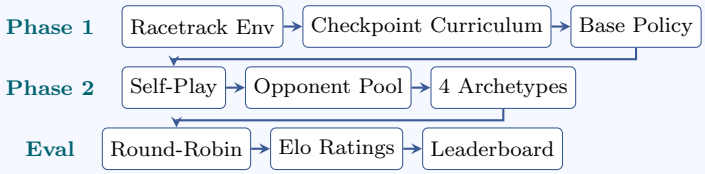
Competitive MARL Racing

Sebastian Cheung, Filip Ilic, Max Mayer, Raahil Gadhoke

1. Motivation & Problem Statement

When our group realized we were all great fans of Formula 1, we knew it would be a very interesting and engaging project topic. F1 racing is considered one of the most mentally intense sports in the world, hence why autonomous racing is challenging for reinforcement learning; agents must learn continuous control decisions while reacting to opponents in real time. Most RL approaches train a single agent against scripted or fixed opponents, which leads to policies that break down against adaptive racers (Bansal et al., 2018). Multi-agent self-play addresses this by letting agents co-evolve against each other, but it often results in strategy collapse where all agents converge to the same behavior. This is a known limitation, since without explicit incentives for diversity, self-play tends to produce a single dominant style rather than a variety of strategies actually seen in real motorsport. Our core question (beyond "can we train an agent to race around a track") is: can reward shaping alone produce meaningfully different competitive racing styles? In real F1, drivers, cars, and teams develop distinct identities and we wanted to see if reinforcement learning agents could develop similar specializations and learn how to use them to their advantage.

2. Approach



Our system is built on highway-env's `racetrack-v0` environment, which simulates a race track with continuous steering and acceleration controls. We implemented Proximal Policy Optimization (PPO) from scratch in PyTorch, where each agent uses an actor-critic network: the actor outputs a Gaussian distribution over steering and throttle, and the critic estimates state values for computing advantages via GAE.

Training happens in two phases. First, a single agent learns basic racing through a checkpoint curriculum: the track is divided into segments, and the agent must consistently reach each one before the next subsequent checkpoint unlocks. We augment observations with five lookahead waypoints from the track geometry, giving the network information about upcoming turns (similar to how F1 drivers know what to expect of the track before they start the race because they study the turns). In the second phase, we fine-tune copies of this base policy through self-play, where a learner races against opponents sampled from a pool of its own past checkpoints.

To produce diverse styles, we built a modular reward system where components (speed, lane centering, collision penalty, overtaking, blocking, drafting) are independently weighted. Four archetypes (Balanced, Speedster, Blocker, Drifter) share the same base driving rewards but add different bonuses on top. Lastly, we evaluate all archetypes in a round-robin tournament scored with Elo ratings.

3. Key Results

Our main finding is that reward shaping alone is enough to produce agents that behave differently while remaining competitively balanced. After training each archetype through self-play (around 100k timesteps each), we evaluated them over 10 episodes on lap completion, average lap time, and head to head Elo ratings.

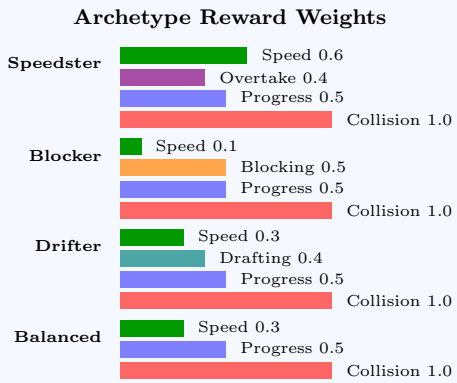
Lap Completion & Speed

Agent	Laps	Rate	Avg Steps	Best
Balanced	10/10	100%	645	559
Blocker	8/10	80%	514	456
Speedster	7/10	70%	518	451
Drifter	7/10	70%	532	486

Self-Play Training Summary

Archetype	Episodes	Updates	Mean Rew.
Blocker	443	48	281.69
Drifter	670	48	185.50
Speedster	934	48	134.00
Balanced	3,632	39	25.31

The results show a clear trade-off between reliability and speed. Balanced completed every lap but was the slowest (average 645 steps). Speedster had the fastest single lap (451 steps) but only finished 70% of the time due to aggressive driving causing crashes. Blocker was in-between, finishing 80% of laps at a moderate pace. In the round-robin Elo tournament (60 matches), all four archetypes finished at exactly 1000 Elo, meaning no specific style dominated, which supports our hypothesis that reward shaping produces diverse but competitively viable strategies. Importantly, reward magnitudes differ across archetypes by design (each weights different components), so cross-archetype reward comparison is not meaningful.



Each archetype shares the same base reward layer (progress, collision penalty) to maintain basic driving ability. The distinguishing components are shown to the left: Speedster prioritizes raw speed and overtaking, Blocker rewards holding position, Drifter incentivizes sitting in the slipstream (drafting), and Balanced adds only a mild speed bonus with no specialty component.

The training summary table on the previous page reveals how reward design affects learning dynamics. Speedster ran 934 episodes in 48 updates; its aggressive style causes frequent crashes, producing many short episodes. Balanced ran 3,632 episodes in only 39 updates, meaning episodes lasted much longer on average since the agent drives cautiously and rarely crashes. Blocker and Drifter fall in between. As said, reward magnitudes are not comparable across archetypes since each weights different components, but the episode counts and completion rates confirm that the archetypes developed genuinely different driving behaviors rather than minor variations of the same policy.

4. Challenges & Lessons Learned

Our biggest challenge was getting PPO to work correctly. Our initial implementation had a shared actor-critic network, which caused gradient interference and training instability. We also discovered that our GAE computation treated crashes and time-limits identically, and our reward normalization needed tuning to keep training stable. These bugs meant agents crashed within about 19 steps on average and clip fractions hit 0.79 (healthy is under 0.25). Fixing these required decoupling actor and critic into separate networks, normalizing advantages per-minibatch instead of across the whole buffer, and properly distinguishing terminated from truncated episodes in the bootstrap calculation.

The second major issue was that agents couldn't see ahead; observations only included the current state, not where the track was going. Adding waypoint observations (5 lookahead points with heading and lane width) and a curriculum that gradually unlocked track sections finally let agents learn to steer through corners. We also had to fix a bug where crossing the finish line was filtered as a "teleport" glitch, preventing lap completion from ever being rewarded.

5. Individual Contributions

We split the project evenly as a team. Rather than assigning fixed roles, we all worked together whenever we had time and contributed across all parts of the codebase without boundaries. Everyone touched environment setup, agent training, reward design, evaluation, and the final presentation and materials.

Member	Primary Contributions	Approx. %
Sebastian Cheung	All areas (environment, agents, rewards, eval, presentation)	25%
Filip Ilic	All areas (environment, agents, rewards, eval, presentation)	25%
Max Mayer	All areas (environment, agents, rewards, eval, presentation)	25%
Raahil Gadhoke	All areas (environment, agents, rewards, eval, presentation)	25%

6. Repository & References

GitHub Repository: `https://github.com/sebcheung/Competitive-MARL-Racing`
Project Tracker: We used shared documents instead of GitHub Projects. This gave us more space to write notes, document challenges, and organize tasks, and was easier to access on mobile.
Key References:

[1] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. "Proximal Policy Optimization Algorithms," *arXiv:1707.06347*, 2017. Core RL algorithm used for all agent training.

[2] Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., & Mordatch, I. "Emergent Complexity via Multi-Agent Competition," *ICLR*, 2018. Self-play framework and opponent pool design.

[3] Leurent, E. "An Environment for Autonomous Driving Decision-Making," *GitHub: highway-env*, 2018. Simulation environment our racetrack is built on.

[4] Elo, A. *The Rating of Chessplayers, Past and Present.* Arco, 1978. Rating system used for agent evaluation.