

**Reinforcement Learning Method Choice:  
AI Agent for Car Racing Game**

Version: 1.0  
Author: José Henrique Nóbrega Pereira  
Fontys ICT  
Date: June 6, 2025

|  |
| --- |
| A black background with purple text  AI-generated content may be incorrect. |

**Introduction**

The aim of this project is to train an agent capable of playing the Car Racing environment from the Gymnasium library. The first key decision that required critical investigation was the selection of an appropriate reinforcement learning method. Ultimately the method chosen was Proximal Policy Optimization (PPO), but this decision followed detailed comparison of other methos such as Q-Learning, Deep Q-Networks (DQN) and Dueling DQN. This document will outline the reasoning behind my choice.

**Method Requirements**

To choose an appropriate reinforcement learning method, the environment and its characteristics should be taken into account. The main factors considered are:

* The observation space and the action space are continuous.
* The game visual input is a 96x96 RGB frame
* The reward updates along with the running of each episode and is always changing

These factors make methods designed for discrete and low-dimensional environments challenging to implement and obtain good performance.

**Exploration of Different Methods**

**Q-Learning**

The first method considered was Q-Learning. This is one of the simplest and can be effective in small and discrete environments. This method became immediately apparent that it would not be a good fit for this project due to its discrete nature. To adapt the continuous environment to the discrete nature of Q-learning, the actions would need to be discretized which would result in losing details on the actions performed and the agent would not be able to do smooth inputs, which is desirable when driving a car. This results in unnatural and jerky driving and loss of fine control over the car.

**Deep Q-Network DQN**

Next method considered was Deep Q-Network, which improves Q-learning by adding deep neural networks. Although it has achieved success in similar pixel-based environments, the action space is still discrete, suffering from the same limitations as Q-learning when controlling the car.

**Dueling DQN (DDQN)**

Dueling DQN improves on DQN by separately estimating the state value and the advantage of each action. This separation leads to more efficient learning and better generalization. The actions are still discrete but the improvement in the architecture should make this method a sufficient choice. One other important consideration to have been the hardware limitations of each method. Since the training will be running on a laptop, the resources required should be on the lighter side, since there is limited time to train the agent and most likely it will require training multiple agents, if it take 2 weeks to train a single agent, the project will not be feasible due to time constraints. This eliminates DDQN since it is quite a resource extensive method.

**Deep Deterministic Policy Gradient (DDPG)**

The first method considered with continuous actions was Deep Deterministic Policy Gradient. This is an off-policy actor-critic algorithm that handles continuous action spaces directly making it better suited for this project. It makes use of two different networks: an actor network that outputs actions and a critic network that evaluates those actions. However, it is extremely sensitive to hyperparameters, making the training unstable if the parameters are incorrect. While on paper is looks like an attractive option, its sensitivity and need to extensive tuning made it not practical given my time and hardware constraints.

**Proximal Policy Optimization (PPO)**

PPO is also an actor-critic algorithm, but unlike DDPG, it is on-policy, meaning it learns from data collected using the current version of the policy. One of the defining features is the use of a clipped surrogate objective function, which constrains the extent to which the policy can change each episode. This ensures that the training remains stable and reliable, even in complex environments. Although PPO is less sample-efficient than off-policy counterparts such as DDPG, often requiring more interactions with the environment to achieve similar performance, it offers notable robustness to hyperparameter choices, which simplifies tuning. The continuous nature avoids challenges regarding discretization and places lower computational demands on hardware, making it feasible for the training hardware of this project.

**Method Choice**

Based on the methods explored, PPO seems to be the best option. It has a continuous action space by default, which is a must have when dealing with this type of environments dealing with controlling a car which requires fine control over the actions. The training robustness it provides is also a major factor PPO has over DDPG. The training might be longer due to lower sample-efficiency with PPO, but it is much more likely to provide results and not require many training restarts due to instability.

For these reasons PPO comes out as the best trade-off when considering multiple factors like stability, simplicity and performance. It allows the agent to learn smoothly, does not require extensive hyperparameter tuning and is suitable to be run on a laptop with reasonable training speed.