words, for each radius, the agent will follow a different set of thresholds for its rules.

Through these set of rules, the agent can make informed decisions based on the racing conditions, ensuring appropriate actions are taken to possibly optimize the performance in the race.

### 2.2.2. Vanilla DQN

An agent is trained using vanilla DQN as a function approximator. The main network is trained only on states which laps cleared ends with .75, and the target network aims at states exactly 1 lap after.

Hyper-parameters are tuned using grid search, and the best parameters are chosen based on highest point of convergence of the average cumulative reward per radius.

The final agent is then trained for 2000 episodes. No more improvements were observed for training episodes after 2000.

### 2.2.3. TD(Lambda) Agent

6 agents are trained across lambda values of 0, 0.2, 0.4, 0.6, 0.8, 1. Each of the agents undergo hyperparameter tuning where an agent under a combination of parameters is trained for 5 episodes on a track of radius 900. The agent is then evaluated against a track with a random radius, and this value is used to compare model performances. A random search is conducted until 5% of the search space has been explored, and the best performing combination is used as the final model parameters for a given lambda value.

Assessing the top 3 parameter combinations for each lambda value as below, it is noticed that they generally choose parameters with an early stopping of after lap 160.

The final models are then trained for 25 episodes on a track of random radius each.

### 2.2.4. QRDQN, TRPO, ARS, RecurrentPPO

These models are built from stable baseline 3 environments.

# 3. Results and Evaluation

## 3.1. Evaluation method

Given the varying radius and weather sequences of the question, a fixed environment is needed to assess the performance fairly and accurately across the different models. To do this, an evaluation environment is created which helps us to load fixed weather sequences and radius during state transitions. In total, 81 different weather configurations were curated. The first 39 configurations consists of non-random, pre-determined weather sequences that are meant to help us observe how model reacts to specific changes in weathers. There are 3 main types of weather changes, each across 13 radius intervals (600, 650, … , 1200), giving 39 in total, and they are linear upwards, linear downwards, and flat oscillating. The plot below illustrates how these 3 trajectories differ.

A graph with different colored lines

Description automatically generated

Diagram 6: Wetness per lap for the 3 weather configurations.

The next 42 configurations are randomly generated using the original environment and stored for evaluation purposes. Each model (agent) will be tested on all 81 weather configurations. For each configuration, the agents are ranked based on total rewards, and the top agents are found by summing the count of weather configurations which the agent ranked first in.

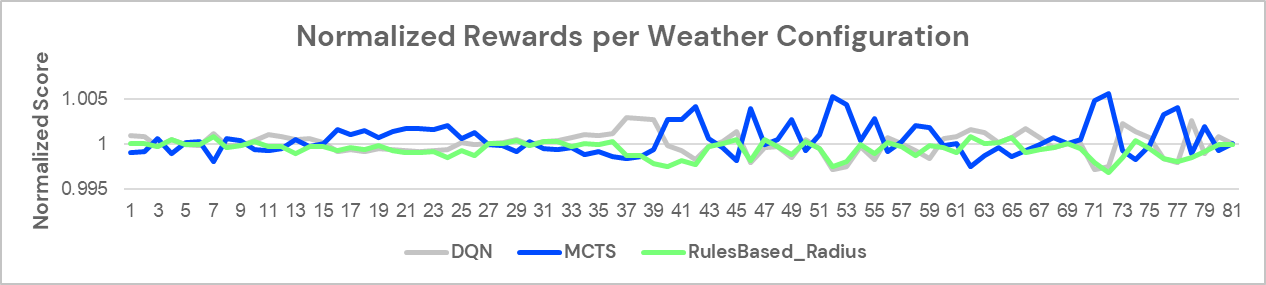
## 3.2. Results and Final Agent Selection for Task 1

The top 3 models in descending order of performance are rules-based (32 wins), open loop MCTS (28 wins) and DQN (21 wins), with performance referring to number of wins out of the 81 weather configurations.

The rules-based model outperformed the other models. In terms of variance, it also displayed the most consistent results across the 81 weather configurations as shown in the chart below.

MCTS was a strong contender. In configurations where both rules based and DQN performed poorly, MCTS was able to give good results (see configurations 4, 7, 45, 62-65, 74). However, the opposite was also true and more evidently so (see configurations 16-24, 40-42, 46, 49, 52-53, 55, 71-72, 76-77, 79).

DQN performed decently. For the most parts, its performance shadows that of rules-based, sometimes peaking over slightly. However, for the others, it was unable to outperform the other 2 models. Even when rules-based is taken out of the ranking, MCTS still have a higher win rate.



The Normalized rewards are calculated using the reward divided by the average rewards of all other agents for that weather configuration. Given the negative nature of rewards, the lower the normalized reward, means a better reward.

**For all reasons above, we decided to use the Rulesbased Agent as our final submission for Task 1.**

## 3.3. Model behaviours

To better understand the results obtained, we analysed the behaviours of the top 3 models for certain weather configurations.

Weather config: 0 (Linear upwards)

A graph with a red line

Description automatically generated

Notable behaviour

* (-11,093) Rules-based will act immediately after valid weather changes
* (-11,102) DQN shows heavy oscillation but is similar to rules-based behaviour.
* (-11,087) MCTS behaviour erratic but seems to react appropriately to weather changes

Weather config: 20 (Linear downwards)

A graph of a staircase

Description automatically generated with medium confidence

Notable behaviour

* (-16,007) Rules-based will act immediately after valid weather changes.
* (-16,009) DQN shows heavy oscillation but is like rules-based behaviour.
* (-16,014) MCTS behaviour erratic but seems to react appropriately to weather changes

Weather config: 37 (Flat oscillating)

A graph with red and blue lines

Description automatically generated

Notable behaviour

* (-18,694) Rules-based responds to 60% Wet weather condition.
* (-18,770) DQN shows heavy oscillation in tyres, which is hard to interpret.
* (-18,588) MCTS takes a different tyre trajectory compared to rules-based, using all 4 tyre types. The difference in results is also much larger here.

# 4. Task 2

## 4.1. Modifications

Modifications to the problem statement in Task 1 is distinguished into 3 individual categories, namely modifications to environment, modifications to action space and modifications to reward.

### 4.1.1. Modifications to Environment

*Track complexity.* A new state is added to the environment called track complexity. Track complexity takes the range of 0 to 1 and is determined whenever the environment is initialized and remains the same throughout the episode.

*Accidents.* Accidents happen when an agent commits to a reckless driving strategy for a specific lap. The rate at which accidents happen is determined by track complexity. The formula for calculating the probability of accident is *1.5% \* (1 + complexity)*, essentially translating to a minimum of 1.5% chance to run into an accident on a track of 0 complexity and a maximum of 3% on the maximum track complexity of 1. When an agent runs into an accident, it is disqualified and removed from the race.

*Participants.* Other agents are also included in the environment as participants of the race. By keeping track of the time taken by each agent to reach a specific point in the race, the placing of the current agent can be determined. The placing is passed as a state in the form of number of cars leading in front of the agent and number of cars trailing behind the agent.

### 4.1.2. Modifications to Action Space

*Driving strategy.* Two driving strategies are available for agents to choose from, namely conservative and aggressive. Agents get a 10% boost to velocity when committing to reckless driving behaviour. Action spaces are effectively expanded to 10 actions compared to that of Task 1 with the addition of driving strategy.

### 4.1.3. Modifications to Rewards

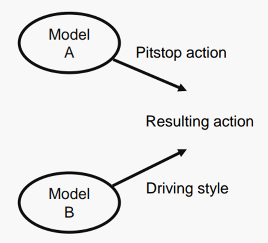
*Final reward.* Reward is given out at the end of the race. The amount of reward is determined by the final placing of each agent. For agent who finished first, a reward of 1000 is given. For agent who finished second, a reward of 500 is given. For agent who finished third, a reward of 400 is given. All agents who rank lower than third placing are given the same reward of 80. In the unlikely event of a tie, agents with the same placing receive the same rewards. Disqualified agents receive no reward, which is 0.

## 4.2. Models

Models used are built using DQN from the stablebaseline3 library. DQN is chosen as it is proven to be one of the best performing models for Task 1.

### 4.2.1. Split Models

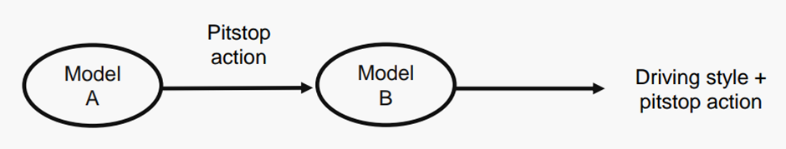
Two individual models are used to make decisions on the pitstop stop action and driving strategy individually. The final action to be taken is obtained by combining the outputs of both models.



The model used to predict the pitstop action is trained using the environment in task 1, since the modifications to the environment does not affect tyre conditions and hence pitstop action. On the other hand, the model used to determine driving strategy is trained on laps completed, track complexity and current placing. While track complexity and current placing are new additions to the environment and hence should be used to train the new model, laps completed is also included to observe if the agent makes different decisions during different parts of the race under the same conditions.

### 4.2.2. Stack Models

A new DQN model is trained on top of a base model, trained using the environment in task 1. Under this configuration, the pitstop action is predicted by the base model. The pitstop action is then passed into the new model to produce an action which combines both pitstop action and driving strategy.



### 4.2.3. DQN Model

A DQN model is trained using the new environment.

## 4.3. Trainings and Results

### 4.3.1. Training

Split models, stack models and the DQN model are trained against an environment with 9 other agents acting as opponents. The 9 agents are:

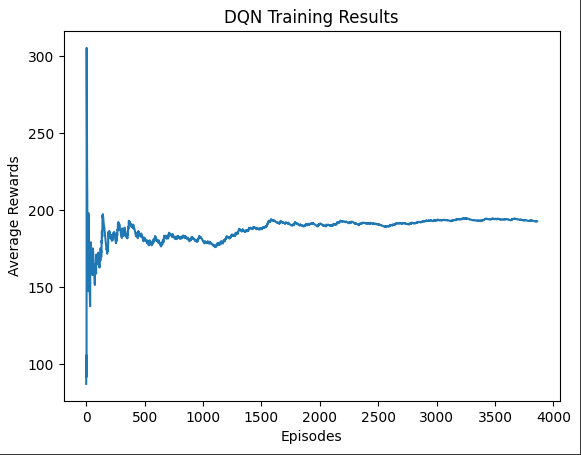
1. Static agent which takes the same action throughout the race
2. Random agent with 0% probability of choosing a reckless driving strategy
3. Random agent with 5% probability of choosing a reckless driving strategy
4. Random agent with 10% probability of choosing a reckless driving strategy
5. Random agent with 20% probability of choosing a reckless driving strategy
6. Agent with DQN model trained in task 1
7. Agent with DQN model trained in task 1 with 5% probability of taking the reckless counterpart of the predicted action
8. Agent with DQN model trained in task 1 with 10% probability of taking the reckless counterpart of the predicted action
9. Agent with DQN model trained in task 1 with 20% probability of taking the reckless counterpart of the predicted action

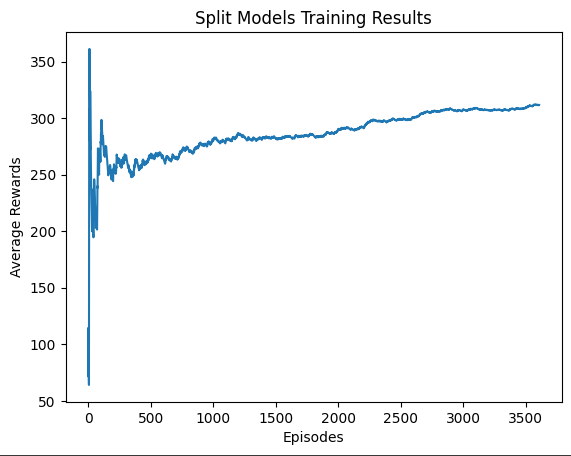
During training, a lap reward of 1 is set to encourage agents to complete as many laps as possible. Rewards for fourth placing and below are also differentiated by decreasing the reward by 5% for every subsequent placing. This configuration is to train agents to continuously improve their placing.

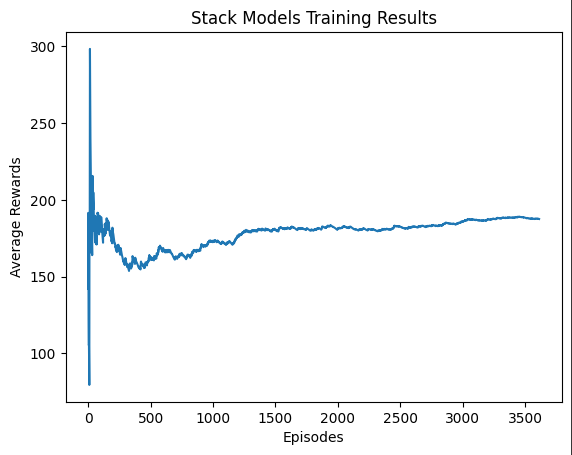
Evaluation is run every 2000 timesteps and training is stopped when model stops improving for 50 consecutive evaluations. During evaluation, agents no longer receive a reward for completing each lap and rewards are no longer differentiated after fourth placing. The environment used during evaluation is to simulate the actual environment that will be used to investigate model performance.

### 4.3.2. Results and Discussions

The following figures shows average rewards for each model during training.

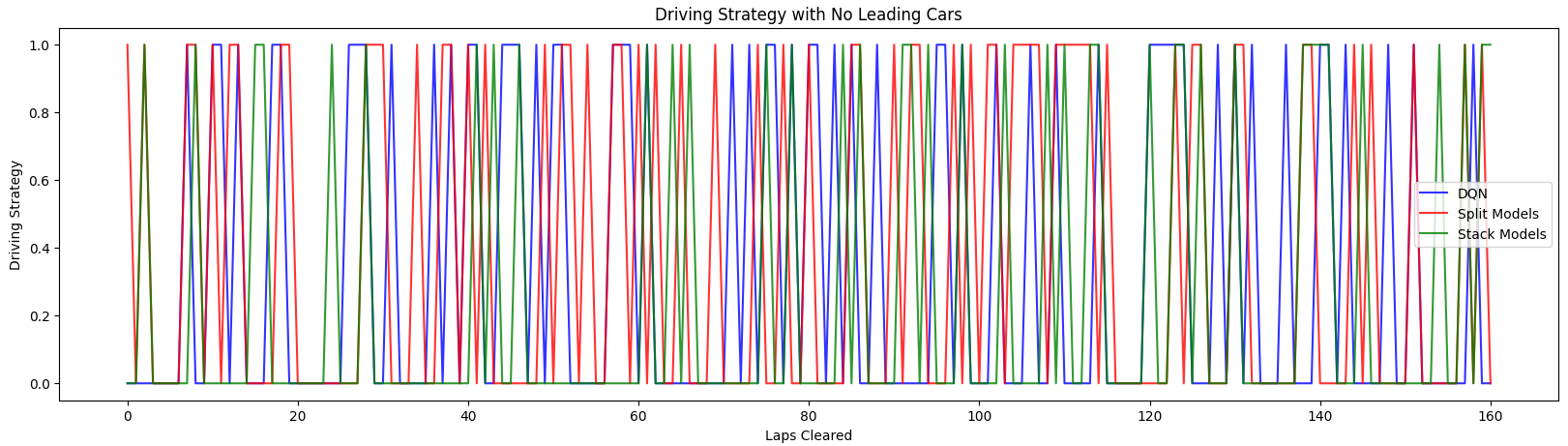


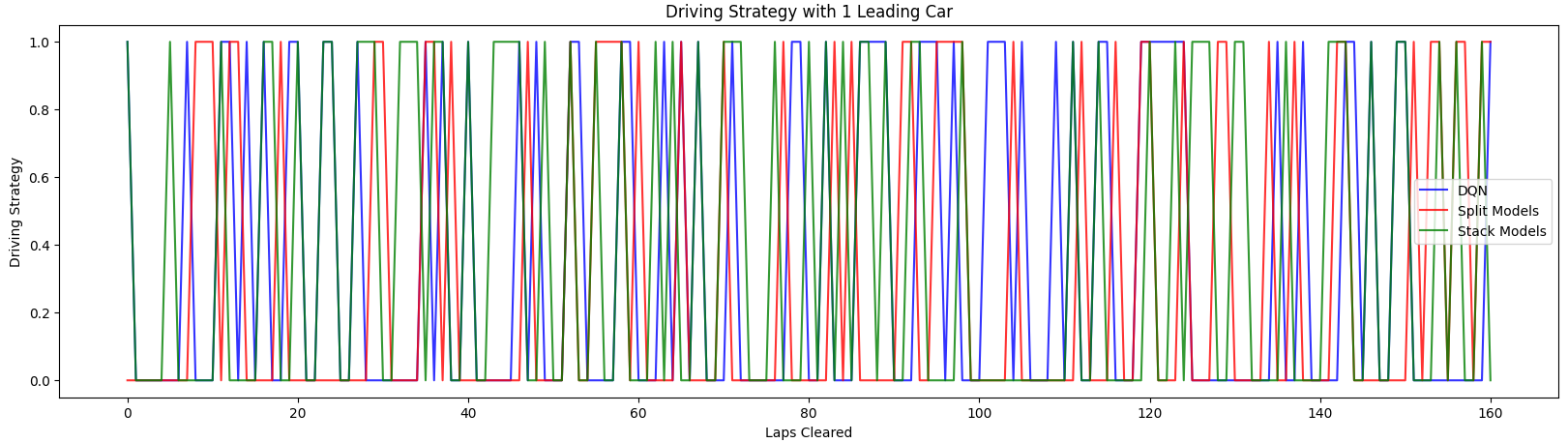


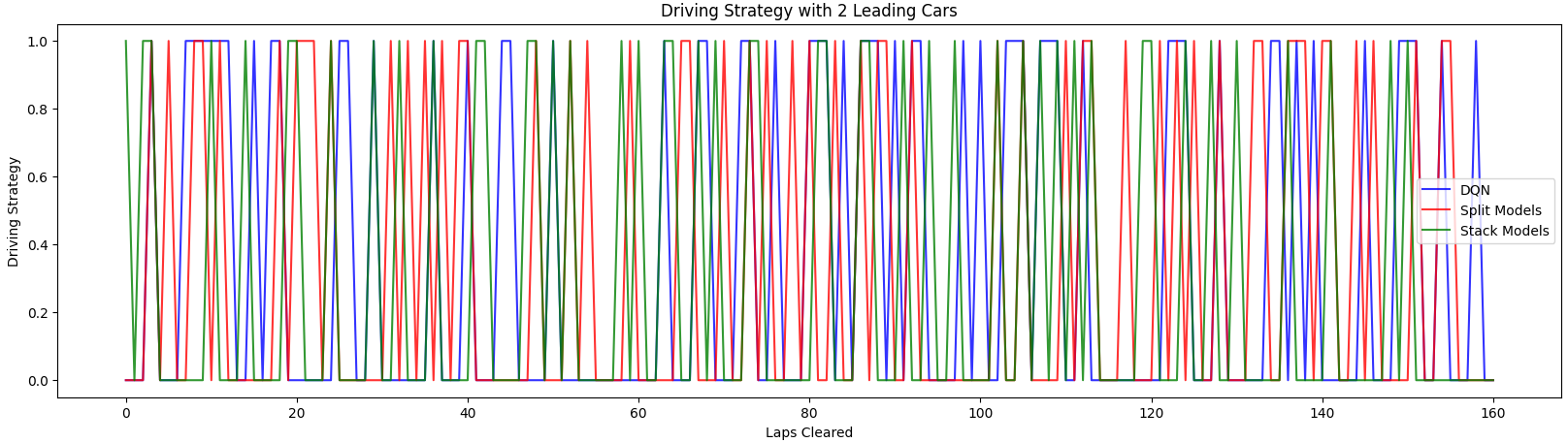


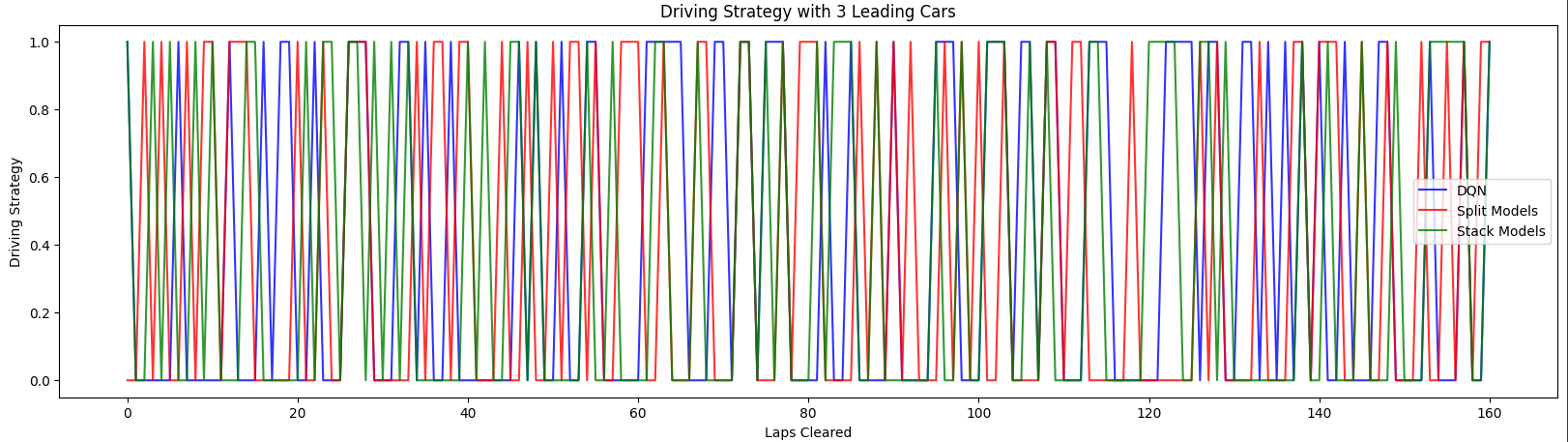
From the graphs above, all 3 models demonstrated convergence as average rewards first increased with training and subsequent stabilized. Split models show the highest average rewards at the end of training, followed by DQN and stack models. DQN and stack models exhibit relatively similar average rewards at the end of training.

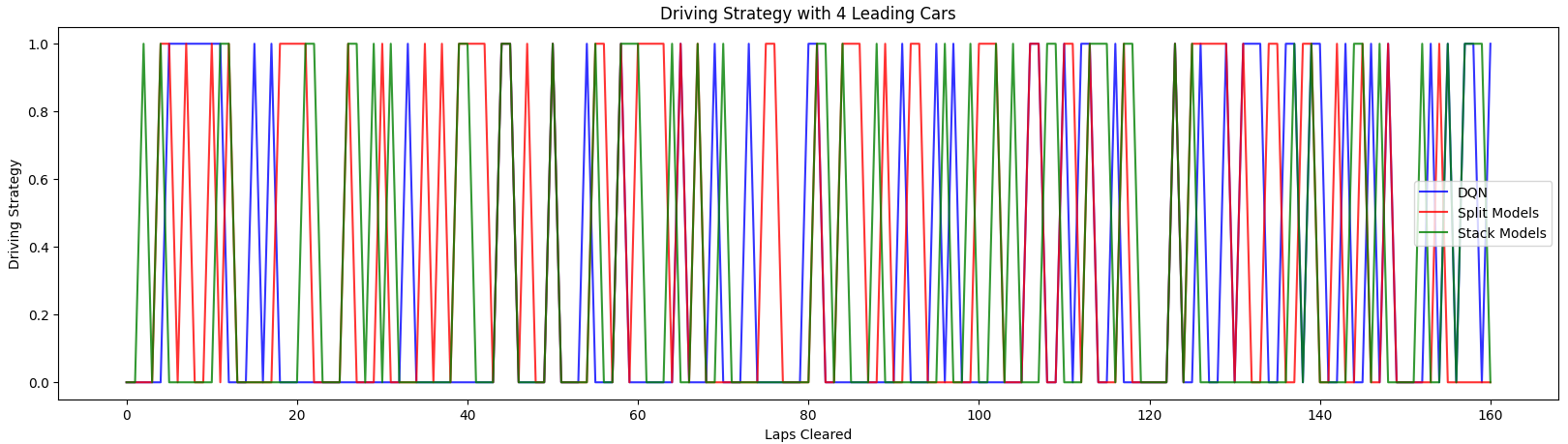
In attempt to gain insights on how each of the 3 models make decisions on driving strategies, actions taken by each model under different laps cleared and current placing are investigated. In the following graphs, 0 indicates conservative driving strategies while 1indicates reckless driving strategies.





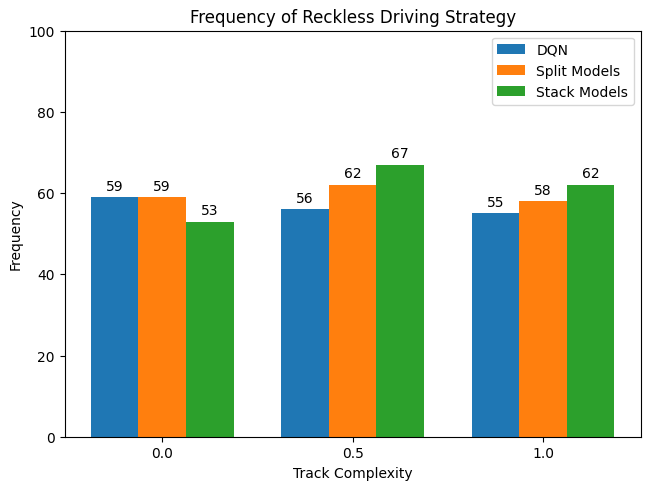




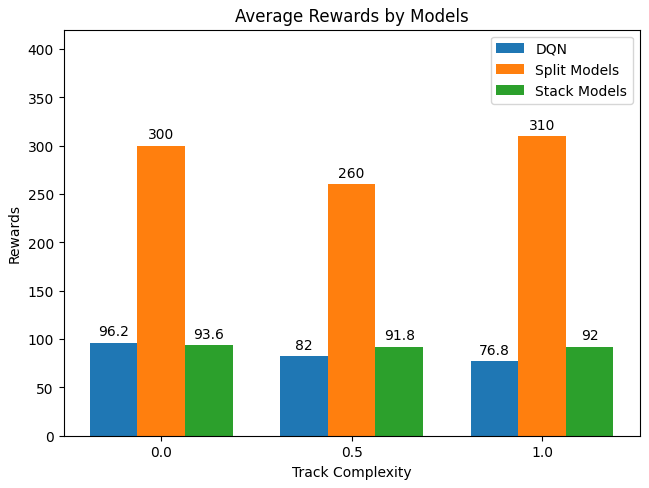


Laps cleared has little to no effect on decisions on driving strategies across all models alike. On the other hand, it is evident that reckless driving strategies are chosen less often than conservative driving strategies when current placing is below fourth placing. This behaviour is consistent with expectations as reward does not increase when improving from fifth to fourth placing.

The relationships between frequency of selecting reckless driving strategies are also investigated. We expect models to choose reckless driving strategies less frequently at higher track complexities. However, such relationship is not evident in models other than DQN. Split models do not seem to prefer one driving strategy or another based on track complexity while stack models favour reckless driving strategies with higher track complexity.

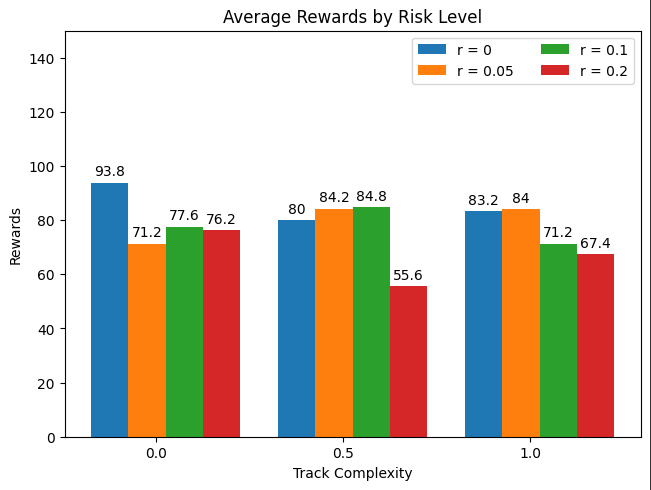


The models are evaluated in 100 races for each track complexity of 0, 0.5 and 1.0. Opponents are the same agents used in training, mentioned above in section 4.3.1. The average rewards obtained by each model across 100 races are shown in the chart below.

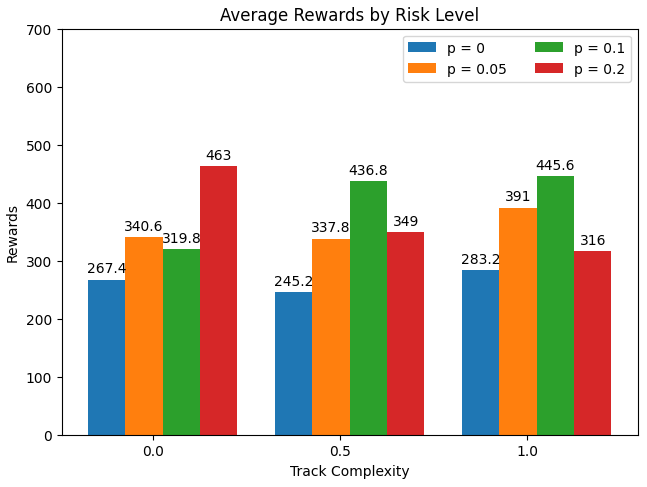


The results above are consistent with what is observed during training, with split models performing the best out of the three models while DQN and stack models achieving similar performances. However, even the best performing model, the stack models, obtained a maximum of around 300, which is lower than the reward given to the third place in the test environment. This finding prompts us to look at competing agents for better performing algorithms.

First, we ran the same evaluation using random agents. The chart below shows that random agents with 0 to 20% chance of adopting a reckless driving strategy have worse performance compared to the trained models.

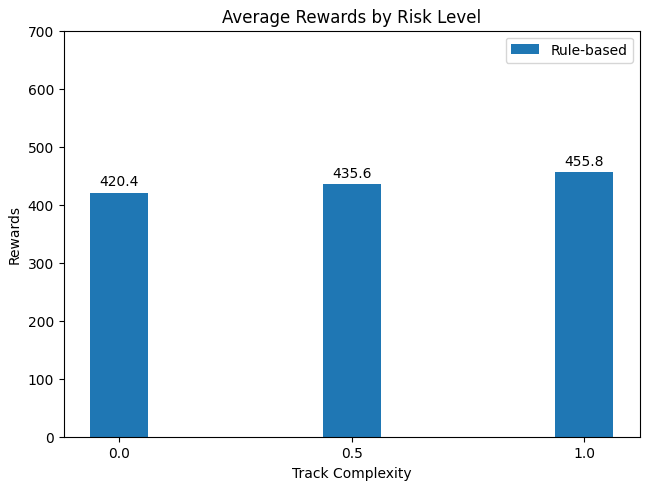


Next, the same evaluation process is run on the probabilistic agents with DQN model trained on task 1.



As shown in the chart above, we can conclude that the top 3 spots are usually taken up by these probabilistic agents as all agents obtained average rewards of above 300 for across all complexities.

Combining the findings on driving strategies does not change based on laps cleared and that probabilistic agents with higher risk levels perform better at lower track complexity, we propose a simple rule-based probabilistic agent which improves on existing probabilistic agents. This rule-based probabilistic agent adopts a dynamic risk level which decreases with track complexity and avoids adopting a reckless driving strategy when there are no cars leading in front of it. The performance of the rule-based probabilistic agent with risk level ranging from 0.1 to 0.5 is shown below.



It is evident that this rule-based probabilistic agent performs better than all other agents, concluding that using a DQN model trained on pitstop actions and combining the pitstop action with a rule-based probabilistic approach to driving strategy is best suited for the modified environment.

# References

Piccinotti, D., Likmeta, A., Brunello, N., & Restelli, M. (2021). *Online planning for F1 race strategy identiﬁcation - github pages*. Online Planning for F1 Race Strategy Identification. <https://prl-theworkshop.github.io/prl2021/papers/PRL2021_paper_1.pdf> Taken from: Journal: Association for the Advancement of Artificial Intelligence (AAAI), [www.aaai.org](http://www.aaai.org/)

# Appendix A: Rollout Policies and their Details

**Random rollout policy:** The random rollout policy simply selects actions randomly. This policy is simple to implement, but it is not very effective.

**Opponent-aware rollout policy:** The opponent-aware rollout policy uses a model of the opponent's behavior to select actions. This model is trained on data from previous races. The opponent-aware rollout policy is more effective than the random rollout policy, but it is more difficult to implement.

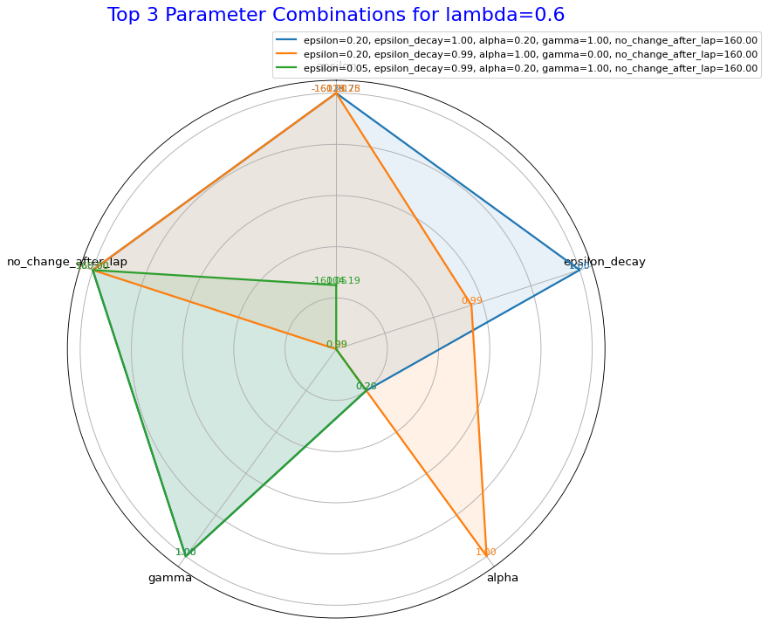
**Static opponent policy:** The static opponent policy assumes that the opponents will follow the same strategy throughout the race. This policy is simple to implement, but it is not very effective, as the opponents are likely to adjust their strategy during the race.

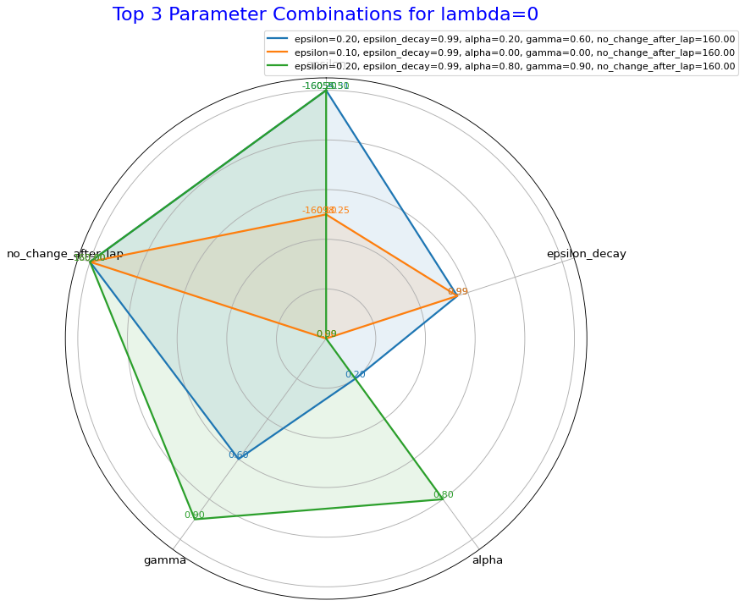
**Dynamic opponent policy:** The dynamic opponent policy updates its model of the opponents' behavior based on their actions during the race. This policy is more effective than the static opponent policy, but it is more difficult to implement.

# Appendix B: TD(Lambda) HPT results

­A diagram of a triangle

Description automatically generatedA diagram of a hexagon with different colored triangles

Description automatically generatedA diagram of a triangle with different colored triangles

Description automatically generatedA diagram of a diagram

Description automatically generatedA diagram of a hexagon

Description automatically generated