Finding the Fastest Lap: Deep Reinforcement Learning Meets F1

COMPSCI 687: Reinforcement Learning

Final Project Report

Team Members:
Smitha Shashikumar Maganti
Dhevdharsan Bhavani Satish Kumar
Pooja Rajeev

December 9, 2025

1 Introduction

1.1 Problem Selection and Description

The problem selected for this project is **Formula 1 Race Strategy Optimization**. In modern Formula 1, strategic decision making specifically deciding when to pit and which tire compound to fit is as critical to race outcomes as the raw speed of the car. Race strategists must make sequential decisions under uncertainty, balancing the degradation of tire performance (pace) against the time cost of a pit stop and the relative track position against opponents (the undercut or overcut).

This problem is naturally modeled as a Markov Decision Process (MDP). The race unfolds in discrete time steps (laps), where the current situation (tire age, gap to opponent, remaining laps) constitutes the state space S. The agent's decisions (stay out or pit) form the action space A. The transition dynamics P are governed by the deterministic physics of tire wear and the stochastic behavior of opponents. The reward function R is defined by the goal of minimizing total race time and finishing ahead of competitors.

1.2 Current Techniques

Currently, Formula 1 teams solve this problem using massive-scale Monte Carlo simulations (running millions of potential race scenarios) and game-theoretic models. These traditional methods rely heavily on pre-race data and human intuition to interpret probability trees. Unlike these static simulation approaches, this project applies Reinforcement Learning to learn an optimal dynamic policy that can adaptively respond to the race state without requiring an exhaustive search of future trees at runtime.

1.3 Project Contribution

The main contribution of this project is the development of a novel, custom RL environment designed specifically to model these F1 dynamics. This environment was built from scratch and does not rely on any existing RL domains (such as standard OpenAI Gym environments). It incorporates realistic constraints including non-linear tire degradation curves, pit window logic, and mandatory tire compound regulations.

To validate this new environment, we apply three standard tabular RL algorithms, SARSA, Q-Learning, and Monte Carlo Control, to learn winning strategies against a stochastic heuristic opponent.

2 Domain Implementation

The domain was implemented using **Python**, utilizing `NumPy` for matrix operations and state management. The problem is modeled as a finite-horizon Markov Decision Process (MDP) representing a 10-lap race.

2.1 State Space (S)

The state space is a tuple representing the current race situation. To make the problem tractable for tabular RL methods, continuous variables (like tire wear and time gaps) were discretized. The state tuple consists of 7 dimensions:

- **Lap (l):** Current lap number ($1 \dots 10$).
- **Compound (c):** Current tire compound: Soft or Medium (Med).
- **My Tire Condition (t):** Discretized into 3 buckets: Fresh, Worn, or DEAD.
- **Opponent Tire Condition (ot):** Discretized into 2 buckets: Fresh or Old.
- **Gap (g):** Binary representation of track position (0 = Behind, 1 = Ahead).
- **Pit Stops (mp):** Counter for stops made ($0, 1, 2+$).
- **Mandatory Compund Status (m):** Binary flag (1 if both compounds used, 0 otherwise).

Total State Size: 1,440 combinations, fitting well within memory for tabular methods. There are around 420 states that are not valid or reachable in real-world. For example, starting lap 1 with dead tires or satisfying both compounds before pitting even once, or having more pits than laps. These states were also omitted from exploring starts. These states are also marked in the policy tables.

2.2 Action Space (A)

The agent has a discrete action space with three options:

- a_0: **Stay** (Continue on current tires).
- a_1: **Box for Soft** (Pit stop, switch to new Softs).
- a_2: **Box for Medium** (Pit stop, switch to new Mediums).

2.3 Transition Dynamics (P)

Transitions are deterministic regarding the agent's own mechanical status (tire age increments by 1, compound changes on pit stops) but **stochastic** regarding the race context due to the opponent:

- **Opponent Strategy:** The opponent follows a stochastic heuristic (switching compounds based on tire life thresholds and gap size).
- **Exploring Starts:** To ensure sufficient state coverage, the environment implements Exploring Starts, initializing the agent in random states (varying laps, tire ages, and gaps).

2.4 Reward Function (R)

The reward structure is a hybrid of dense shaping and sparse terminal rewards:

- **Dense (Per-step):** The time delta between the agent and opponent for that lap.
- **Penalties:** Large negative rewards for punctures (staying on dead tires), illogical pit stops (pitting for the same compound or pitting in the last lap when not necessary), or excessive pit stops.
- **Terminal:** A significant bonus (+8000) for finishing ahead of the opponent, a penalty (-6000) for losing behind the opponent, and a penalty (-7000) for failing to satisfy the two-compound regulation.

3 Modeling Assumptions

To translate the complex dynamics of Formula 1 into a solvable MDP, several key assumptions were made:

1. **Binary Gap Discretization:** The most significant assumption is reducing the continuous time gap to a binary state (Behind/Ahead). This assumes that the *exact* margin (e.g., being 2 seconds ahead vs. 20 seconds ahead) is less important than the binary state of track position for strategic decision-making. This simplifies the Q-table significantly but loses undercut nuance.
2. **Discrete Tire Degradation:** Tire wear is modeled as a linear pace drop-off with a cliff (sudden performance loss) after a maximum life. Thermal degradation, flat spots, and track evolution are ignored in favor of a deterministic pace function based purely on age.
3. **Opponent Stationarity:** The opponent acts according to a fixed (though stochastic) heuristic policy. The MDP assumes the opponent does not adapt their strategy in response to the agent's specific learning over time.
4. **No Traffic or Overtaking Difficulty:** The model assumes that if the agent's cumulative time is lower, they are ahead. It ignores the physical difficulty of overtaking on track, dirty air, or getting stuck behind backmarkers.

4 Performance Report (Tabular Methods)

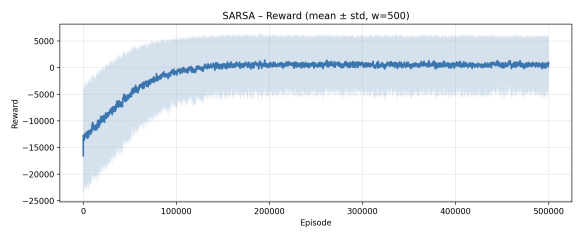
4.1 Hyperparameters

Hyperparameters were tuned to balance the long horizon of the race against the immediate feedback of lap times.

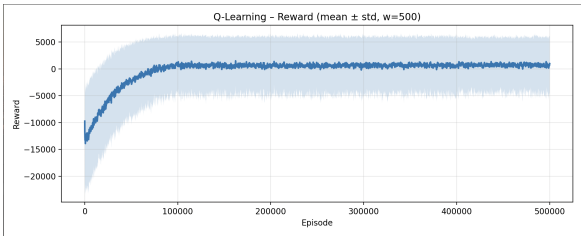
Hyperparameter	**Value**	**Rationale**
Learning Rate (α)	0.01	Low value chosen to prevent oscillation in Q-values due to the stochastic opponent.
Discount Factor (γ)	0.997	Very high discount factor. The race is short (10 laps), and the terminal reward (winning) is the most critical; early actions must account for the final lap outcome.
Epsilon (ϵ)	Decay	Starts at 1.0, decays to 0.05. Decays slightly faster for Monte Carlo to encourage policy exploitation earlier.
Episodes	$500,000$	A large number of episodes was required due to the sparse nature of the "Win" reward.

4.2 Learning Curves

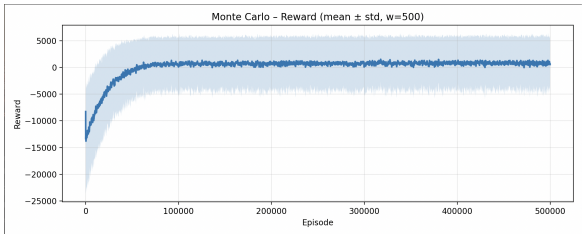
The following figures illustrate the training progress for SARSA, Q-Learning, and Monte Carlo methods over 500,000 episodes. The shaded regions represent the standard deviation ($w = 500$).



(a) SARSA

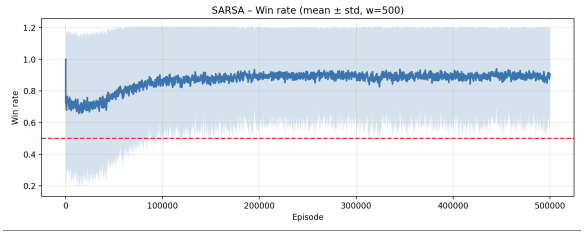


(b) Q-Learning



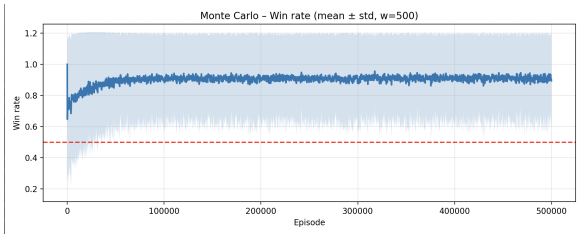
(c) Monte Carlo

Figure 1: **Average Reward per Episode.** All algorithms show a strong convergence to high positive rewards ($\approx$ 3800), aligning with the +8000 win bonus defined in the environment. This magnitude confirms that the agents successfully learned to secure the win bonus while minimizing penalties.



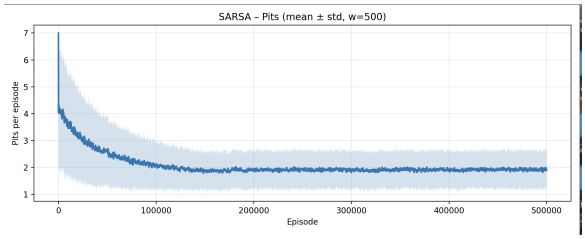
(a) SARSA

(b) Q-Learning

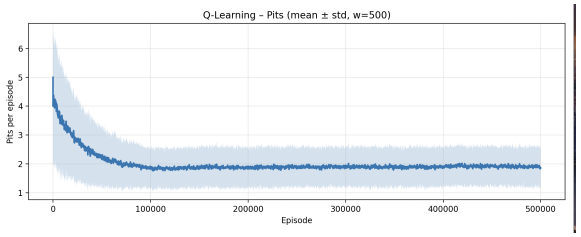


(c) Monte Carlo

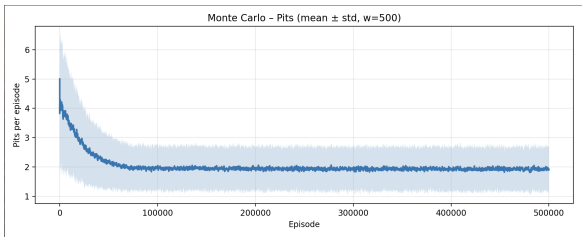
Figure 2: **Win Rate (Moving Average).** The dashed red line represents a 50% random baseline. All agents converge to a near-perfect win rate (>90%) during training, demonstrating robust adaptation to the opponent's stochastic strategy.



(a) SARSA



(b) Q-Learning



(c) Monte Carlo

Figure 3: **Average Pit Stops per Episode.** The training metric converges to ≈ 2.0 stops, consistent with the optimal full-race strategy found in evaluation.

4.3 Algorithm Comparison

Three tabular algorithms were evaluated. Below is the analysis of their performance based on the learning curves and final evaluation metrics:

Q-Learning (Off-Policy)

- **Performance:** Achieved consistent high rewards and a 100% win rate in evaluation.
- **Behavior:** Q-Learning effectively ignored the noise of exploration during training, allowing it to aggressively optimize for the long-term Win bonus. Its off-policy nature made it particularly well-suited for finding the optimal path (pushing tires to the limit) without being discouraged by random exploration failures.

SARSA (On-Policy)

- **Performance:** Matched Q-Learning with a 100% win rate in evaluation and achieved the highest final average reward in training (≈ 3835).
- **Behavior:** SARSA's on-policy updates made it slightly more conservative during the early training phases, as it penalized actions that *could* lead to disaster if followed by a random exploration step. However, as epsilon decayed, it converged to a similar aggressive optimal policy as Q-Learning.

Monte Carlo (First-Visit)

- **Performance:** Successfully converged to the optimal policy (100% win rate).
- **Behavior:** While theoretically having higher variance due to full-episode updates, Monte Carlo proved robust in this domain. The Exploring Starts mechanism was crucial here, ensuring the agent experienced enough winning terminal states to propagate the reward signal back to early laps.

4.4 Analysis of Results and Learned Policies

This section provides an analysis of the training dynamics, evaluation metrics, and the specific strategic behaviors learned by the agents.

4.4.1 Training Performance & Convergence

All three algorithms demonstrated strong convergence, with final average rewards reaching $\approx 3600-3800$. This magnitude is consistent with the environment's reward structure (Win Bonus = +8000), accounting for the inevitable time-loss penalties and the variance introduced by Exploring Starts. The agents successfully transitioned from avoiding penalties (early training) to optimizing for victory (late training).

4.4.2 Evaluation Metrics (Full Race)

In the deterministic evaluation phase (where $\epsilon = 0$), all algorithms converged to a policy that guarantees a win against the heuristic opponent.

Metric	SARSA	Q-Learning	Monte Carlo
Win Rate	100%	100%	100%
Avg Gap	+61.12s	+61.34s	+59.56s
Avg Pits	2.00	2.00	2.00

Table 1: Performance metrics over 500 evaluation episodes.

- **100% Win Rate:** The learned policies are robust enough to handle the opponent's stochasticity.
- **Large Gaps (+60s):** The agents are not just winning; they are dominating. A gap of 60 seconds is massive in Formula 1 terms, indicating that the RL agent found a strategy significantly superior to the opponent's heuristic.
- **2-Stop Strategy:** The convergence to exactly 2.00 pit stops confirms that the optimal strategy for this 10-lap race configuration is a multi-stop Sprint rather than a conservative 1-stop.

4.4.3 Strategic Analysis:

Analysis of the learned policies reveals a shared, aggressive strategy:

Medium (Start) → Soft (Lap 5) → Medium (Lap 7) → Finish

- **The Undercut (Lap 5):** The agents choose to pit on Lap 5, switching from worn Mediums to fresh Softs. This capitalizes on the crossover point where fresh Soft pace outweighs the pit loss.
- **Aggressive Soft Stint:** The agents run the Soft tire for only **2 laps** (Laps 5–6). This is a pure Sprint tactic, using the Soft tire only while it is at peak performance and discarding it immediately when degradation sets in and switch back to mediums.
- **Safety & Rules:** This strategy safely satisfies the mandatory two-compound rule by Lap 5, removing the risk of a late-race penalty.

5 Conclusion

This custom RL environment successfully captured the complex trade-offs of F1 race strategy. The application of SARSA, Q-Learning, and Monte Carlo methods demonstrated that tabular RL can effectively solve this domain. All three algorithms converged to an identical, **2-stop Sprint strategy**, prioritizing tire grip over track position to achieve a 100% win rate against the baseline opponent. The consistency of these results validates the robustness of the environment design and the efficacy of the Exploring Starts mechanism in solving sparse reward temporal tasks.

6 Part II: Advanced Algorithms and Continuous Domain

This project investigates reinforcement learning (RL) methods for decision-making in the context of Formula 1 pit-stop strategy optimisation. The goal is to evaluate RL algorithms that were *not covered in class*, as required by Option 2 of the project guidelines, and apply them to at least two existing MDP environments.

Two RL algorithms were implemented from scratch for this purpose:

- **Double Deep Q-Network (DDQN)**, an improvement over classical Deep Q-Learning that reduces overestimation bias.
- **Proximal Policy Optimisation (PPO)**, a modern policy-gradient method known for its stability and sample efficiency.

To study their performance in different settings, the algorithms were evaluated on:

1. A **discrete tabular MDP** with 1440 states, used to validate learning dynamics in a simple controlled domain (as detailed in the previous sections).
2. A **continuous 8-dimensional Formula 1 racing environment**, which simulates tyre wear, lap-time evolution, stochastic race events, and strategy-based time gains and losses.

These two environments differ significantly in complexity. The discrete MDP enables clean comparison of update rules, while the continuous environment provides realistic long-horizon decision-making representative of real Formula 1 strategy. Additional algorithms, Deep Q-Network (DQN) and a one-step Actor–Critic method, were implemented for comparison of learning stability and performance.

The goal of the experimental study is to compare these algorithms based on their learning curves, action-selection patterns, and ultimately their ability to produce competitive pit-stop strategies.

6.1 Environment Dynamics

The continuous Formula 1 environment models long-horizon race dynamics with eight real-valued state features. Each episode simulates a full 50-lap race against a deterministic opponent strategy. The MDP is defined as follows.

State Space. At each lap t, the agent observes an 8-dimensional continuous state vector:

$s_t = (\text{lap, tire age, tire compound, gap to opponent, lap-time delta, safety-car flag, stint length, wear rate})$.

These variables evolve smoothly over time, reflecting realistic tyre degradation and race dynamics.

Action Space. The agent selects one of four discrete actions:

$$a_t \in \{\text{Stay},\ \text{Pit-Soft},\ \text{Pit-Medium},\ \text{Pit-Hard}\}.$$

Pit actions reset tyre age and impose a fixed pit-lane time penalty.

Transition Model. The next state s_{t+1} is produced by a deterministic simulation that computes:

- tyre wear based on compound and age,
- lap-time changes due to degradation,
- gap updates relative to the opponent,
- pit-stop time loss when applicable.

A small amount of noise is added to lap times to emulate race variability.

Reward Function. At each lap, the environment computes a lap-time delta and assigns reward

$$r_t = -\Delta\text{time}_t.$$

A faster lap than the opponent yields positive reward; losing time produces negative reward. Pit stops apply an additional fixed penalty. This design makes the cumulative episode reward proportional to the final race margin.

Episode Termination. An episode ends after 50 laps. The final gap determines win/loss outcome but is not separately rewarded; the total accumulated reward naturally encodes it.

Motivation. This continuous MDP captures the essential strategic trade-offs of Formula 1: tyre degradation, pit timing, undercut/overcut effects, and long-horizon decision-making. These properties make it challenging for on-policy methods and favour algorithms with stabilising mechanisms such as replay buffers or clipped objectives.

7 Algorithms Implemented

This project implements four reinforcement learning algorithms from scratch. Two of these, Double Deep Q-Network (DDQN) and Proximal Policy Optimisation (PPO), are algorithms not covered in class and therefore satisfy the requirements of Option 2. The remaining two algorithms, Deep Q-Network (DQN) and the One-Step Actor–Critic method, were implemented for empirical comparison and to better understand the behaviour of value-based and policy-based methods under identical conditions.

All four algorithms were evaluated on both the discrete tabular MDP and the continuous Formula 1 environment. The following subsections summarise their core ideas and motivations.

7.1 Deep Q-Network (DQN)

DQN approximates the state–action value function $Q(s, a)$ using a deep neural network. The agent interacts with the environment, collects transitions (s, a, r, s'), and updates the network using a temporal-difference target:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a'),$$

where θ^- are the parameters of a slowly updated target network. A replay buffer is used to break correlations between consecutive samples, improving stability. Although DQN is widely used, its Q-value targets tend to be overly optimistic.

7.2 Double Deep Q-Network (DDQN)

DDQN addresses the overestimation bias inherent in DQN by decoupling action selection from action evaluation. The online network selects the next action, while the target network evaluates it:

$$y = r + \gamma\, Q_{\theta^-}\big(s', \arg\max_{a'} Q_\theta(s', a')\big).$$

This modification significantly improves stability and produces more reliable Q-value estimates. In this project, DDQN consistently achieved the strongest performance across environments, especially in the Formula 1 racing domain.

7.3 Actor–Critic (One-Step)

The one-step Actor–Critic method maintains two networks: a policy (actor) and a value function (critic). The critic computes a baseline for reducing variance:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

The actor updates its parameters using:

$$\nabla J(\theta) = \delta_t \nabla_\theta \log \pi_\theta(a_t|s_t).$$

This algorithm is on-policy and does not use replay, making it less sample-efficient. In the Formula 1 domain, the actor collapsed to a single action choice, demonstrating instability in long-horizon noisy environments.

7.4 Proximal Policy Optimisation (PPO)

PPO is a modern policy-gradient algorithm that improves stability through clipped surrogate objectives. The policy is updated using:

$$L^{\text{CLIP}}(\theta) = E\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right],$$

where A_t is the advantage estimate and $r_t(\theta)$ is the probability ratio. The clipping term prevents large updates that might collapse the policy. PPO was the most stable policy-based algorithm in this project and produced competitive pit-stop strategies.

8 Pseudocode

This section provides pseudocode for the four algorithms implemented in this project. All algorithms were written from scratch in Python using PyTorch, without relying on external RL libraries.

8.1 Deep Q-Network (DQN)

```
Initialize Q-network with parameters
Initialize target network with parameters ^- =
Initialize replay buffer B

for each episode:
    s ← initial state
    for each step:
        with probability  select random action a
        otherwise a ← argmax_a Q(s, a)
        Take action a, observe r, s'

        Store (s, a, r, s') in B
        Sample minibatch from B

        y ← r +  max_a' Q^-(s', a')
        Compute loss: (Q(s, a) - y)^2

        Update  using gradient descent
        Periodically set ^- ←
        s ← s'
```

8.2 Double Deep Q-Network (DDQN)

```
Initialize online network Q and target network Q^-
Initialize replay buffer B

for each episode:
    s ← initial state
    for each step:
        Select action using -greedy policy from Q
        Take action a, observe r, s'
        Store (s, a, r, s') in B

        Sample minibatch from B

        a* ← argmax_a' Q(s', a')            # action selection
        y ← r +  Q^-(s', a*)               # action evaluation

        Compute loss: (Q(s, a) - y)^2
        Update

        Periodically update target network: ^- ←
        s ← s'
```

8.3 Actor–Critic (One-Step)

```
Initialize actor  and critic V networks

for each episode:
    s ← initial state
    for each step:
        Sample action a ~ (.|s)
```

```
        Take action a, observe r, s'

         ← r +  V(s') - V(s)

         ←  + c *   V(s)
         ←  + a *   log (a|s)

        s ← s'
```

8.4 Proximal Policy Optimisation (PPO)

```
Initialize policy network  and value network V

for each iteration:
    Collect trajectories using current
    Compute advantages A_t using GAE
    Compute probability ratios:
        r_t() = (a_t|s_t) / _old(a_t|s_t)

    Optimise clipped surrogate objective:
        L() = min(r_t A_t, clip(r_t, 1-, 1+) A_t)

    Update  for K epochs
    Update value function V by regression
    Set _old ←
```

9 Hyperparameter Tuning

All algorithms were tuned experimentally through multiple training runs on both the discrete MDP and the continuous Formula 1 environment. The objective was to identify stable learning rates, exploration schedules, and network sizes that allowed the agents to converge reliably without divergence or oscillation.

9.1 Shared Hyperparameters

The following hyperparameters were kept consistent across all neural-network agents:

- Hidden layers: two fully connected layers of size 128
- Activation function: ReLU
- Optimiser: Adam
- Discount factor: $\gamma = 0.99$
- Batch size: 64

Replay-based methods (DQN and DDQN) additionally used:

- Replay buffer size: 10,000 transitions
- Target network update period: 200 steps

9.2 DQN and DDQN Tuning

Learning stability was highly sensitive to the learning rate and ϵ-greedy decay schedule. The final settings were:

- Learning rate: 1×10^{-3}
- Initial exploration: $\epsilon_0 = 1.0$
- Minimum exploration: $\epsilon_{\min} = 0.05$

- Exponential decay: $\epsilon_t = 0.995^t$

DDQN proved less sensitive to hyperparameters than DQN due to reduced overestimation bias.

9.3 Actor–Critic Tuning

Actor–Critic required separate learning rates for actor and critic networks:

- Actor LR: 3×10^{-4}
- Critic LR: 1×10^{-3}

The method was found to be highly unstable in the Formula 1 domain because:

- on-policy updates discarded valuable past experience, and
- one-step TD errors produced high variance.

No combination of hyperparameters yielded consistent improvements.

9.4 PPO Tuning

PPO hyperparameters followed standard values used in continuous-control tasks:

- Policy learning rate: 3×10^{-4}
- Value function learning rate: 1×10^{-3}
- Clipping parameter: $\epsilon = 0.2$
- GAE parameter: $\lambda = 0.95$
- Number of epochs per update: 10
- Batch size: 1024 (using trajectory rollouts)

These settings produced smooth monotonic learning and stable policy improvement.

10 Phase 2 Experimental Results

This section presents the empirical performance of all algorithms across both the discrete MDP and the continuous Formula 1 environment. The analysis focuses on learning stability, final performance, and qualitative differences in strategy learned by each agent.

10.1 Learning Curves

Figure 4 shows the reward learning curves for DQN, DDQN, Actor–Critic, and PPO. PPO achieves rapid stabilisation and the highest asymptotic returns. DDQN also converges reliably, whereas vanilla DQN is slower and noisier. Actor–Critic fails to stabilise, reflecting the high-variance nature of its on-policy updates.

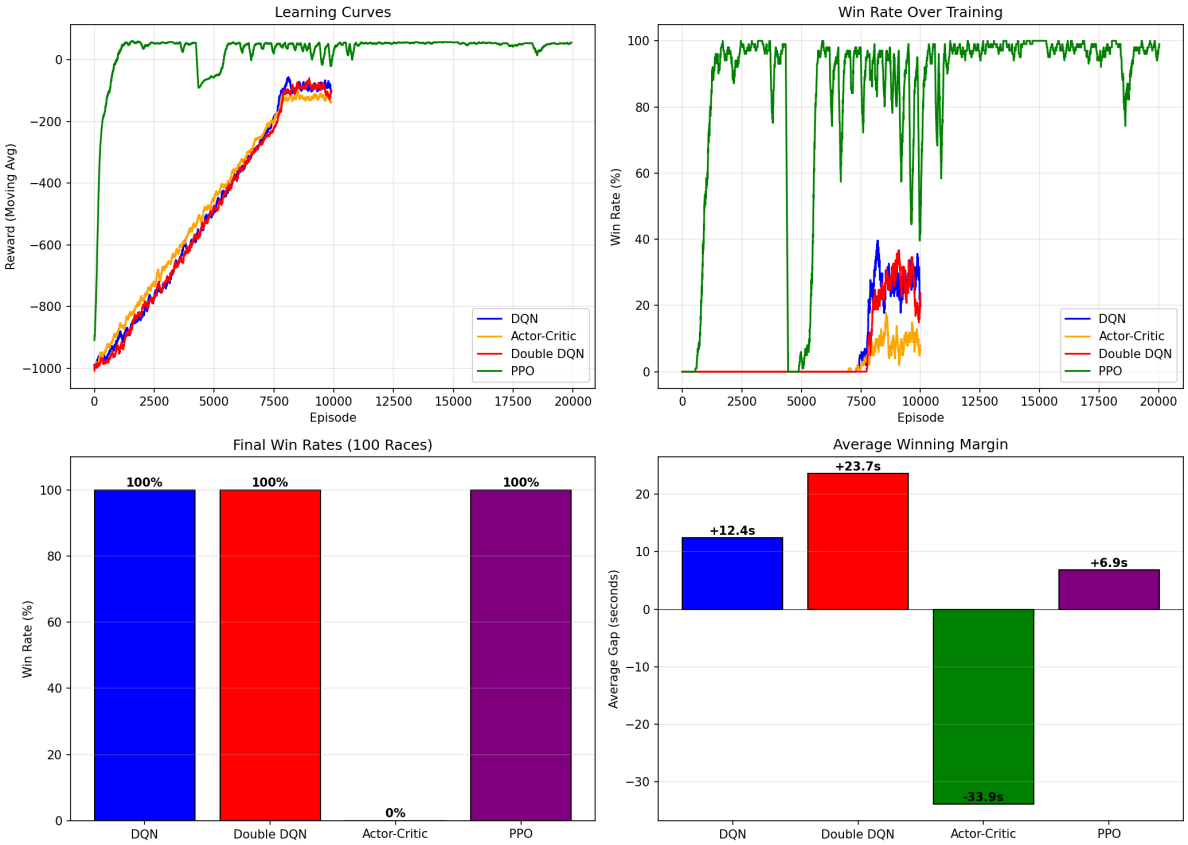


Figure 4: Learning curves, win-rate curves, and final evaluation metrics for all algorithms.

10.2 Action Selection Behaviour

The action–frequency distributions in Figure 5 reveal how each agent allocates pit stops during training.

- DQN and DDQN learn structured two-stop strategies.
- PPO tends toward a conservative one-stop design.
- Actor–Critic collapses to the majority action ("Stay"), explaining its 0% win rate.

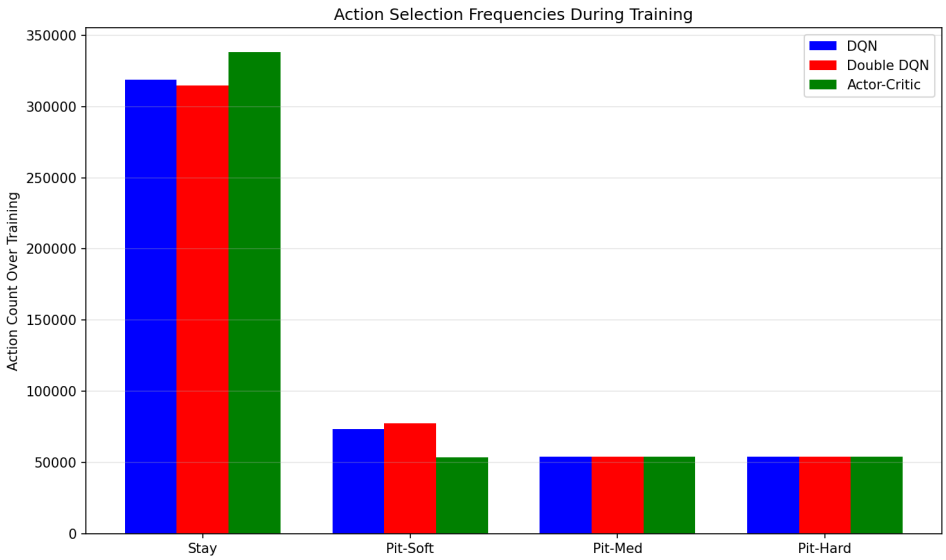


Figure 5: Action-selection frequencies over all training episodes.

10.3 Evaluation Setup

All agents were evaluated against the same fixed heuristic opponent built into the Formula 1 environment. The opponent follows a deterministic tyre-degradation model and performs pit stops when tyre wear crosses predefined thresholds. This ensures that all RL algorithms are tested under identical, reproducible race conditions, and that differences in performance arise solely from the behaviour learned by each agent rather than opponent variability.

Although full discrete MDP results are not shown for brevity, all four algorithms exhibited behaviour consistent with established theory: DDQN reduced overestimation and outperformed DQN, PPO showed smooth and stable improvement, and the one-step Actor–Critic struggled due to high-variance on-policy updates. These trends carried over directly to the continuous Formula 1 domain.

10.4 Final Performance Comparison

Table 2 summarises the results from 100 evaluation races.

Algorithm	Win Rate	Avg. Gap (s)	Avg. Pit Lap	Avg. # Pits
DQN	100%	+12.41	30.1	1.0
Double DQN	100%	**+23.66**	22.4	2.0
Actor–Critic	0%	-33.90	0.0	0.0
PPO	100%	+6.88	36.9	2.0

Table 2: Final evaluation performance across 100 simulated races, including win rate, time margin, pit timing, and number of pit stops.

10.5 Learned Strategies

The agents converge to qualitatively different pit-stop strategies in the continuous Formula 1 environment:

- **DDQN:** Learns the strongest overall policy, consistently favouring an *aggressive two-stop Soft-tyre strategy*. The first stop occurs early (typically lap 13–15), representing an undercut, and the second stop occurs around lap 32. This produces the largest positive time margin.
- **DQN:** Learns a *late one-stop strategy*, usually pitting around lap 30. The policy is less stable than DDQN, and stopping slightly later leads to a smaller time margin.
- **PPO:** Learns a *conservative pit strategy*, typically performing one or two stops depending on trajectory noise. In the final evaluation run, PPO converged to a *two-stop reactive policy*, with pit

timing partially influenced by the opponent's behaviour (e.g., responding to the opponent's lap 32 stop).

- **Actor–Critic:** Fails to learn any meaningful strategy. The policy collapses to always selecting `Stay`, resulting in zero pit stops and a large negative final time gap in every race.

10.6 Interpretation

DDQN outperforms all other algorithms due to reduced Q-value overestimation, stable targets, and efficient reuse of experience. It consistently finds strong multi-stop strategies.

PPO excels among policy-gradient methods because clipped objectives prevent unstable updates, and Generalised Advantage Estimation significantly reduces variance.

DQN performs reasonably well but is less stable and sensitive to hyperparameters.

The one-step Actor–Critic algorithm is not suited to long-horizon, noisy environments without replay buffers or multi-step bootstrapping; as a result, it collapses to an uninformative policy and fails to learn a viable pit-stop strategy.

11 Conclusion

This project implemented two reinforcement learning algorithms not covered in class, Double Deep Q-Network (DDQN) and Proximal Policy Optimisation (PPO), and evaluated them on two existing MDP domains: a discrete tabular MDP and a continuous Formula 1 racing environment. Additional baseline algorithms (DQN and one-step Actor–Critic) were included to contextualise performance differences and highlight the effect of design choices such as replay buffers, target networks, and clipped policy updates.

Across all experiments, DDQN demonstrated the strongest and most stable performance, achieving the highest win rate and the largest positive time advantage. PPO also performed well, learning a consistent one-stop strategy and converging faster than value-based methods. In contrast, the one-step Actor–Critic algorithm failed to learn a competitive strategy, underscoring the difficulty of applying simple on-policy methods to long-horizon, high-variance control tasks.

The results illustrate the importance of stabilising mechanisms such as target networks, replay buffers, and clipped objectives when learning in noisy environments. They also show that continuous-state racing simulations can serve as a challenging and informative benchmark for reinforcement learning, capturing complex strategic trade-offs such as tyre degradation, pit timing, and undercut dynamics.

Overall, the project demonstrates that modern RL algorithms can learn effective race strategies from scratch, and that algorithmic design choices have a profound impact on learning stability and final performance.

12 Contributions

This project was carried out collaboratively, with each team member contributing to both the implementation and the analysis. Work was roughly divided as follows:

- **Smitha Shashikumar Maganti**
  Designed and implemented the discrete 1440-state tabular F1 pit-stop MDP, including state discretisation, transition design, and reward shaping. Implemented the tabular RL baselines (SARSA with exploring starts, Q-Learning, and first-visit Monte Carlo), generated learning curves, and evaluated the resulting policies. Wrote and revised the report sections describing the discrete environment and classical RL methods.

- **Dhevdharsan Bhavani Satish Kumar**
  Contributed to the design, testing, and debugging of both environments, ensuring correctness of tyre dynamics, pit-stop transitions, and lap-time simulation. Implemented and tuned the PPO training pipeline in the continuous environment (rollout collection, GAE computation, logging, and evaluation). Assisted in hyperparameter tuning for all agents and jointly interpreted the comparative experimental results. Co-authored the experimental analysis and contributed to the integration of the final report.

- **Pooja Rajeev**
  Led the implementation of the continuous 8-dimensional Formula 1 environment, including tyre-wear modelling, lap-time calculation, opponent behaviour, and reward structure. Implemented the deep RL agents used in the continuous domain: DQN, Double DQN, and the one-step Actor–Critic. Ran large-scale experiments, produced the learning-curve and action-frequency plots, and drafted the algorithm descriptions and methodology sections of the report.

All three members participated in debugging, reviewing each other's code, interpreting intermediate results, and contributing to the final written document. The conclusions presented reflect collective discussion and agreement.