**Group ID:** 1  
**Students:**

* Abdulla Al-malki – 202009135
* Syed Subzwari – 202105006
* Abdollah Kandrani – 202104093

**Overview**  
Our project focuses on the Gymnasium CarRacing environment, which is built on Box2D—a 2D physics engine for games. In this environment, a random racing track is generated each episode, and the objective is to train an agent to drive through the entire track successfully. The project aims to apply both **tabular reinforcement learning** and **function approximation methods** to this environment and perform a comparative analysis between their performance.

The action space in CarRacing can be either **continuous** or **discrete**. In the continuous version, the action is a 3-dimensional vector: steering (ranging from -1 to 1), gas (0 to 1), and brake (0 to 1). The discrete version includes five predefined actions: 0 = do nothing, 1 = steer left, 2 = steer right, 3 = gas, and 4 = brake. The observation space is a 96x96 RGB image representing the top-down view of the track and the car.

The final deliverables will include a Jupyter Notebook containing the implementation, a written analysis comparing the two RL approaches, and a discussion with conclusions about their strengths, weaknesses, and applicability in continuous control environments.

**Literature Review**

**System Model and Problem Definition**

In this project, we aim to deploy tabular and functional approximation methods on the CarRacing-v3 environment and comparatively analyze their performance. In this environment, a random racing track is generated each episode, and the objective is to train an agent to drive through the entire track successfully. Gymnasium describes this as a control task, which means that we aim to learn policies that will try to maximize the total expected reward during episodes by staying on the track and reaching its destination quickly.

**Observation Space:**

The Observation Space is 96x96 RGB images of the car and track in a top-down view. It should be noted that the state isn’t explicitly given like in Toy Text environments where states are defined as numbers like s = 8. Rather, the images are given as the observed state, and the agent must be able to derive features from it to classify it as a state. While this may seem inconvenient, we need to understand that this is because Toy Text environments are small, so it is easy for us to get a state number. However, in Car Racing the observable state possibilities are so huge that storing it is infeasible. Also, because the number of states is so high, there is an extremely low chance of encountering the same state again even in later episodes. Due to this, we need to generalize the returned image after ‘taking a step’ in the environment regardless of if we are using tabular or function approximation. Actually, pure tabular methods would theoretically use all the states possible but this is infeasible, and even more so for us as we neither have the computational power, time or storage to achieve this.

For a rough idea, both tabular and functional approximation will need us to extract features. In tabular methods we need to be able to discretize the features as well, most probably by organizing them into bins and forming states from them. This is known as State – Aggregation. In function approximation, there is no need to do this as we just need to update our weights. The more encountered states throughout episodes will end up influencing the weights more, and hopefully that leads the car to perform optimally. We can look at the strategies in the slides for ways to construct our features.

**Action Space:**

For this environment, there are two types of action spaces: Continuous and Discrete. If the action space is defined to be continuous, an action is defined as a 3-dimensional vector [s, g, b] containing steering, gas and brake respectively. Steering ranges from -1 to 1, where -1 is fully left and +1 is fully right. Gas and brake range from 0 to 1. The discrete version includes five predefined actions: 0 = do nothing, 1 = steer left, 2 = steer right, 3 = gas, and 4 = brake. As per online research, we will have to create discrete actions on our own. We can do this by predefining the continuous actions as a discrete action, so we could map the continuous action [0, 1, 0] as the gas, for example.

**Reward Structure:**

The agent is punished by -0.1 every frame and is rewarded with +1000/N for every track tile visited. The idea of punishing for every frame aims to encourage the agent to finish as quickly as possible. +1000/N may seem confusing at first but the breakdown of this reward is clever. Initially through the episode, the reward gained is high as “N” is small. However, as we progress through the episode “N” increases, and the reward gained decreases. This is so that the car tries to adopt efficient pathways (take minimum number of steps) to attain maximum reward. The episode terminates if all the tiles are visited (meaning the lap is done perfectly) or if the car goes far off the track where it receives -100 punishment and dies.

**Methodology:**

In Chapter 9, we have touched upon three prediction methods: Gradient Monte Carlo, TD(0) and LSTD with mainly a focus on the first two. Within these methods, feature construction is important as both methods rely on features (derivative of the linear function approximator is equal to the features!).

TD(0) seems to be a better method for a couple of reasons. Firstly, we expect the episodes to be long which TD(0) works well with. This is because TD(0) allows the agent to learn online every step of the way meaning it is more sample efficient. Gradient MC on the other hand, would require us to wait for the whole episode to finish before updating weights. The hope is that TD(0) will converge to the optimal policy faster, which is highly desirable to us due to hardware limitations. From this, we are free to choose SARSA, Expected SARSA or Q-Learning for control in the approximation method.