International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Development and Evaluation of Autonomous Parking System Utilizing Reinforcement Learning Agents within Unity3D Environment

Kushagra[1], Abhinav Sajjan[2], Shreyashi Jaiswal[3], Himanshu Mishra[4], Samender Singh[5]

[1,2,3,4]4th Year Student, B.Tech in Computer Science & Engineering with AI & ML, Email:
[5]Associate Professor, Computer Science & Engineering Dept, Ajay Kumar Garg Engineering College, Ghaziabad, UP, INDIA

Abstract
This paper describes how RL agents in the Unity Environment can perform parking. The goal of the study is to propose a method that makes use of reinforcement learning techniques offered by the Unity ML-Agents framework within Unity's realistic 3D simulation in order to solve the requirement for autonomous parking solutions. The suggested solu- tion's design, execution, and assessment are highlighted in the paper. In complex situations, the system offers an adaptive and realistic frame- work for autonomous parking. The outcomes of thorough performance testing and comparative analysis highlight the usefulness and promise of the suggested approach in the area of autonomous car parking. The discussion of the results, difficulties faced, and prospects for additional study and advancement in autonomous car parking technology round up the report.

Keywords: Unity ML-Agents, Unity Game Engine, Autonomous Park- ing System, Reinforcement Learning

1 The Introduction

The everyday evolution of Artificial Intelligence and Virtual Simulations is leading the way to new technological advancements. Within this paradigm, the task of au- tonomous parking requires great precision and adaptability for intricate scenarios. The problem can be addressed by the approaches of Machine Learning (ML) and Reinforcement Learning (RL). This paper focuses on utilising the capabilities of RL provided by the Unity ML-Agents framework, within the Unity3D simulation envi- ronment to solve the problem of autonomous vehicle parking.

The question arises, what is Reinforcement learning? Let's take the example of Volleyball (Fig 1). Initially, the agents do not have any information on how to play the game. They'll start by taking random actions and through trial and error, they'll learn that: [Zha]

- If they hit the ball and it goes over the net to the other side of the court, they score points (positive feedback),
- If they let the ball hit the floor on their side of the court, they lose a point (negative feedback).

Doing the things that lead to positive outcomes will teach the agents to hit the ball over the net whenever it's on their side of the court. Technically, Reinforcement learning is a subdomain of machine learning which involves training an 'agent' (here the volleyball player) to learn the correct sequences of actions to take (hitting the ball over the net) in a given state of its environment (the volleyball game) to maximize its reward (scoring points) [Zha][SB18]. The RL training process includes 2 key steps/phases: Exploration and Exploitation. The developer's training algorithm will decide when the agent should explore the environment and when to exploit the gained information. We will take a deeper look into this in further sections

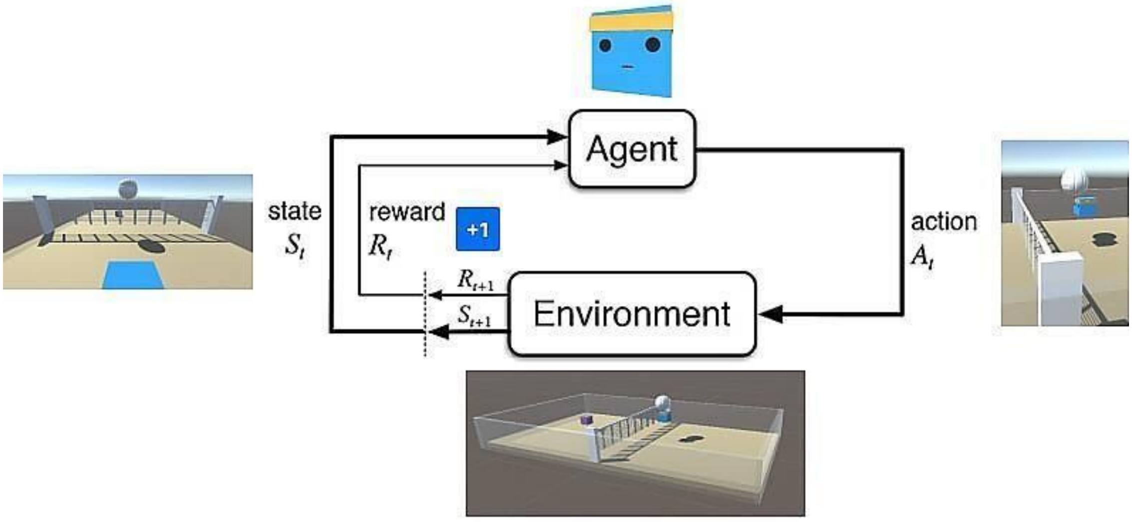

Figure 1: Reinforcement Learning

The robust physics engine and realistic rendering provided by the Unity are ideal for creating simulated environments, they closely mimic the real-world challenges which further strengthens the training for our RL agent. As autonomous vehi- cle technology progresses, the development of intelligent parking systems becomes crucial. These types of systems can aid in transforming current automobiles into partially autonomous vehicles as they can be provided as third-party modules. This proposal leverages Unity's capabilities to create a simulated environment where a virtual car equipped with an RL agent can learn to navigate diverse parking scenar- ios, hence addressing the pressing need for an automatic parking system. This paper is structured as follows: [JBT+20]

- We begin with an introduction of the problem at hand, its significance, and the scope of this paper,
- Then, we describe the current and previous works on the problem and also propose a series of solution(s), including our primary RL agent using Unity ML-Agent in the Unity3D Environment,
- We then describe the Unity engine and Unity ML-Agents Toolkit, a general platform and discuss its ability to enable research and how we can achieve the proposed solution using them,
- We next outline the architecture, functionality and tools provided by the Unity ML-Agents Toolkit which enables the deployment of RL Agent within Unity environments on example Parking Scenarios,
- Then, we assess the outcome and conclude by proposing future avenues of our findings.

2 The Problem

With the increase in the popularity of self-driving cars, the world's roads are pro- jected to be dominated by such cars in the next decade. But it also introduces various challenges, including but not limited to, Lane detection, Lane following, Signal detection, and obstacle detection and avoidance. For this paper, we will fo- cus on the task of parking. Car parking is a complex task which requires the driver to handle:

- spatial awareness,
- trajectory planning,
- real-time decision-making, and more (Fig 2).

These challenges combined are so significant that a traditional algorithm can not overcome them. This paper identifies this problem and aims to train an RL agent to navigate and park a car autonomously, recognizing the dynamic and complex nature of the parking situation. The main reason to opt for the RL technique over other sorts of algorithms, and the Unity Environment is due to the facilities provided, we will take a closer look at the Unity Engine, and the Unity ML-Agents framework (including RL tools provided) in further sections.

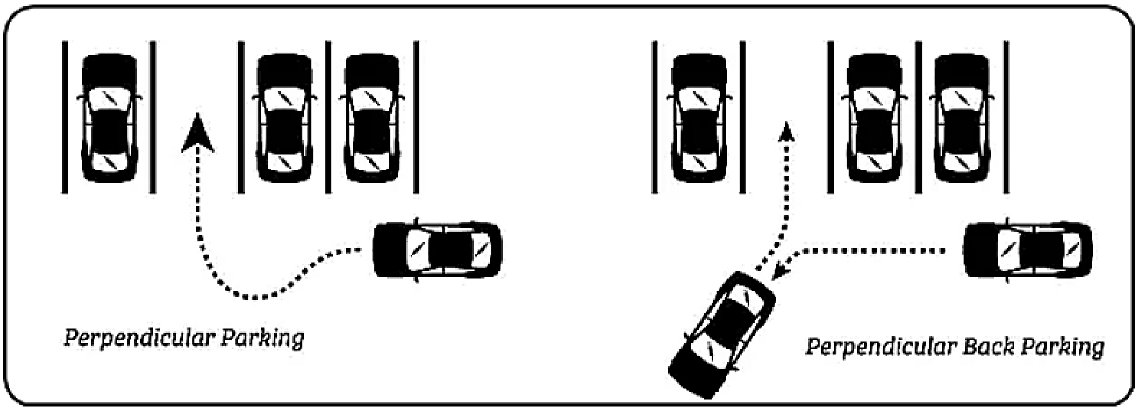

Figure 2: Perpendicular Parking Scenarios

The proposal seeks to address fundamental questions such as:

- How can we construct a system capable of parking a vehicle on its own?
- What factors contribute to successful navigation in various parking scenarios?
- and How efficient will the system be?

By honing in on these challenges, the paper aims to contribute to the development of robust and adaptive RL models capable of handling the problems associated with automated car parking. Moreover, it will inform about the challenges of the field and whether the ML and RL approach is feasible and effective or not.

2.1 Significance

The paper is directly aimed at the advancement of autonomous vehicles and how to execute the car parking task autonomously, showing its significance in the field. Automated parking systems are an integral part of the broader self-driving technol- ogy, and demand intelligent agents capable of making swift and accurate decisions in real time.

By employing RL techniques, this project endeavours to create a model that not only learns optimal parking strategies but also adapts to diverse scenarios, showcas- ing the adaptability necessary for real-world applications. Moreover, the RL model will be evaluated in both training and testing phases which further outlines its effi- ciency and relevancy as a proposed solution. The paper also includes the prospect of the solution proposed which provides a peek into the autonomous parking future, it will also lay out the disadvantages of our approach which will further aid any research in the field. The paper, as stated earlier, makes use of Unity3D Engine and Unity ML-Agents framework. This shows the effectiveness of the software and the framework in the field of machine learning-oriented research.

The above-mentioned points promoted the need and proved the significance of this paper, also to conduct research and propose new more effective and feasible solutions by other fellow researchers.

Objectives

Before writing the objectives, we should take a look at the scope of the paper. This will increase our understanding of the objectives and what to expect.

The scope of this paper encompasses the development of an autonomous car parking system using Reinforcement Learning (RL) within the Unity simulation environ- ment. The primary focus is on training a virtual agent to autonomously navigate and park a car in diverse scenarios, emulating real-world challenges. The system will address various aspects of automated parking, including spatial awareness, tra- jectory planning, and real-time decision-making. We will also evaluate the resulting model in the training and testing phase, and conclude with the results and future uses or alterations. We will also briefly discuss the impact of this approach on the field and what areas should future researchers pay utmost attention to.

Now, various objectives of this paper include:

- Develop an RL model tailored for car parking in Unity, integrating state-of- the-art algorithms to enable effective learning and decision-making.
- Design and implement a simulation environment within Unity that encom- passes a range of parking scenarios, capturing the complexities of real-world parking challenges.
- Train the RL agent to navigate and park a virtual car autonomously, em- phasizing adaptability to different parking space configurations and dynamic environments.
- Evaluate the performance of the trained RL agent based on key metrics, in- cluding success rate, parking accuracy, and computational efficiency.
- Contribute insights to the broader field of ML applications in simulated en- vironments, offering solutions and methodologies for training RL agents in complex tasks, specifically in the context of automated car parking.

3 The Related Work

Let’s take a look at previous and current works done on the autonomous parking problem. Most of the work is done utilising single and multi-agent Reinforcement Learning, and Machine Learning techniques. However, few researchers have also implemented Unity3D and Unity ML-Agents framework for the task. Some of the previous works are:

Clara Barbu and Stefan Alexandru Mocanu: *On the development of Autonomous Agents using Deep Reinforcement Learning.* [BM21] : The pa- per presents a general study of autonomous agents with their development powered by deep reinforcement learning. This is combined with autonomous vehicles via an example of a vehicle agent parking autonomously in the virtual parking environment provided by the Unity3D Engine. The agent is utilizing Deep Q-Learning, Double Deep Q-Learning, and Experience Replay.

Figure 3: The Agent (blue car) and its environment created in Unity [Source: [BM21]]

The paper resulted in a model (Fig 3.1) able to park a car using Deep Q- Learning techniques, but the model took more than 72 hours to train. The results for a more general application (Ball-Cube) were more promising and quick utilising the Double Deep Q-Learning.

Mohamed Fethi Dellali and Mohamed El Mahdi Bouzegzeg: *Au- tonomous Parking Simulation using Unity Game Engine and Reinforcement Learning.* [DB22] : The report implemented an autonomous parking simu- lation using Unity3D Game Engine, Unity ML-Agents framework, and Rein- forcement Learning. They started with a discussion of various artificial intel-
ligence (AI) subsets and their methods, followed by a detailed discussion of reinforcement learning, Unity game engine, and ML-Agents.

The report resulted in a model which can seek out the empty parking lot in a parking area and execute the parking task correctly. The model trained for over 12 million steps in 12 hours with a theoretical success rate of 97% during the training phase.

Omar Tanner: *Multi-Agent Car Parking using Reinforcement Learning.* [Tan22] : The paper aimed to train a model able to perform in a multi-agent system (Fig 3.2) where other cars can also communicate and aid in parking tasks.

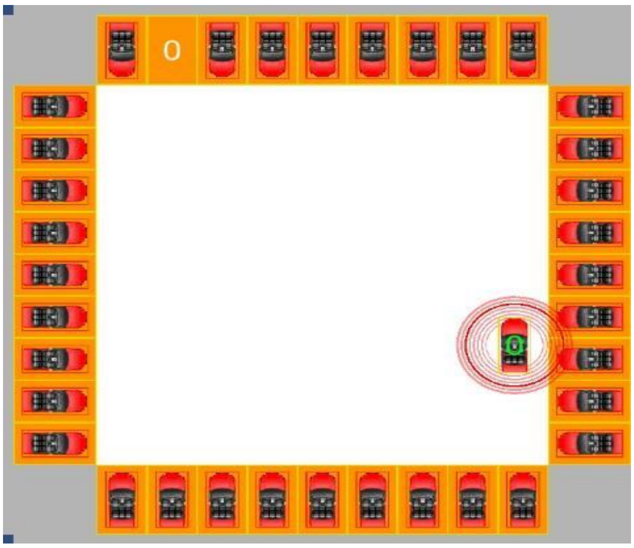

Figure 4: Multi-Agent System for Autonomous Parking [Source: [Tan22]]

The fixed goals and obstacles environment yielded a model using up to 7 agents with a success rate of 98.1%. If the model only implemented 2 agents instead of 7, the rate bumped up to 99.3%. This proved the effectiveness of the multi-agent system in the autonomous parking scenario.

- **Yusef Savid, Reza Mahmoudi, Rytis Maskeliu¯nas, and Robertas Damaˇseviˇcius**: *Simulated Autonomous Driving Using Reinforcement Learn- ing: A Comparative Study on Unity's ML-Agents Framework.* [SMMD23] : The paper compares the performance of several different RL algorithms and configurations on the task of training kart agents to successfully traverse a racing track (Fig 3.3) and identifies the most effective approach for training kart agents to navigate a racing track and avoid obstacles in that track.

Figure 5: The Race Track environment in Unity [Source: [SMMD23]]

The paper also explored the effectiveness of behavioural cloning; a technique of copying human skills or inputs and training the model to closely mimic them, in the area of racing simulators. The results when compared to the Proximal Policy Optimization algorithm, noticed only a deviation of 23.07% in value loss and only a 10.64% deviation in cumulative reward, hence confirming the usefulness of behavioural cloning for improving the performance of intelligent agents for racing tasks.

Now let's propose our solution for Autonomous Parking:

We tackle the situation using Reinforcement Learning, specifically using the Prox- imal Policy Optimization (PPO) algorithm. We will utilise Unity ML-Agents to implement this RL model using the PPO. The model will be trained, tested, and evaluated in the Unity Engine. The agent will be using the "Ray Perception Sensors" component provided by Unity for sensing the environment. They behave as real-life Lidar sensors. The agent will be able to access a CarController script which will provide the actions the agent can perform (drive, steer, brake). The agent will be placed in a dynamic simulation environment which will constantly change with each episode to promote the adaptability of the RL model. The reward system for the agent will promote "reverse parking" over the traditional front parking to induce good parking etiquette. Additionally, it will penalise the agent on a collision to promote task completion. The agent will be evaluated in 2 ways: parking success and model parameters. In the former one, we evaluate the "Efficiency Percentage" or the total parking agent did in a number of cases over a limited period of time during both training and testing. In the latter, we refer to Tensorboard for model parameters such as extrinsic reward, episode length, policy loss, etc. during the training phase. The final results are a combination of both and the final statement will be derived from both findings.

4 The Tools & Technologies Used

Let's look into brief details of specific tools (Unity Engine and Unity ML-Agents) and software technologies or methods (Reinforcement Learning, Artificial Neural Networks, and Proximal Policy Optimization) we have utilised for our work.

4.1 Unity Engine

Unity is a cross-platform game engine developed by Unity Technologies, first an- nounced and released in June 2005 at Apple Worldwide Developers Conference as a Mac OS X game engine [Wik]. Over the years, it has grown into a cross-platform powerhouse, supporting development for a multitude of devices and platforms, from desktop and mobile to consoles and virtual reality.

4.1.1 Key Features

1. **Physics Simulation:** The engine includes a built-in physics engine that en- ables realistic simulation of object interactions, collisions, and dynamics.
2. **Scripting and Programming:** Unity3D supports scripting and program- ming in C#. Developers can write custom scripts to define game logic, be- haviour, and interactions.
3. **Asset Pipeline:** Unity3D features a streamlined asset pipeline that facilitates the import, management, and manipulation of various asset types, including 3D models, textures, audio files, animations, and shaders.
4. **Cross-Platform Development:** One of Unity3D's defining features is its cross-platform development capabilities. Developers can write code once and deploy their games to a wide range of platforms.

4.2 Unity ML-Agents

The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source project that enables games and simulations to serve as environments for training 5intelligent agents [Tec]

4.2.1 Key Features

1. It supports various training situations and environment configurations.
2. Several Deep Reinforcement Learning algorithms (PPO, SAC, MA-POCA, self-play) are supported for training single-agent, multi-agent cooperative, and multi-agent competitive situations.
3. Assistance for using two imitation learning algorithms (GAIL and BC) to learn from demonstrations.

It supports training with several instances of the Unity environment running at once. This speeds up the process without compromising the adaptability.

4.3 Reinforcement Learning

Reinforcement Learning can be defined as a technique for problem-solving where an intelligent agent is trained using experiences. The agent will be put in the problem environment at a particular state or situation, where it can perform certain actions that will generate rewards or penalties and transfer it into a new state. A state can be defined as a particular scenario in a problem and used by the agent to perform actions and get to a solution. The reward is a positive incentive the agent receives when it comes close to the desired output whereas the penalty is a negative reward which is given when the agent either deviates from the solution or makes a blunder. Penalties are significantly higher than the rewards to make sure the agent never repeats the negative actions.

The reinforcement learning process generally results in a model capable of per- forming the task it was trained for with great efficiency. The model is typically represented with an artificial neural network, a multilayer feed-forward neural net- work in our case, which has node functions and weights calculated according to its learned behaviour from the training phase.

4.4 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) are computational models inspired by the struc- ture and functioning of biological neural networks in the human brain. ANNs consist of interconnected nodes, or neurons, organized in layers, allowing them to learn com- plex patterns and relationships from data.

Feedforward Neural Networks: Feedforward Neural Networks (FNNs): FNNs are the simplest type of neural networks, where information flows in one direction from the input layer to the output layer without feedback loops.

4.5 Proximal Policy Optimization

Based on policy gradient approaches, proximal policy optimization (PPO) seeks to maximize predicted cumulative rewards by repeatedly improving an agent's policy. Fundamentally, PPO makes use of a surrogate objective function to direct policy updates while guaranteeing effective and consistent learning dynamics.

5 The Architecture & Working

5.1 Working

The solution can work in 2 modes inside the Unity Editor: Training and Testing.

Training: During Training mode, the agent operates without any Neural Net- work guidance initiating the training process. To commence the training we execute the following command in the appropriate environment in the anaconda command prompt.mlagents-learn --run-id="modelNameOrId"

The run-id defines the model name and is used by the tensorboard for model stats. The behaviour during this process is defined by the trainer config.yaml, which dictates actions such as periodic Neural Network exports and checkpoint creation. These actions are crucial for preserving progress.

Testing: During Testing mode within the Unity Editor, a Neural Network is essen- tial. This passed Neural Network, or model is subsequently put to the test in various parking scenarios. The performance assessment occurs agent by agent, facilitated by EfficiencyCal.cs, and collectively for all agents, managed by EfficiencyComb.cs. This process persists endlessly and can only be halted by selecting "Stop" in the Unity Editor.

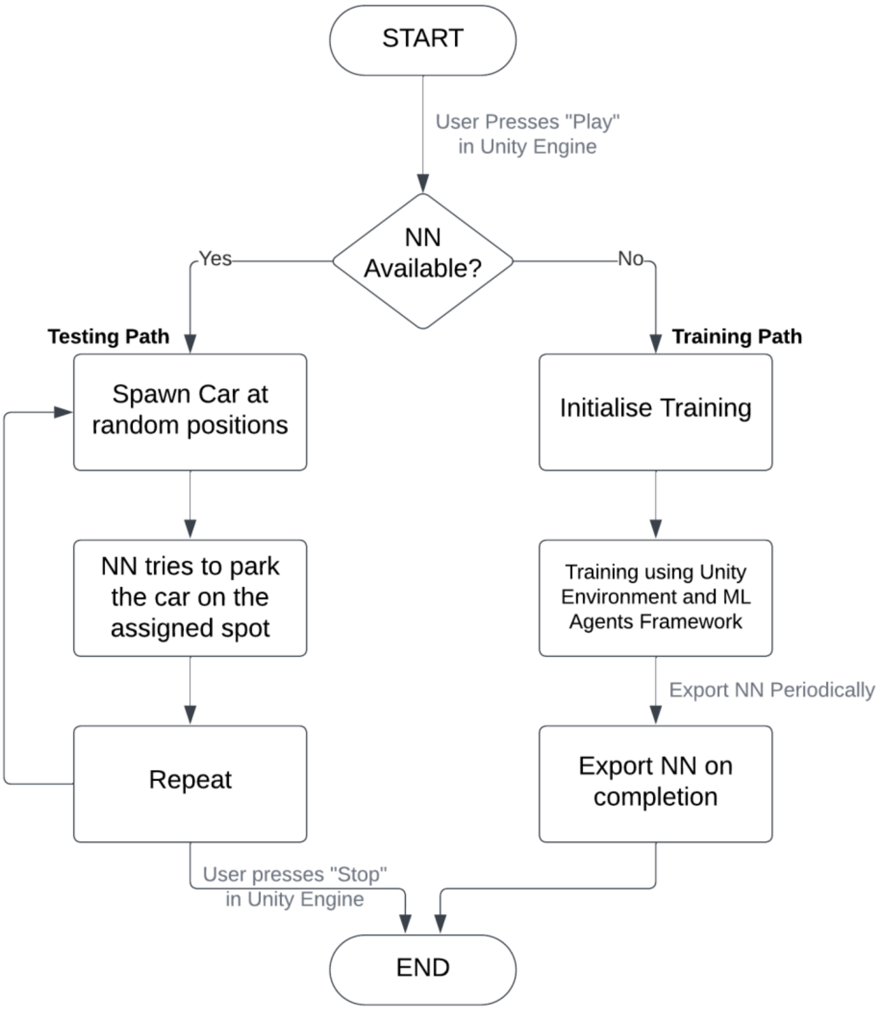

Figure 6: Flow chart showing the working in Unity Editor

In the exported application, the solution works only in testing mode. Testing begins when the user selects "Start" and chooses a model from the list provided. We've included a total of 7 models, comprising 4 development models and 3 export models, all saved on the drive. Following this selection, the "Final Scene" is loaded, functioning akin to "Unity Editor: Testing Mode". Moreover, users can opt to "Reset" the scene, "Go Back" to the main menu to try another model, or "Quit" the application.

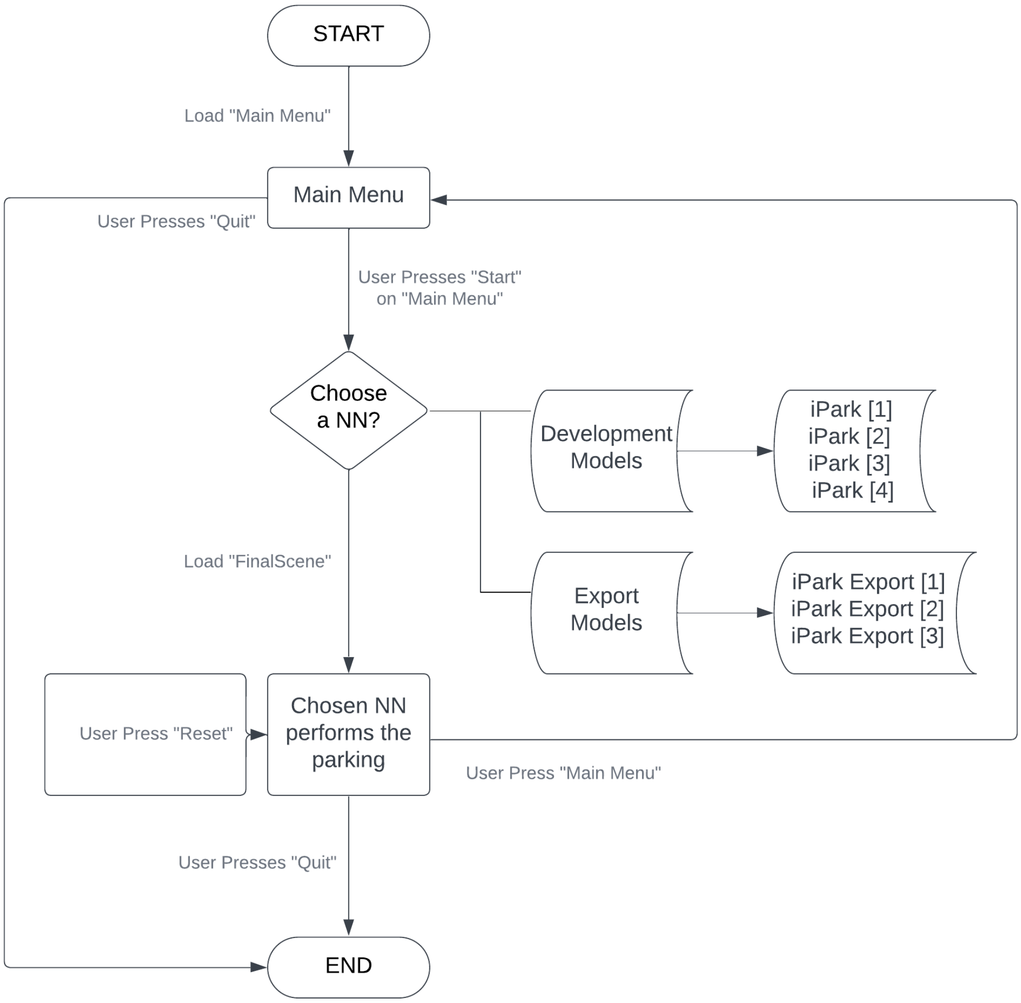

Figure 7: Flow chart showing the working in exported Application

6 The Results

We created a total of 7 models for this purpose and compared training and testing results of each. The training results mainly consisted of model or policy parameters such as the "Cumulative Reward", "Episode Length", "Policy Entropy" etc. Addi- tionally, we calculated the parking efficiency during training. The testing results is the parking efficiency of the model over an evaluation period of 4hrs. Let's define the training specific terms first then take a look at the training and testing results for our most effective model (iPark Export [02]). At the end, we will discuss the distribution and how the reader can use the application himself.

6.1 Training Results Related Terminology

1. **Cumulative Reward:** Cumulative Reward in TensorBoard graphs tracks the total reward obtained by the agent during training or evaluation. It offers a quick overview of the agent's overall performance and its ability to achieve goals within the environment.
2. **Episode Length:** Episode Length in TensorBoard graphs represents the du- ration of each episode during training. It indicates how long the agent interacts with the environment before reaching a terminal state or completing a task. Tracking episode length helps monitor the efficiency and effectiveness of the agent's decision-making process over time.
3. **Policy Loss:** Policy Loss in TensorBoard graphs reflects the discrepancy be- tween the predicted actions of the agent and the optimal actions determined by the policy during training. It measures how

well the agent's policy approx- imates the desired behaviour and provides insights into the training progress and stability of the reinforcement learning algorithm.

4. **Value Loss:** In TensorBoard, the Value Loss metric tracks the error between predicted and observed returns during training. A good performance shows a decreasing trend over time, indicating improved accuracy in predicting future rewards. Fluctuations may occur, but overall, the curve should converge to a low and stable level, signalling successful learning by the agent.
5. **Policy Entropy:** Policy Entropy in TensorBoard measures the uncertainty or randomness of the agent's action selection. A good agent should maintain a moderate level of entropy to encourage exploration and prevent premature convergence to suboptimal policies. An ideal scenario shows a decreasing trend in entropy as the agent learns to make more confident and informed decisions over time, but without diminishing too quickly, ensuring a balance between exploration and exploitation.

6.2 iPark Export [02] Training Results

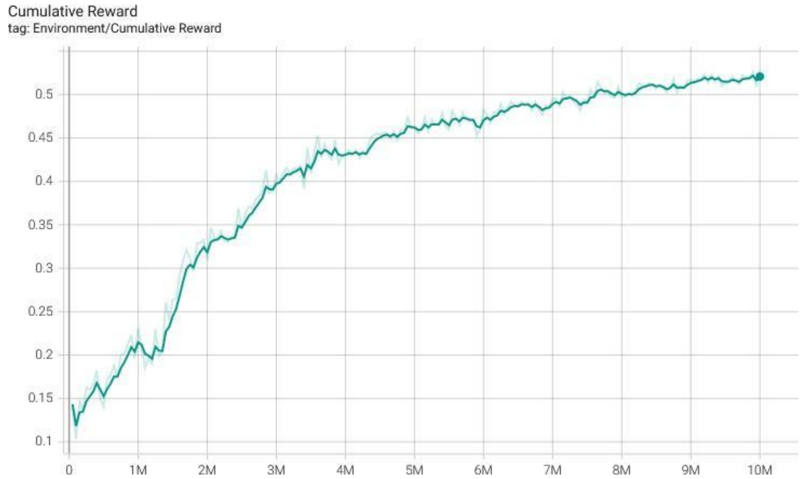

Figure 8: Cumulative Reward for iPark Export [02]

The cumulative reward graph starts at 50k steps with a reward value of 0.1434, and ends at 10M steps with a value of 0.5269 in 3 hours 28 mins. This shows clearly that the model is learning to park. The graph is increasing steadily throughout the period. This shows that the model was learning new behaviours and did not mature early.

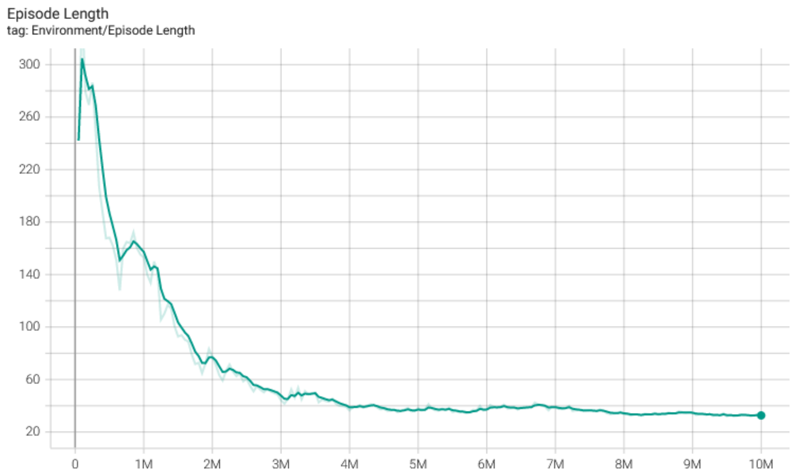

Figure 9: Episode Length for iPark Export [02]

The episode length graph starts at 50k steps with an episode length of 242 and ends at 10M steps with a length value of 32 (32.37). This shows that the model was learning new optimal behaviours and was able to park with fewer steps in each episode. Moreover, the graph was almost flat from 4M steps (36.35) which shows that the model was able to find optimal settings very early.

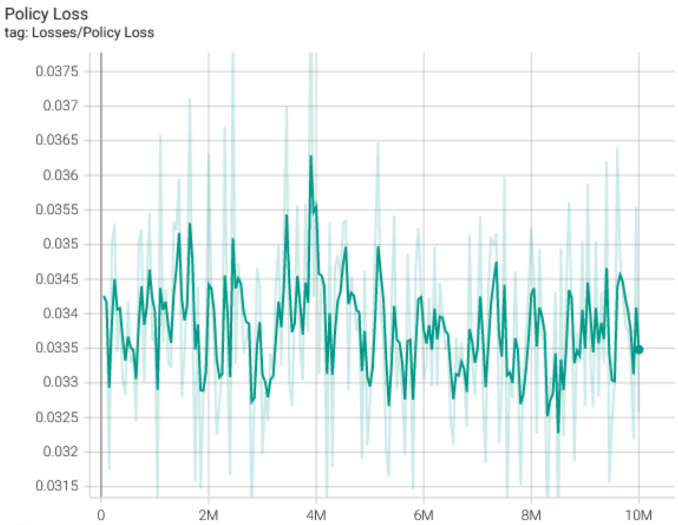

Figure 10: Policy Loss for iPark Export [02]

The policy loss graph starts at 50k steps with a value of 0.03426 and ends at 10M steps with a value of 0.03257. This decrease in value show that the model was able to find a optimal policy function. Moreover, the graph is constantly declining, meaning that the model was improving throughout the training period. The fluctu- ations show that the agent is learning from the environment.

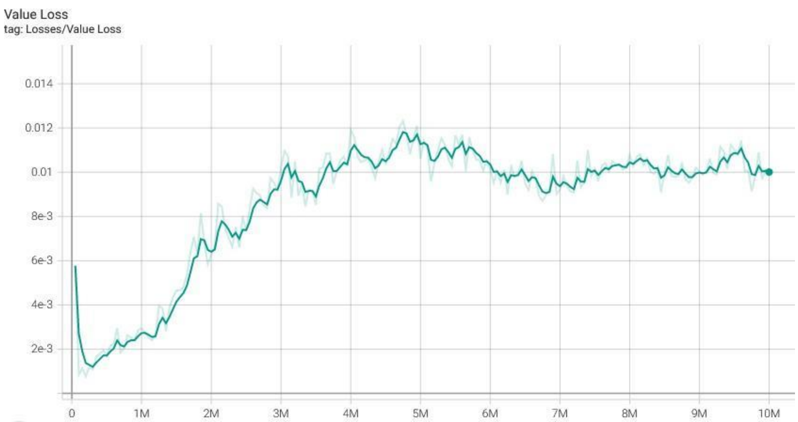

Figure 11: Value Loss for iPark Export [02]

The value loss graph starts at 50k steps with a value of 5.768e-3 (0.005768) and ends at 10M steps with a value of 9.9214e-3 (0.0099214). Value loss shows the differ- ence between the predicted value of state-action pairs by the agent’s value function and the actual observed returns received during training. An ideal behaviour will be a decreasing or constant graph. The graph initially increased till 4.75M steps, but then it declined and stayed continuous from 7M steps. Morover, the overall increase was very low (0.0041534) which shows the model really performed well in value loss and find an optimal solution.

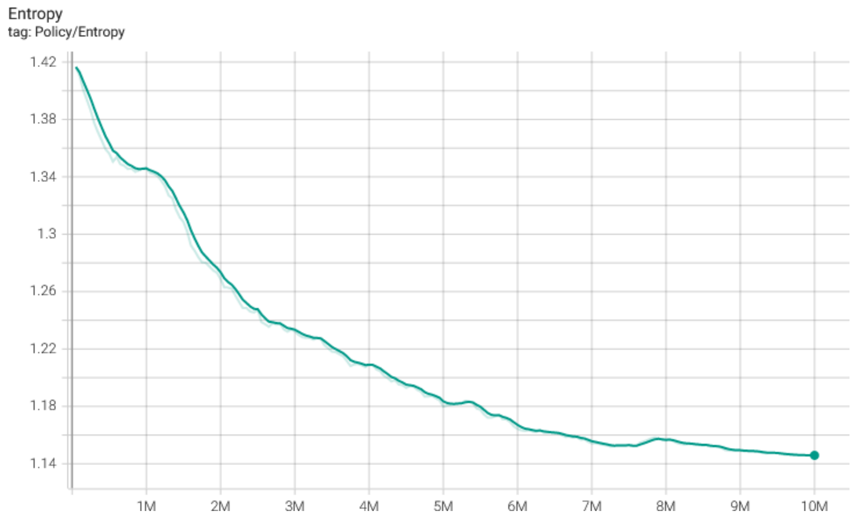

Figure 12: Policy Entropy for iPark Export [02]

The policy entropy graph starts at 50k steps with a value of 1.417 and ends at 10M steps with a value of 1.146. The decreasing trend is favourable here. It shows that the agent was able to balance between exploration and exploitation without converging to a sub-optimal solution.

Training Parking Efficiency

training efficiency reported in the " Efficiency . txt" by the performance metric component
08 -05 -2024 15:30:55 (Training Model)
Efficiency 79 .27053 %
Total Park 170392
Total Collision 44558
Total Cases 214950

The EfficiencyCal and EfficiencyCombined scripts reported this efficiency of the model. This is for whole of the training period and will differ from real testing.

6.2.1 Graph Analysis

The cumulative graph demonstrates significant progress, with the final value ex- ceeding the average (0.5269). Notably, the training period was the shortest among all, showcasing efficient learning. However, relying solely on this metric may not accurately predict the model's performance in test scenarios. Similarly, the episode length graph displays an ideal trend, with the final value among the lowest, indicating the successful optimization of episode length by the model. The policy loss graph also exhibits a consistent decline, reflecting effective policy improvement throughout training. However, the value loss presents a challenge, with a continuous rise for most of the training period, though the model managed to stabilize it towards the end. Despite these challenges, the policy entropy remains ideal, indicating proper functionality of the Proximal Policy Optimization (PPO) algorithm. Although the efficiency data during training is promising, it's essential to note that it may not nec- essarily correlate with testing results. Further insights into the model's performance will be gained during the testing phase.

6.3 iPark Export [02] Testing Results

training efficiency reported in the
" Efficiency . txt" by the performance metric component

16 -05 -2024 14 :34 :04 (Testing Model " iPark Export [02]
-10000076 (Unity. Barracuda . NNModel)")
Efficiency 89 .37852 %
Total Park 31539
Total Collision 3748
Total Cases 35287
The 4-hour evaluation test resulted in the agent attempting a total of 35,287 parking scenarios. It parked a total of 31,539 times and collided 3,748 times. This gives us an 89.37852% efficiency which is 10.10799% more than training data.

6.3.1 Data Analysis

The 4-hour evaluation test resulted in the agent attempting a total of 35,287 parking scenarios. It successfully parked 31,539 times and collided 3,748 times, resulting in an efficiency of 89.37852%. This efficiency is 10.10799% higher than the training data, representing the highest improvement recorded. The model also performed exceptionally well in every training parameter, establishing itself as the best-suited model for the task.

6.3.2 All model comparison

Training Efficiency		**Testing Efficiency**	
Model Name	**Efficiency %**	**Model Name**	**Efficiency %**
iPark [01] 21-11-2023	–	iPark [01] 21-11-2023	78.56705%
iPark [02] 28-03-2024	74.95232%	iPark [02] 28-03-2024	84.407%
iPark [03] 29-03-2024	84.70142%	iPark [03] 29-03-2024	86.45386%
iPark [04] 30-03-2024	79.63393%	iPark [04] 30-03-2024	88.92231%
iPark Export [01] 04-05-2024	75.3954%	iPark Export [01] 04-05-2024	85.36852%
iPark Export [02] 06-05-2024	79.27053%	iPark Export [02] 06-05-2024	89.37852%
iPark Export [03] 08-05-2024	79.2570%	iPark Export [03] 08-05-2024	88.61481%

Table 1: Comparing models on their respective training and testing data

6.4 Deployment

The deployed application setup can be found here:
iParkSetup.exe [https://github.com/KushagraYashu/iPark/releases/download/setup /iParkSetup.exe]
After downloading, the program can be installed by running and following the in- structions in the setup.

7 The Conclusion

In conclusion, this research successfully demonstrates the viability and effectiveness of employing Reinforcement Learning (RL) agents within the Unity3D environment to tackle the complex problem of autonomous parking. By integrating the Unity ML-Agents framework, we have shown that virtual agents can be trained to nav- igate and park vehicles autonomously in a variety of scenarios that closely mimic real-world conditions. The RL approach, particularly within the robust and versa- tile Unity simulation, proved to be a powerful method for developing adaptive and intelligent parking systems.

Throughout this study, the RL agents displayed significant capabilities in spatial awareness, trajectory planning, and real-time decision-making. The performance metrics, including success rate and parking accuracy, indicated that the RL agents could consistently and efficiently execute parking maneuvers across different config- urations and dynamic environments. These results underscore the potential of RL techniques in advancing autonomous vehicle technologies, particularly in enhancing the functionality and reliability of self-parking systems.

Moreover, this research contributes valuable insights into the broader application of machine learning in simulated environments, offering a blueprint for future stud- ies aiming to train RL agents for complex tasks. The use of Unity3D as a simulation platform not only provided a realistic training ground but also facilitated the explo- ration of various parking scenarios, thereby enhancing the agent's learning process.

Looking ahead, there are several avenues for further research. Future work could ex- plore the integration of additional sensors and real-world data to improve the realism and robustness of the simulation. Additionally, investigating other RL algorithms and hybrid approaches could further enhance the efficiency and effectiveness of the autonomous parking system. The insights gained from this study pave the way for more sophisticated and adaptable autonomous systems, contributing to the ongoing evolution of intelligent transportation solutions.

In summary, the development and evaluation of the autonomous parking system utilizing RL agents within the Unity3D environment demonstrate a promising step forward in the realm of autonomous vehicle technology. The findings from this research highlight the practical applications of RL and set the stage for future in- novations in automated parking and beyond.

References

1. [BM21] Clara Barbu and Stefan Alexandru Mocanu. On the development of Autonomous Agents using Deep Reinforcement Learning. *Scientific Bulletin, University Politehnica of Bucharest*, Vol 83(Series C):97–115, 2021.
2. [DB22] Mohamed Fethi Dellali and Mohamed El Mahdi Bouzegzeg. Autonomous Parking Simulation using Unity Game Engine and Reinforcement Learning. *Univ Blida*, 2022.
3. [JBT+20] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A General Platform for Intelligent Agents. *arXiv:1809.02627v2*, 2020.
4. [SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second Edition. MIT Press, 2018.
5. [SMMD23] Yusef Savid, Reza Mahmoudi, Rytis Maskeliu¯nas, and Robertas Damaˇseviˇcius. Simulated Autonomous Driving using Reinforcement Learning: A Comparative Study on Unity's ML-Agents Framework. *In- formation*, 14(5):290, 2023.
6. [Tan22] Omar Tanner. Multi-Agent Car Parking using Reinforcement Learning.
7. *Book of Abstracts, ICUR 2022, ArXiv 2206.13338*, 2022. [Tec] Unity Technologies. Unity ML-Agents Toolkit.
8. [Wik] Wikipedia. Unity (game engine).
9. [Zha] Joy Zhang. A hands-on introduction to deep reinforcement learning using Unity ML-Agents.