A Project Report

on

Reinforcement Learning using Unity

*carried out as part of the **Artificial Intelligence Lab DS3230** Submitted*

by

Kushagra Yadav
209309060

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Data Science and Engineering

MANIPAL UNIVERSITY
JAIPUR

School of Computing and Information Technology
Department of Data Science and Engineering

MANIPAL UNIVERSITY JAIPUR
JAIPUR-303007
RAJASTHAN, INDIA

April 2023

CERTIFICATE

Date: 21/04/2023

This is to certify that the minor project titled **Reinforcement Learning using Unity** is a record of the bonafide work done by Kushagra Yadav (209309060) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Data Science of Manipal University Jaipur, during the academic year 2022-23.

Dr. Avani Sharma

Project Guide, Department of Information Technology
Assistant Professor, Department of Information Technology
Manipal University Jaipur

Dr. Akhilesh Kumar Sharma

HoD, Department of Data Science and Engineering
Manipal University Jaipur

ABSTRACT

The project involves developing a 2D snake game using reinforcement learning and ray tracing. The objective of the project is to train an agent to play the game using a reinforcement learning algorithm that incorporates ray tracing, which will enable it to learn and improve its performance through trial and error while also allowing for more accurate collision detection.

The project involves conducting a literature review to gain an understanding of the state-of-the-art in 2D game development, reinforcement learning, and ray tracing, designing the system architecture, collecting data, implementing the game and the reinforcement learning algorithm with ray tracing, training the algorithm, evaluating its performance, and deploying the final version of the game and the algorithm.

The project uses Unity as the game engine and Python API(provided by ML-Agents) for controlling the agent simulation loop of the game.

TABLE OF CONTENTS

1. Introduction

1.1 Introduction

Snake game is a classic game where the player controls a snake that moves around a game board, collecting food while avoiding obstacles and its own body. In recent years, reinforcement learning has gained popularity as a technique for training intelligent agents to play games. In this project, we used Unity and reinforcement learning to create a basic 2D snake game where an AI agent will learn to play the game.

Our goal is to create a simple, yet challenging snake game where the player can train an AI agent to learn the game rules and make strategic decisions on its own. We used Unity's powerful game engine and reinforcement learning techniques to train the agent. The agent will receive feedback on its actions in the form of rewards and penalties, which it will use to improve its decision-making capabilities.

1.2 Problem Statement

To create a basic 2D snake game in Unity using reinforcement learning, where an AI agent will learn to play the game on its own. The game should include basic game mechanics, such as movement of the snake, food spawning, and collision detection. The AI agent should learn to make strategic decisions by receiving feedback in the form of rewards and penalties, and use this information to improve its decision-making capabilities.

1.3 Objectives

- Develop a 2D game in Unity
- Implement reinforcement learning
- Design a reward system
- Train the AI agent
- Evaluate the AI agent

2. Background

2.1 Literature Review

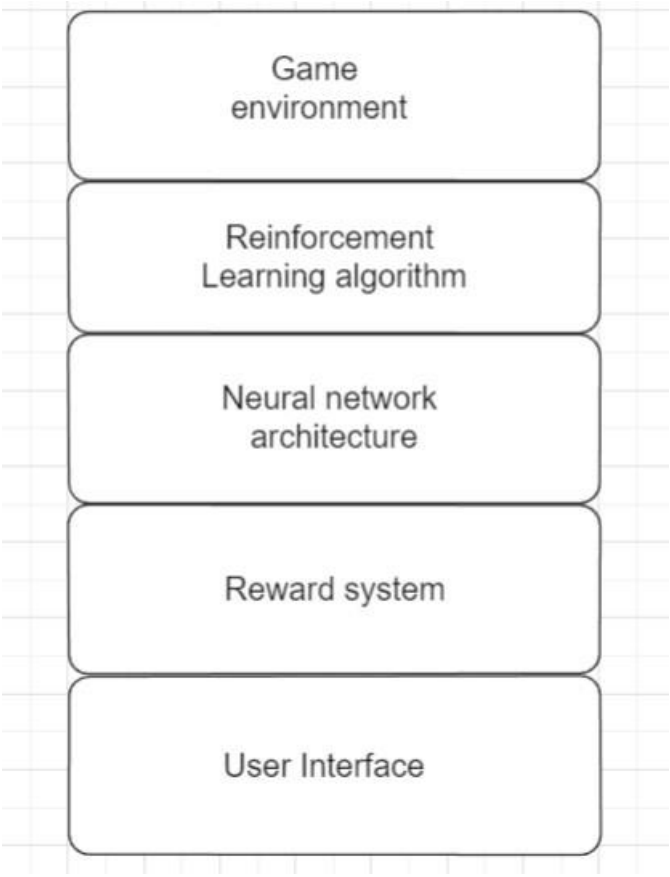
Reinforcement learning is an important area of study because it may enable society to automate tasks that we, in the past, never thought could be automated. Autonomous driving vehicles is one of the uses for reinforcement learning. Other uses may comprise having robots that can perform tasks like preparing ingredients and cooking food, with the same robot.

Raycast2D method is used to check if the snake collides with its own body or with the game border. This is done by casting a ray from the head of the snake towards its body segments or towards the game border. If the ray intersects with any of these objects, the information regarding it is received by the snake.

Ray casting is a common technique used in game development for detecting collisions between objects, detecting where a player is aiming a weapon or other object, and for generating realistic lighting and shadows. preparing several different dishes without any human intervention or any specific programming for the executed tasks. Stock market trading can also be a task executed by a reinforcement learning agent, that instead of being programmed with specific rules, can learn the rules of best trading by itself.

3. System Design & Methodology

3.1 System Architecture

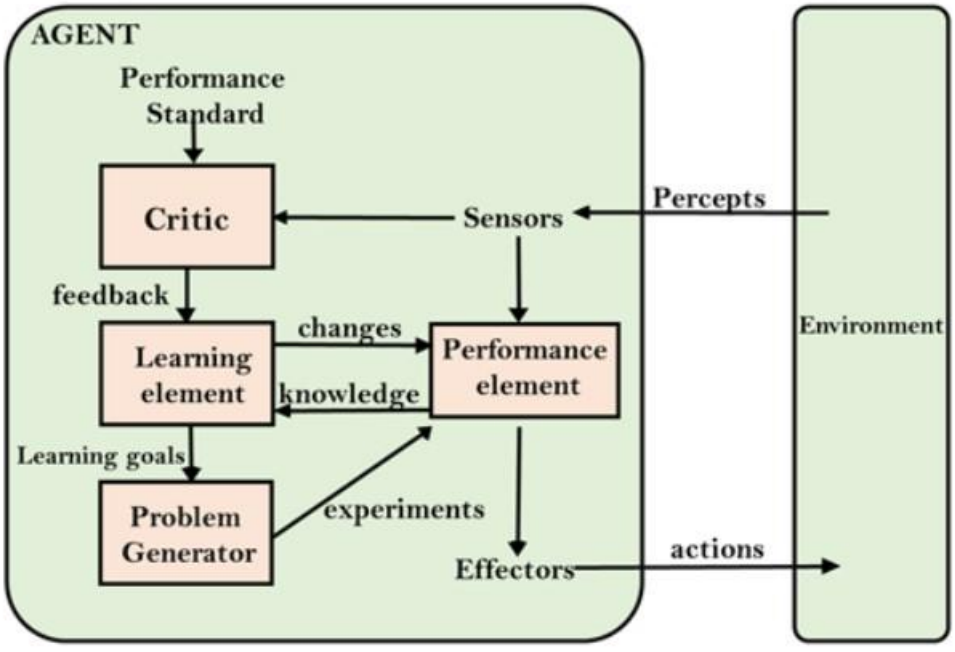


Raycast2D method is used to check if the snake collides with its own body or with the game border. This is done by casting a ray from the head of the snake towards its **body segments or towards the game border. If the ray intersects with any of these** objects, the game is over. Ray casting is a common technique used in game development for detecting collisions between objects, detecting where a player is aiming a weapon or other object, and for generating realistic lighting and shadows.

3.2 Agent

We used **learning agent** in this project as it can learn from its past experiences. It starts to act with basic knowledge and then able to act and adapt automatically through learning. A learning agent has mainly four conceptual components, which are:

1. **Learning element:** It is responsible for making improvements by learning from environment
2. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
3. **Performance element:** It is responsible for selecting external action
4. **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.



3.3 Environment

- Fully observable environment - When an agent sensor is capable to sense or access the complete state of an agent at each point in time, it is said to be a fully observable environment.

- Single-agent environment – When there is only one agent in the Environment, it is said to be a single-agent environment.

- Sequential environment – In this type of environment, the previous decisions can affect the future decisions. The next action of the agent depends on what action he has previously taken and what action he is supposed to take in the future.

3.4 Development Environment

- Scripting language – We used C# for this project as a script must be Attached to a GameObject in the scene in order to be called by Unity. So C# is the scripting language that is supported by Unity.

- Game Engine – We used Unity for developing our 2-D snake game as it is an easy platform to create a basic 2-D game and implement Reinforcement Learning with it.

- Reinforcement Learning Frameworks -We used pytorch to implement reinforcement learning algorithms.

- Neural network library – We used Pytorch to implement the neural network architecture used for processing game state.

- IDE – We used Visual Studio code for writing and debugging the C# script that is integrated with Unity.

- Visualization – We used TensorBoard to get the statistics and visualizations about the cumulative reward and episode length.

4. Implementation

There are 4 scripts used in this project – for Snake, for Food, for Score and for Highscore.

1) Snake Script

This is a script written in C# for a Snake game in Unity using the Unity Machine Learning Agents (ML-Agents) package. The ML-Agents package is used to implement the artificial intelligence (AI) for the snake in the game. The AI is trained using reinforcement learning.

The script defines a class called "Snake" that extends the "Agent" class provided by the ML-Agents package. The "Agent" class represents an AI agent in the environment. The "Snake" class has several fields and methods, including:

CollectObservations(VectorSensor sensor) : This method is called every time an observation is requested by the agent. It adds observations to the provided VectorSensor object, which is used to represent the agent's observation space.

OnActionReceived(ActionBuffers actions) : This method is called every time the agent receives an action. It takes in an ActionBuffers object, which contains the actions requested by the agent.

Start(): This method is called once at the beginning of the agent's lifecycle. In this implementation, it simply resets the agent's state.

Update(): This method is called every frame of the simulation. In this implementation, it handles user input to update the agent's direction, computes rewards for the agent based on the distance to the target, and updates the agent's score based on its progress.

FixedUpdate(): This method is called every physics update, at a fixed interval. In this implementation, it updates the positions of the agent's body segments to follow the head, and moves the agent forward in its current direction.

Grow(): This is a custom method used to add a new body segment to the agent's body. It instantiates a new **Transform** object based on the provided **segmentPrefab**, sets its position to be the same as the agent's last body segment, and adds it to the list of body segments.

reset_state() is a method that resets the hidden state of the model to its initial value.

Heuristic() method used to allow a human player to control the game character using the keyboard arrow keys. The method maps the keyboard inputs to specific actions that the game character can perform, such as moving up, down, left, or right.

OnTriggerEnter2D() is a method in Unity game engine that gets called when a 2D collider enters or exits the trigger attached to the game object. This method is used to detect collisions or triggers between objects in a 2D game environment.

2) Food Script

The script for a food object in a game defines a GridArea with BoxCollider2D which represents the area where the food can spawn. Upon starting, the RandomizePosition() function is called, which sets the food's position to a random point within the area defined by GridArea. If the food collides with the Player object, the OnTriggerEnter2D() function is called and the food's position is randomized again.

3) HighScore Script

This is a script written in C# for a Unity game engine project that displays the high score on a UI text element.

The script starts by importing the necessary libraries: System.Collections, System.Collections.Generic, UnityEngine, and UnityEngine.UI.

It then declares a variable 'highScore' of type Text, which is the UI text element that will display the high score.

In the Start() function, the script initializes the 'highScore' variable by getting the component of the Text element attached to the same game object as the script.

In the Update() function, the script updates the 'highScore' text to display the current high score. It does this by accessing the 'HighScore' static variable from another script called 'ScoreScript', which stores the high score value.

4) Score Script

This is a script written in C# for a Unity game engine project that tracks and displays the player's score and high score.

It then declares two public static integer variables: 'TotalScore' and 'HighScore', which will store the current score and high score values. These variables are public and static so that they can be accessed and modified from other scripts.

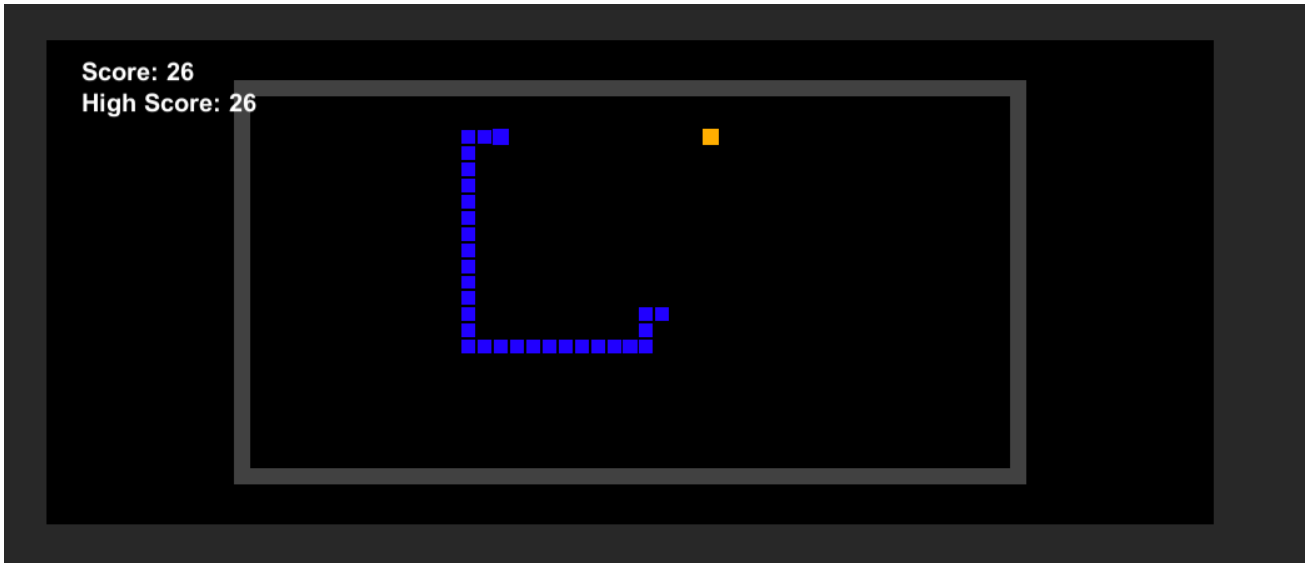
The script also declares a variable 'score' of type Text, which is the UI text element that will display the player's score.

In the Start() function, the script initializes the 'score' variable by getting the component of the Text element attached to the same game object as the script.

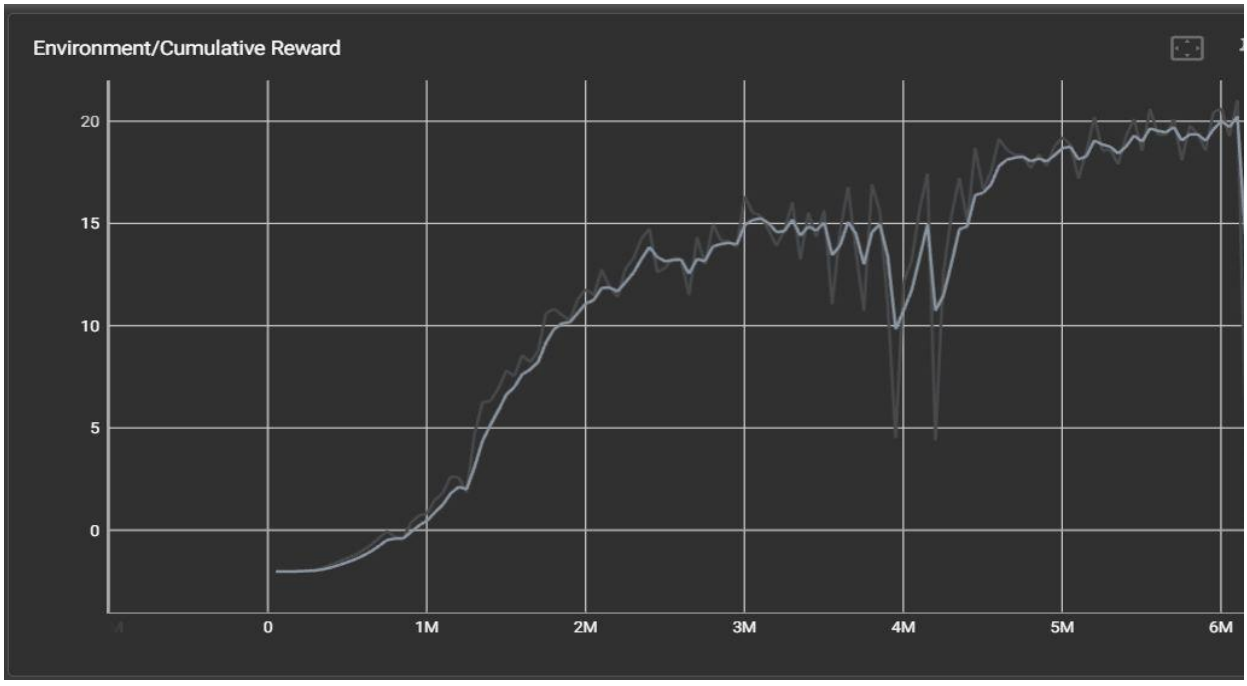
In the Update() function, the script updates the 'score' text to display the current score. It does this by accessing the 'TotalScore' static variable and concatenating it with the "Score: " string.

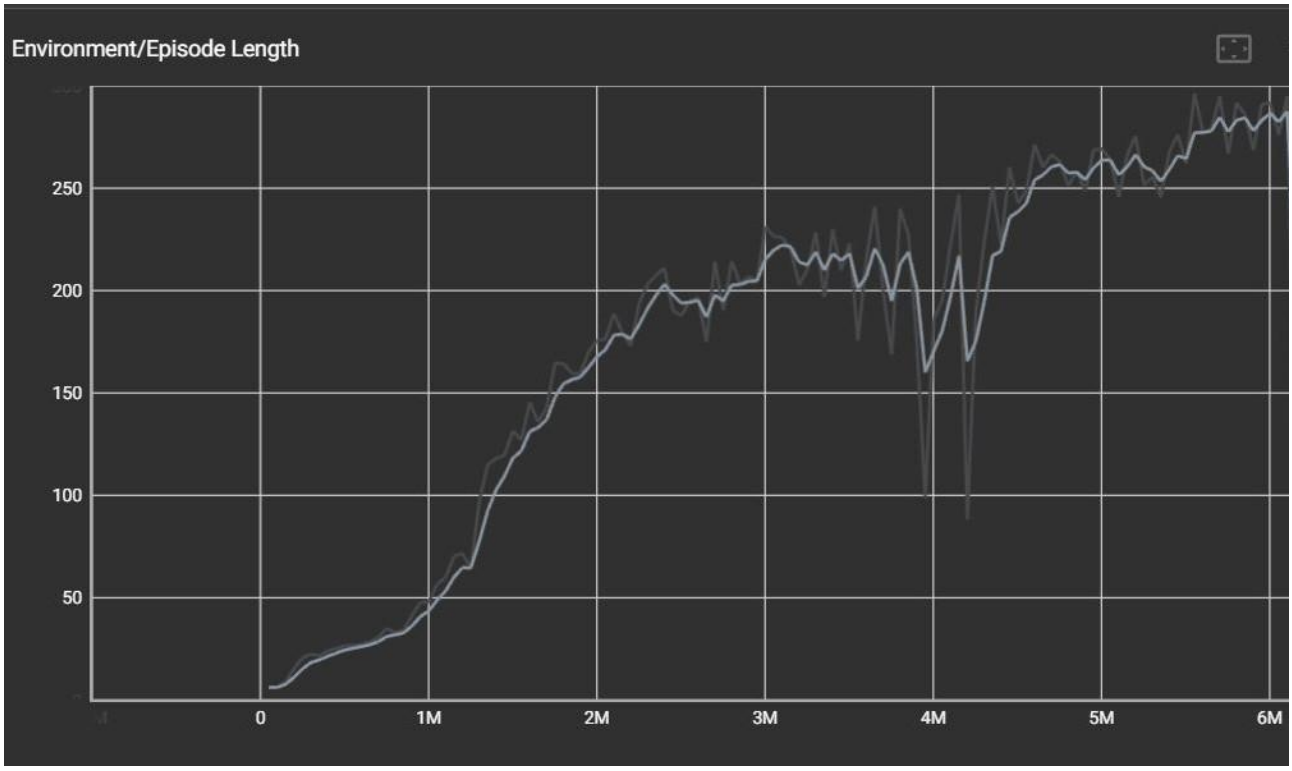
The script also checks if the current score is higher than the current high score. If it is, then the script updates the 'HighScore' static variable to match the current score.

5. Results



We used TensorBoard to perform visualization of the training process. It helps in tracking metrics such as loss and accuracy. It also provides visuals of the cumulative reward w.r.t. number of rounds.




Environment/Episode Length
250
200
150
100
50
0
1M
2M
3M
4M
5M
6M