CHROME DINO GAME

Submitted by

Chaykam Varun Reddy [RA2111047010200]
Koduru Avinash Reddy[RA2111047010201]

Under the Guidance of

Dr. R.UDENDHRAN

Assistant Professor
Department of Computational Intelligence

In partial satisfaction of the requirements for the degree of

BACHELORS OF TECHNOLOGY
in
Artificial Intelligence

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

May 2024

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this Course Project Report titled **"CHROME DINO GAME"** is the bonafide work done by **CHAYKAM VARUN REDDY [RA2111047010200], KODURU AVINASH REDDY [RA2111047010201]** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE
Faculty In-Charge
Dr.R. UDENDHRAN
Assistant Professor
Department of Computational Intelligence
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

HEAD OF THE DEPARTMENT
Dr. R Annie Uthra
Professor and Head ,
Department of Computational Intelligence,
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

TABLE OF CONTENTS

ABSTRACT

The project aims to develop a Chrome Dino game player using reinforcement learning (RL) techniques, leveraging a Deep Q-Network (DQN) for implementation. The Chrome Dino game is a classic browser-based game where a dinosaur must navigate obstacles by jumping or ducking. RL offers a promising approach to train an agent to effectively navigate the game environment. The game environment is simulated, defining states, actions, rewards, and the learning algorithm. States represent the game's current state, including obstacle positions and the dinosaur's status. Actions correspond to the dinosaur's possible moves. Rewards are designed to encourage survival and penalize collisions with obstacles. The DQN is used to train the agent, with plans to optimize its performance by adjusting the reward function. Challenges include designing a reward function that balances incentivizing survival, obstacle avoidance, and progress. The project aims to evaluate the agent's performance and explore further enhancements to improve gameplay. The outcome will be a trained RL agent capable of playing the Chrome Dino game proficiently, demonstrating the effectiveness of RL in gaming applications.

INTRODUCTION

The Chrome Dino game player project seeks to apply reinforcement learning (RL) techniques, specifically utilizing a Deep Q-Network (DQN), to develop an autonomous agent capable of playing the classic Chrome Dino game. This game, available in Google Chrome's offline mode, presents a simple yet challenging scenario where a dinosaur must navigate a desert landscape, jumping over cacti and ducking under flying pterodactyls to avoid obstacles. RL offers a powerful framework for training such an agent by allowing it to learn from its interactions with the game environment.

The project's primary goal is to create an RL agent that can effectively learn to play the Chrome Dino game, demonstrating the capabilities of RL in a gaming context. To achieve this, the project will first simulate the game environment, defining the game's states, actions, rewards, and the learning algorithm. States will include information about the dinosaur's position, the positions of obstacles, and other relevant game variables. Actions will correspond to the dinosaur's possible moves, such as jumping and ducking. Rewards will be designed to incentivize behaviors that lead to successful gameplay, such as avoiding obstacles and progressing further in the game.

A key aspect of the project will be the optimization of the DQN through the modification of the reward function. The reward function plays a crucial role in shaping the agent's behavior, influencing its decisions during gameplay. By adjusting the reward function, the project aims to improve the agent's performance and enhance its ability to learn effective strategies for playing the game.

PROBLEM STATEMENT

The Chrome Dino game presents a captivating challenge for players, requiring precise timing and quick decision-making to navigate obstacles and achieve high scores. However, manually mastering the game's dynamics can be demanding and time-consuming. Thus, the need arises for an automated solution that leverages Reinforcement Learning (RL) techniques to train an agent capable of playing the game proficiently.

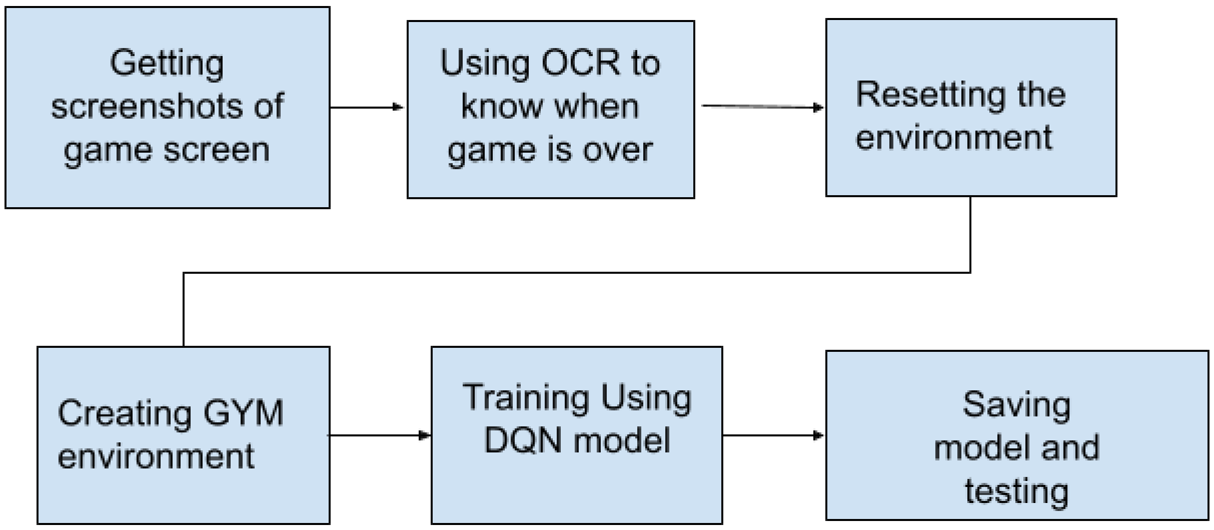
1.Environment Simulation: Creating a faithful simulation of the Chrome Dino game environment, including obstacle generation, dinosaur movement, and collision detection.

2.RL Problem Formulation: Defining the RL problem by specifying states, actions, rewards, and the learning algorithm. This involves designing an effective state representation, determining appropriate actions for the agent, defining reward structures that incentivize desired behaviors, and selecting an RL algorithm suitable for training the agent.

3.Training Efficiency: Implementing the chosen RL algorithm and optimizing its parameters to ensure efficient and effective training of the agent. This includes exploring techniques for accelerating convergence and improving sample efficiency.

4.Performance Evaluation: Assessing the trained agent's performance by evaluating its ability to navigate the game environment, evade obstacles, and achieve high scores.

ARCHITECTURAL DIAGRAM


Getting screenshots of game screen
Using OCR to know when game is over
Resetting the environment
Creating GYM environment
Training Using DQN model
Saving model and testing

In the context of the Chrome Dino game, the DQN takes screenshots or states of the game environment as input. These inputs are typically preprocessed to extract relevant features, such as the positions of obstacles and the dinosaur, before being fed into the neural network. The output of the DQN is a Q-value for each possible action, indicating the expected cumulative reward if that action is taken in the current state.

Based on these Q-values, the agent selects an action to take in the game. This action is then executed in the game environment, and the resulting reward and next state are observed. This process is repeated iteratively, with the DQN continuously updating its Q-values based on new experiences, gradually improving its policy for playing the game.

MODULES APPLIED

1. GYM -
To train our model using DQN, we needed a Gym environment. Since we were using a custom environment for the Chrome Dino game, we used Gym to convert our custom environment into a Gym environment. This conversion process involved mapping the states, actions, and rewards of our custom environment to the corresponding Gym interface, allowing us to seamlessly integrate our environment with the DQN training process. By leveraging Gym's flexibility and compatibility with RL algorithms, we were able to train our agent effectively and efficiently, demonstrating the versatility of Gym in supporting a wide range of RL applications.

2. PyDirectInput -
To enable the agent to interact with the game environment, we are using a library that allows the agent to take actions such as pressing the SPACE bar, moving the mouse, and clicking the mouse. This library bridges the gap between the agent's decision-making process and the game's interface, enabling the agent to control the dinosaur in the Chrome Dino game based on its learned policy. By integrating this library with our training pipeline, we can fully automate the training process and evaluate the agent's performance in a real game environment.

3. Tesseract -
Using OCR for detecting when the game is over and needs to be restarted is a clever approach. By recognizing specific visual cues or text in the game interface that indicate the game has ended, such as a "Game Over" message, the agent can reset the game state and begin a new episode. This allows the agent to continuously improve its gameplay through iterative training cycles, enhancing its ability to achieve higher scores and navigate the game environment more effectively.

4. MSS -
Using the MultipleScreenShots (MSS) library to capture input or states from the custom environment is a practical choice. This library likely helps in capturing screenshots of the game at regular intervals or when specific events occur, providing the necessary visual information for the agent to make decisions. By integrating MSS into the training pipeline, you can ensure that the agent receives accurate and up-to-date information about the game state, enabling it to learn more effectively and improve its performance over time.

5. OpenCV -
Converting the input from RGB to black and white is a smart move for improving interpretability, especially for models like DQN where simpler inputs can lead to more effective learning. Using the OpenCV module for this purpose is a practical choice, as it offers efficient tools for image processing and conversion. This preprocessing step can help the DQN model focus on relevant features of the game environment, potentially leading to faster and more accurate learning.

6. PyTorch -
Using PyTorch to import the DQN model is a great choice, as PyTorch offers a powerful framework for building and training neural networks. The DQN model, being the core of your agent, will benefit from PyTorch's flexibility and efficiency in handling deep learning models. With PyTorch, you can easily define and customize your DQN architecture, train it on your game environment, and fine-tune it to improve its performance. Additionally, PyTorch provides tools for debugging and visualizing your model, helping you understand its behavior and make necessary adjustments. Overall, PyTorch's capabilities make it an excellent choice for implementing complex deep learning models like DQN for reinforcement learning tasks.

METHODOLOGY

1. Creating an Environment:

Creating a custom environment for the Chrome Dino game with the help of MSS, Tesseract, PyDirectInput, and OpenCV to handle input, decision-making, and screen interpretation is a sophisticated approach. Converting this custom environment into a Gym environment allows you to leverage Gym's standardized interface for reinforcement learning, enabling easier integration with the DQN model and other Gym-compatible algorithms. Validating the Gym environment ensures that it meets the required criteria for compatibility, ensuring a smooth training process for your agent. Overall, this process demonstrates a comprehensive and thoughtful approach to adapting a real-world game scenario into a suitable environment for reinforcement learning.

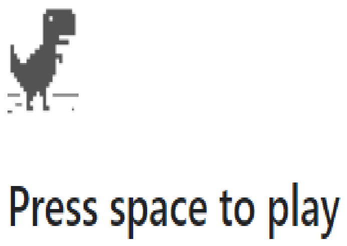
2. Training our model:

We Trained our DQN model for almost 70000 timesteps which is a significant amount of time. A score of 316 indicates that your agent has learned to play the Chrome Dino game reasonably well. It's impressive how the integration of Convolutional Neural Networks with Q-learning has enabled your agent to make decisions based on image inputs, highlighting the power of deep reinforcement learning in handling complex visual environments. This result demonstrates the effectiveness of your RL technique and the capability of your agent to learn and adapt to the game environment.

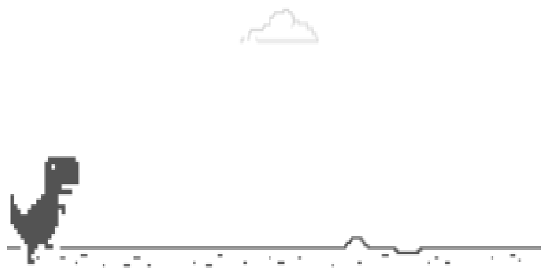
3. Saving and testing:

It's a smart strategy to save our model checkpoints periodically during training, especially at key milestones like every 1000 timesteps. This allowed us to track the progress of your model and identify which checkpoints correspond to the highest rewards. Testing the model using the best-performing checkpoint, which yielded a maximum score of 243, provides valuable insights into its overall performance and highlights areas for potential improvement. Analyzing the behavior of the model at different stages of training can help refine your training process and optimize the agent's performance further. Overall, this iterative approach to training and testing contributes to the overall success of your reinforcement learning project.

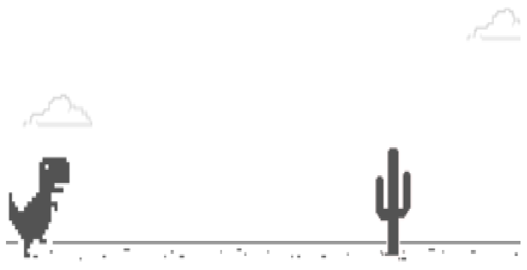
States:



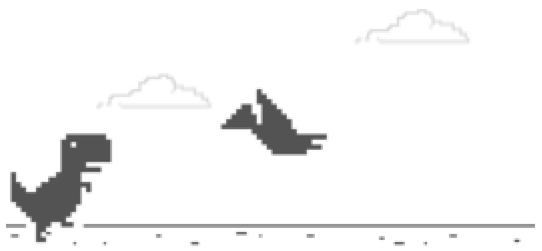
1) Starting State



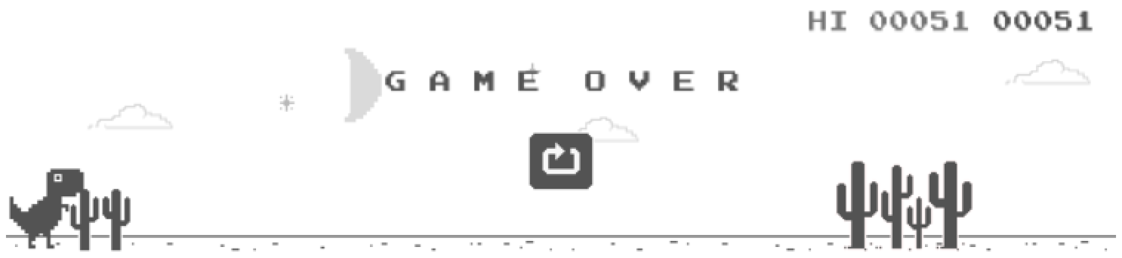
2) No obstacle



3) Cactus as obstacle



4) Bird as obstacle



5) End state

Actions:

There are 3 actions agent can take

- JUMP
- DUCK
- DOING NOTHING

Rewards:

For each time step of for each frame the Dino or agent is alive a reward of +1 is given

DQN model:

The following formula is applied to get optimal policy by calculating the Q-values.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma maxa' Q(s',a') - Q(s,a)]$$

CODE

Loading Necessary Libraries:

```
from mss import mss
import pydirectinput
import cv2
import numpy as np
import pytesseract
from matplotlib import pyplot as plt
import time
from gymnasium import Env
from gymnasium.spaces import Box, Discrete
import gymnasium
```

Creating environment:

```
class WebGame(gymnasium.Env):
    def __init__(self):
        super().__init__()
        # Setup spaces
        self.observation_space = Box(low=0, high=255, shape=(1,83,100), dtype=np.uint8)
        self.action_space = Discrete(3)
        # Capture game frames
        self.cap = mss()
        self.game_location = {'top': 300, 'left': 0, 'width': 600, 'height': 500}
        self.done_location = {'top': 405, 'left': 630, 'width': 660, 'height': 70}


    def step(self, action):
        action_map = {
            0:'space',
            1: 'down',
            2: 'no_op'
        }


        if action !=2:
            pydirectinput.press(action_map[action])


        terminated, done_cap = self.get_done()
        observation = self.get_observation()
        info = {}
        reward=1
        truncated=False
        return observation, reward, terminated, truncated, info


    def reset(self,seed=None):
        time.sleep(1)
        pydirectinput.click(x=150, y=150)
        pydirectinput.press('space')
        observation = self.get_observation()
        info = {}
        return obs, info


    def render(self):
        cv2.imshow('Game', self.current_frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            self.close()


    def close(self):
        cv2.destroyAllWindows()

    def get_observation(self):
        raw = np.array(self.cap.grab(self.game_location))[:,:,:3].astype(np.uint8)
        gray = cv2.cvtColor(raw, cv2.COLOR_BGR2GRAY)
        resized = cv2.resize(gray, (100,83))
        channel = np.reshape(resized, (1,83,100))
        return channel

    def get_done(self):
        done_cap = np.array(self.cap.grab(self.done_location))
        done_strings = ['GAME', 'GAHE']
        done=False
        # if np.sum(done_cap) < 44300000:
        #     done = True
        done = False
        res = pytesseract.image_to_string(done_cap)[:4]
        if res in done_strings:
            done = True
        return done, done_cap
```

Checking if our environment is upto GYM criteria:

```
import os
# Import Base Callback for saving models
from stable_baselines3.common.callbacks import BaseCallback
# Check Environment
from stable_baselines3.common import env_checker

env_checker.check_env(env)
```

Creating and training our model:

```
from stable_baselines3 import DQN
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.vec_env import DummyVecEnv, VecFrameStack

env = WebGame()

model = DQN('CnnPolicy', env, tensorboard_log=LOG_DIR, verbose=1, buffer_size=600000,
learning_starts=1000)

model.learn(total_timesteps=70000, callback=callback)
```

Loading and testing our model:

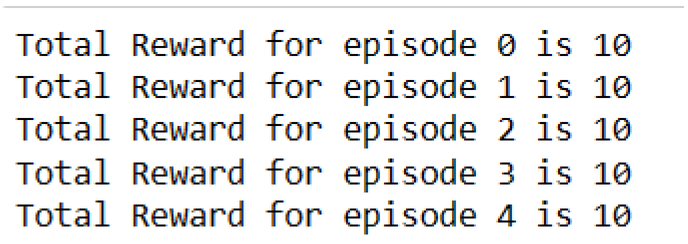
```
model.load('C:\\Users\\varun\\RL\\train2\\best_model_60000.zip')

for episode in range(5):
    obs, info = env.reset()
    terminated = False
    total_reward = 0
    while not terminated:
        action, _states = model.predict(obs)
        obs, reward, terminated, truncated, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode, total_reward))
```

RESULT

Our model gave us a score of 314 during Training and during testing it gave us a score of 243, which is low but further optimization of model and environment parameters and reward function can help us improve this.

Reward is calculated as per each frame it stays alive, the rewards were as below:

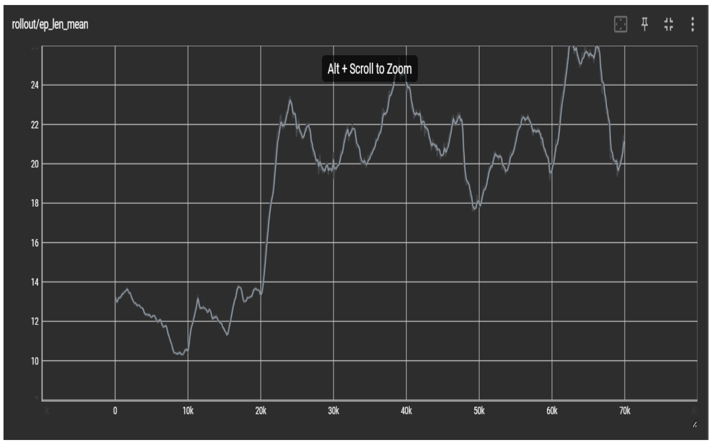


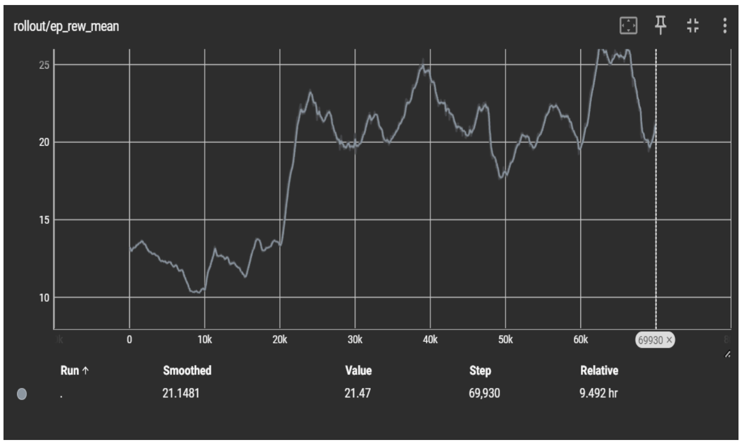
```
Total Reward for episode 0 is 10
Total Reward for episode 1 is 10
Total Reward for episode 2 is 10
Total Reward for episode 3 is 10
Total Reward for episode 4 is 10
```

Training logs are as below:

Mean episode length for every 1000 timesteps:

Mean reward for every 1000 timesteps:

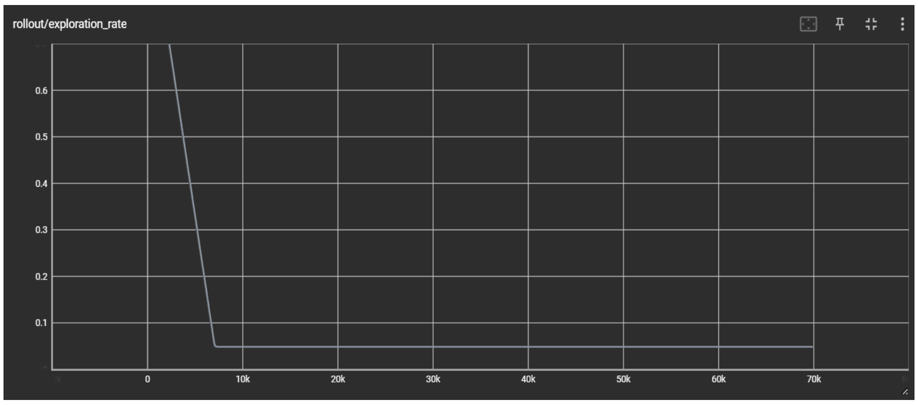




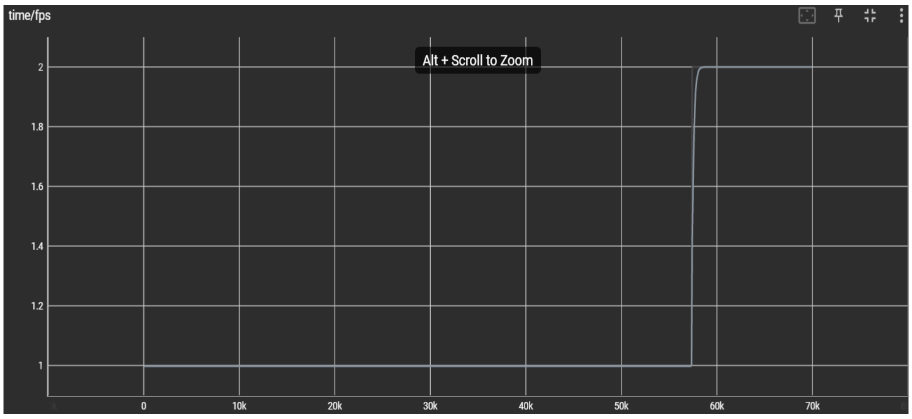
Run ↑	Smoothed	Value	Step	Relative
.	21.1481	21.47	69,930	9.492 hr

As we are rewarding the agent for every frame it stays alive both Graphs are same

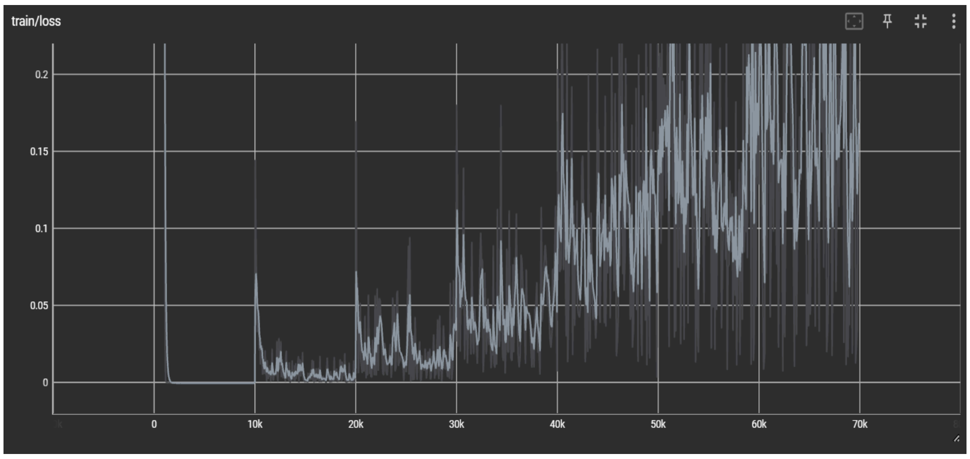
Exploration Rate:



Achieved FPS:



Training loss incurred:



CONCLUSION

In conclusion, the development of a Reinforcement Learning-Based Chrome Dino Game Player represents a significant exploration into the capabilities of artificial intelligence, particularly within the realm of gaming. Throughout this project, we have embarked on a journey to train an RL agent capable of autonomously playing the Chrome Dino game with proficiency.

We began by understanding the intricacies of the Chrome Dino game and formulating the RL problem, defining states, actions, rewards, and selecting suitable RL algorithms. Leveraging techniques such as Q-learning or Deep Q-Networks, we trained our agent to learn optimal strategies for navigating the game environment, avoiding obstacles, and maximizing its survival time.

Through rigorous experimentation and fine-tuning, we optimized the performance of our RL agent, ensuring efficiency and effectiveness in its gameplay. Evaluation of the trained agent demonstrated its ability to adapt to dynamic game scenarios, evade obstacles with precision, and achieve competitive scores.

Furthermore, by integrating our RL agent with the actual Chrome Dino game interface, we showcased the practical applicability of autonomous agents in real-world gaming environments. The seamless interaction between the agent and the game exemplifies the potential of AI-driven solutions to enhance user experiences and automate repetitive tasks.

As we reflect on the outcomes of this project, we recognize the broader implications of our efforts. Beyond gaming, the success of our RL-based Chrome Dino game player underscores the versatility and adaptability of RL techniques in addressing complex problems across diverse domains. From robotics to finance, RL offers a powerful framework for learning optimal decision-making strategies in uncertain and dynamic environments.

FUTURE WORKS

While the Reinforcement Learning-Based Chrome Dino Game Player represents a significant achievement, there are several avenues for future exploration and enhancement:

1. Advanced RL Algorithms: Investigate more advanced RL algorithms such as Proximal Policy Optimization (PPO) or Trust Region Policy Optimization (TRPO) to further improve the agent's learning efficiency and performance.

2. Transfer Learning: Investigate techniques for transferring knowledge learned from the Chrome Dino game to other similar games or environments, accelerating the learning process in new domains.

3. Adaptive Learning: Implement mechanisms for the RL agent to adapt its learning strategy dynamically based on changes in the game environment or the player's performance, ensuring robustness and adaptability.

4. Human-Agent Interaction: Explore ways to incorporate human feedback into the RL training process, allowing human players to interact with the agent and provide guidance or corrections.

5. Reward Function: By changing or incorporating a good reward function we can significantly improve agents performance.

6. Modules: Using other modules can help us achieve a good FPS which is crucial while training.

By pursuing these future works, we can further advance the capabilities and applications of reinforcement learning in gaming and beyond, unlocking new possibilities for intelligent autonomous systems and human-machine collaboration.