



## Assignment Cover Sheet

**Subject Code:** CSCI218  
**Subject Name:** Foundations of Artificial Intelligence  
**Submission Type:** Online  
**Assignment Title:** Group Assignment  
**Student Name:** Basim Jaaskelainen, Mariah Khalifa, Riva Pereira  
**Student Number:** 7066788, 7847233, 7839078  
**Student Phone/Mobile No.** +971524134738, +971562755688, +971505578543  
**Student E-mail:** [bj575@uowmail.edu.au](mailto:bj575@uowmail.edu.au), [mak872@uowmail.edu.au](mailto:mak872@uowmail.edu.au), [rnp649@uowmail.edu.au](mailto:rnp649@uowmail.edu.au)  
**Lecturer Name:** Dr Farhad Oroumchian  
**Due Date:** 1/12/2023  
**Date Submitted:** 30/11/2023

**PLAGIARISM:**

The penalty for deliberate plagiarism is FAILURE in the subject. Plagiarism is cheating by using the written ideas or submitted work of someone else. UOWD has a strong policy against plagiarism. The University of Wollongong in Dubai also endorses a policy of non-discriminatory language practice and presentation.

**PLEASE NOTE:** STUDENTS MUST RETAIN A COPY OF ANY WORK SUBMITTED

**DECLARATION:**

I/We certify that this is entirely my/our own work, except where I/we have given fully-documented references to the work of others, and that the material contained in this document has not previously been submitted for assessment in any formal course of study. I/we understand the definition and consequences of plagiarism.

**Signature of Student:**

**Optional Marks:**

**Comments:**

**Lecturer Assignment Receipt** (To be filled in by student and retained by Lecturer upon return of assignment)

**Subject:**

**Student Name:**

**Due Date:**

**Signature of Student:**

**Assignment Title:**

**Student Number:**

**Date Submitted:**

**Student Assignment Receipt** (To be filled in and retained by Student upon submission of assignment)

**Subject:**

**Student Name:**

**Due Date:**

**Signature of Lecturer**

**Assignment Title:**

**Student Number:**

**Date Submitted:**

# T-REX DINO BOT

CSCI218 Project

## Team

Basim Jaaskelainen - 7066788

Mariah Khalifa - 7847233

Riva Pereira - 7839078



# Table of Contents

Executive Summary .....	3
Background Information About the Project .....	4
Naming Conventions Used.....	4
Dependencies Installed.....	4
Pytorch .....	4
Reinforcement Learning.....	4
Image Processing.....	4
Screen Capturing.....	4
Imports .....	5
Objectives and Goals .....	5
Overview of Reinforcement Learning.....	5
How to Run the Executable File.....	6
Description of the Reinforcement Learning Model .....	6
Deep Q-Network Implementation .....	7
Game Environment .....	7
Data Processing and Interaction with the Model .....	7
Algorithms and Techniques Used.....	7
Game Environment Setup .....	8
Observation Space .....	8
Action Space.....	8
Step Method .....	8
Reset Method .....	8
Get Observation.....	9
Get Done.....	9
Data Collection.....	9
Model Training Process.....	9
Screenshots and Explanation of Output.....	10
Model Trained with 100,000 Timesteps.....	12
Integration with Chrome Dino Game .....	16
Testing Methodology .....	16
Performance Metrics.....	17
Encountered Issues.....	18
Experimental Adjustments .....	18
Expanding the Dino's Field of View.....	18
Mitigating Random Actions.....	19
Challenges with Overfitting.....	19
Limitations.....	19
Conclusion .....	19



## Executive Summary

This report presents the application of reinforcement learning (RL) techniques to the Chrome Dino game, an offline game popular among Google Chrome users. The project's premise is the development and implementation of a Deep Q-Network (DQN) which would be trained and fine-tuned to automate gameplay.

The project's sole success criteria was the successful integration of the DQN model with the Chrome Dino game, in turn creating an environment where the model could learn, adapt, and evolve its gameplay strategies. As the game progresses, the model continuously refines its approach based on the feedback from its interactions with the game.

# Background Information About the Project

The Chrome Dino game is a simple and popular game that starts when you're offline on Google Chrome. In this game, you control a dinosaur that needs to jump over cacti and go under obstacles. It's easy to play but gets harder as you go, making it faster and more challenging.

This game is perfect for testing reinforcement learning (RL), a type of AI that learns by trying different things and seeing what works best. As the game gets harder, the AI has to learn and react faster, making it a great way to see how well reinforcement learning can work in games. This project uses the Chrome Dino game to show how AI can learn and get better at games, giving us insights into how reinforcement learning can be used in other gaming applications.

## Naming Conventions Used

This Program follows the standard pep8 style guide for python where classes use Camel Case along with Instances and Methods using Snake-Case and All Uppercase for constants. This aids in readability.

We utilised Jupyter Lab to code this project.

## Dependencies Installed

### Pytorch

```
!pip install torch torchvision torchaudio
```

### Reinforcement Learning

```
!pip install stable-baselines3[extra] protobuf==3.20.*  
!pip install gym
```

### Image Processing

```
!pip install Pillow
```

### Screen Capturing

```
!pip install pytesseract  
!pip install mss pydirectinput pytesseract
```

## Imports

```
from mss import mss
import pydirectinput
import cv2
import numpy as np
import pyteseract
from matplotlib import pyplot as plt
import time
from gym import Env
from gym.spaces import Box, Discrete
from stable_baselines3 import DQN
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.vec_env import DummyVecEnv, VecFrameStack
```

## Objectives and Goals

The primary objective of this project is to develop an artificial intelligence (AI) model which would be able to play the Chrome Dino game, utilizing the capabilities of reinforcement learning (RL). Our focus is on enabling the AI to not only comprehend the game's mechanics but also to enhance its performance progressively, like human learning patterns.

Our approach entails understanding the foundations of RL before and then applying these principles practically in the Chrome Dino game. And the end goal is to achieve a level of automation in the game where the AI operates independently and efficiently.

## Overview of Reinforcement Learning

Reinforcement Learning falls under the umbrella of machine learning and is where an agent would learn to make decisions within an environment through trial and error. It is a crucial technique in many domains, like financial modeling, robotics and gaming.

The mechanism for learning is based on the interaction between an agent and the environment defined/used. The agent would perform certain functions/actions and according to predefined conditions would receive a “reward” for this action. The more correct actions performed, the higher the reward.

The fundamental parts of our model (and any Reinforcement Learning model) are:

- **Agent:** The decision-maker/ or learner.
- **Environment:** The system with which the agent interacts (in this case the chrome browser)

- **Action:** possible moves or decisions the agent can make (in this case, jumping, crouching or no action)
- **State:** the current condition of the environment.
- **Reward:** feedback given to the agent, defined by the user, which is positive for desired outcomes and null for undesired ones.

## How to Run the Executable File

Download `best_model_90000.zip` and `dino_executable.py` in the same folder.

Install Tesseract in Program files folder using this link:

<https://digi.bib.uni-mannheim.de/tesseract/tesseract-ocr-w64-setup-5.3.3.20231005.exe>

Install all the dependencies:

```
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu121
pip install stable-baselines3[extra] protobuf==3.20.*
pip install opencv-python
pip install mss
pip install Pillow
pip install pytesseract
pip install mss
pip install pydirectinput
pip install gym
```

Go to the folder with the downloaded files in cmd.

Open <chrome://dino/> in chrome, move the window aside, and run the command:

```
python3 dino_executable.py
```

## Description of the Reinforcement Learning Model

Below is a description of the various components and how they are implemented in our model.

## Deep Q-Network Implementation

DQN was our chosen model architecture due to its effectiveness for handling high-dimensional input spaces. This means it is a ideal model for processing pixels in a scene. It utilizes a convolutional neural network to process the visual input, mapping raw pixel data to action values.

The model learns by updating Q-Value, which provides an estimate of the rewards for each action. Learning involves using a combination of random actions and known actions to maximize the total reward.

## Game Environment

The environment is set as a simulation of the chrome dino game, using the python mss library and the pydirectinput library for screen capture and keyboard input simulation respectively, and the OpenCV library for image processing. The observation space is a fixed dimension (100 by 83 pixels) and is captured as grayscale images, and the action space consists of 3 actions (0 for space, 1 for down and 2 for no action).

## Data Processing and Interaction with the Model

The raw data (pixel input) from the screen is resized and converted to grayscale which is the input passed to the DQN model, and this step is critical for allowing the DQN to perform well and reduce the input data complexity. After this, the DQN would predict the best action for each state, and over time the model would learn from the feedback.

The model would be trained by having random episodes of the game, which would be terminated when the dino has crashed into the cactus or the bird, and whenever an action is done which does not result in the dino crashing, the reward is incremented by one.

## Algorithms and Techniques Used

The DQN algorithm was selected for its efficiency in handling high-dimensional input spaces, such as pixels from a game screen. The model uses a convolutional neural network (CNN) to process the visual input from the game.

The core feature of DQN is the use of a neural network to approximate Q-values, which significantly enhances the ability of the algorithm to handle complex state spaces of video



games. And the learning mechanism is built around updating the Q-Values, which provides an estimate at the rewards for each action across the different game states. This helps the DQN to learn optimal strategies over time by continuously adapting its action-value estimations.

In the context of the Chrome Dino Game, the model would be tailored to the specific action space of the game, considering discrete actions (jumping, crouching, or doing nothing). The reward system would play a vital part in the learning process, as this provides incentives for avoiding obstacles and collisions would result in a lower reward score.

## Game Environment Setup

In developing the game environment for the Chrome Dino game, the WebGame class, extending the Env class from reinforcement learning libraries, plays a pivotal role. This environment is created using Python and integrates several key libraries to simulate interaction with the game.

The WebGame class emulates a gymnasium-like environment structured for the Chrome Dino game and interaction with a reinforcement learning agent

### Observation Space

The `__init__` method sets up the observation space as a grayscale image of the game screen, with dimensions (1, 83, 100). The mss library captures this screen data, focusing on areas relevant to gameplay and game completion status.

### Action Space

Defined as a discrete space offering three actions: 'space', 'down', and 'no\_op' (no operation), enabling the agent to perform actions within the game environment.

### Step Method

Executes actions within the game, triggering specific actions using `pydirectinput` based on the agent's input. This method returns updated observations, rewards (defaulted to 1), termination status, and additional information.

### Reset Method

Initiates the game environment, simulating a restart by emulating a mouse click and pressing the 'space' key. It returns the initial observation after resetting the game.

## **Get Observation**

Captures and processes the game screen using screen capturing with mss, converting it to grayscale, resizing to predefined dimensions, and returning the processed image as the observation.

## **Get Done**

Examines a specific game screen area using pytesseract, checking for predefined characters ('G' in this case) to determine if the game should terminate.

This structured environment design mirrors the gymnasium format, facilitating interaction between the reinforcement learning agent and the Chrome Dino game environment.

## **Data Collection**

Data collection involved capturing pixel data from the game screen. This data was used as input for the DQN model, allowing it to learn and make decisions based on real-time visual information.

## **Model Training Process**

The training process involved running multiple episodes of the game, with the model learning from each run. The model's parameters, such as the buffer size and learning rate, were fine-tuned during this process for optimal performance.

## Screenshots and Explanation of Output

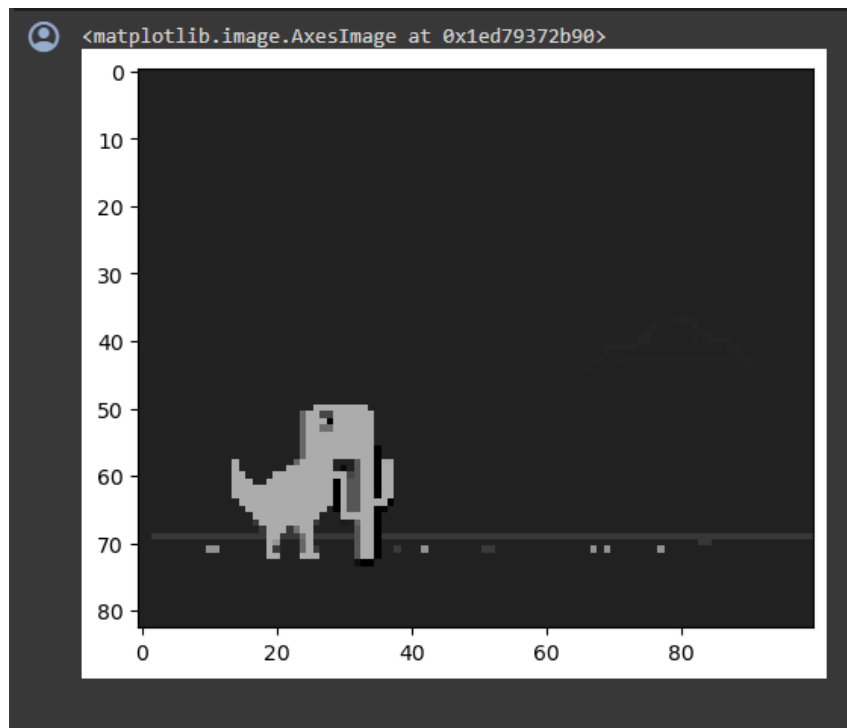


Figure 1 - Observation Space Screen Capture

Above is what the CNN would take as input and process. It is a scan of what is on the users screen in a predefined array (game\_location). After the initial scan is taken, it would be converted to grayscale first, then resized and reshaped and stored in the channel variable which would then be returned as part of the defined get\_observation() function.

```
Total Reward for episode 0 is 16
Total Reward for episode 1 is 15
Total Reward for episode 2 is 13
Total Reward for episode 3 is 13
Total Reward for episode 4 is 13
Total Reward for episode 5 is 13
Total Reward for episode 6 is 14
Total Reward for episode 7 is 11
Total Reward for episode 8 is 13
Total Reward for episode 9 is 15
```

Figure 2 - Output of Test Execution

Initially, we decided to test the code for interacting with the environment and the random move, and we displayed the reward for each episode. As you can see, the reward is fairly standard, averaging at around 11-16, as this is before any obstacle has been crossed.

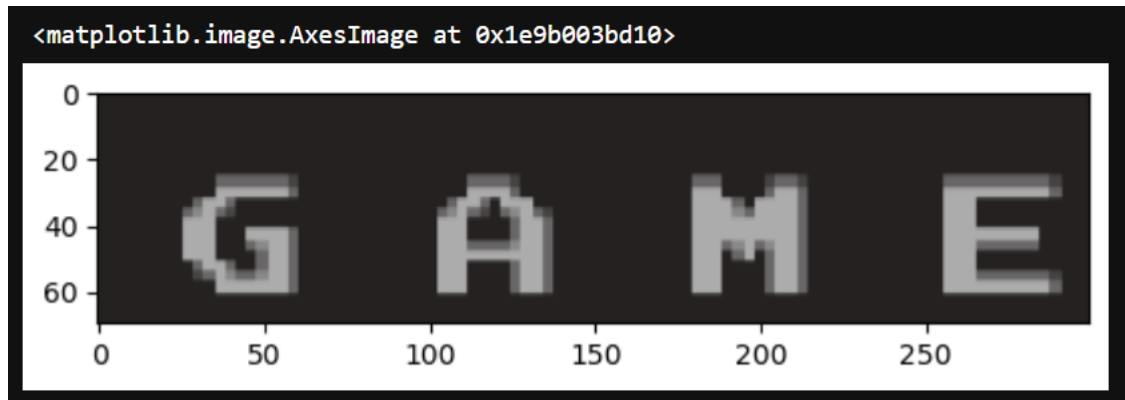


Figure 3 - Screen Capture for Game reset

The above is what would be displayed when the Game is over and since the location is constant we can specify the done\_location pixel range at the beginning of the code. The program would constantly be using OCR to recognize if the character G is present in that pixel range, meaning the game is over and the dinosaur has crashed. The library we have used to achieve this is PyTesseract.

We defined a array of possible outputs which we identified after doing multiple test, and we noticed that the letter G was present in the beginning of all of them. So we defined a conditional statement to check if the first letter detected by the PyTesseract.image\_to\_string() function is the letter G, and if so we would return a variable called done for which the value would be set as True. If done is true, the reinforcement learning model would either restart the game or would end the training, depending on the number of steps it has completed up until that point.

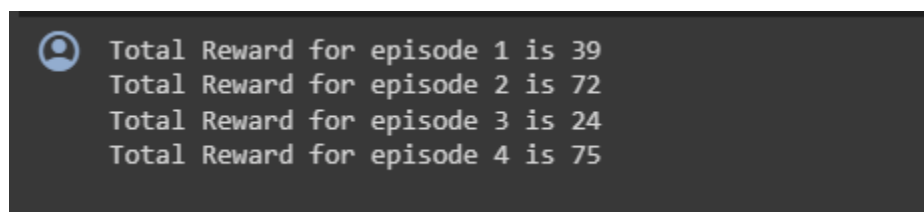
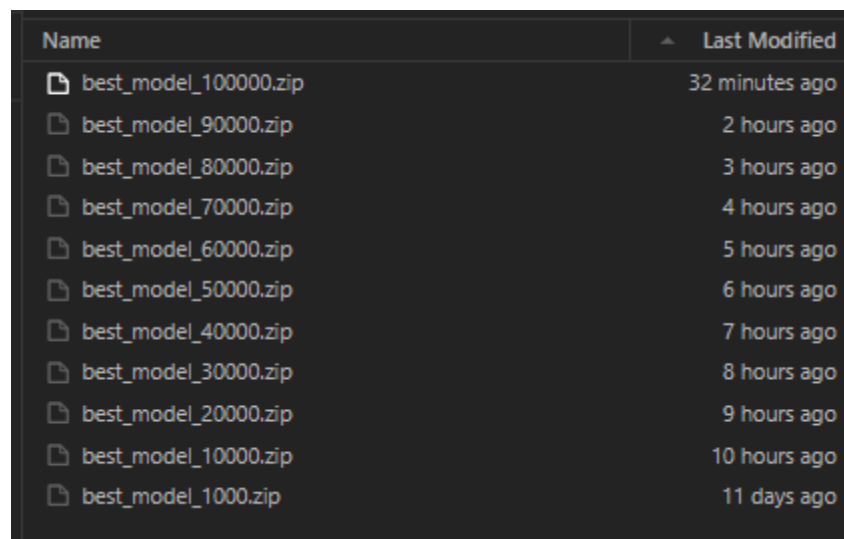


Figure 4 - Execution Output of 30,000 Trained Model

After initially training the model and making it execute 30000 times, the above was the rewards for 4 episodes. The maximum score is 75 which is a significant improve from the previous 11-16 reward range. For the final model we aimed to train 100,000 times to get a more accurate model.

## Model Trained with 100,000 Timesteps

We trained the model 100,000 times and saved an instance of the model every 10,000 executions. The training took about 10 hours.



Name	Last Modified
best_model_100000.zip	32 minutes ago
best_model_90000.zip	2 hours ago
best_model_80000.zip	3 hours ago
best_model_70000.zip	4 hours ago
best_model_60000.zip	5 hours ago
best_model_50000.zip	6 hours ago
best_model_40000.zip	7 hours ago
best_model_30000.zip	8 hours ago
best_model_20000.zip	9 hours ago
best_model_10000.zip	10 hours ago
best_model_1000.zip	11 days ago

*Figure 5 - Train Datasets after Training with 100,000 timesteps*

Initially we thought that best\_model\_100000 would be the most effective due to the extensive traintime and the size of data that would be available however the performance was not as expected.

Below are screenshots of 10 trial runs for each of the models and the average score for each.

```
[34]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_100000')
[34]: <stable_baselines3.dqn.dqn.DQN at 0x1fcd39a9e90>

[35]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 40
Total Reward for episode 2 is 131
Total Reward for episode 3 is 32
Total Reward for episode 4 is 107
Total Reward for episode 5 is 91
Total Reward for episode 6 is 38
Total Reward for episode 7 is 45
Total Reward for episode 8 is 26
Total Reward for episode 9 is 41
Total Reward for episode 10 is 25
```

Figure 6 - Test 100,000 Model

```
[36]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_90000')
[36]: <stable_baselines3.dqn.dqn.DQN at 0x1fcd2cda210>

[37]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 99
Total Reward for episode 2 is 137
Total Reward for episode 3 is 114
Total Reward for episode 4 is 108
Total Reward for episode 5 is 60
Total Reward for episode 6 is 263
Total Reward for episode 7 is 110
Total Reward for episode 8 is 226
Total Reward for episode 9 is 94
Total Reward for episode 10 is 75
```

Figure 7 - Test 90,000 Model

```
[38]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_80000')
[38]: <stable_baselines3.dqn.dqn.DQN at 0x1fcd389f990>

[39]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 36
Total Reward for episode 2 is 82
Total Reward for episode 3 is 24
Total Reward for episode 4 is 173
Total Reward for episode 5 is 83
Total Reward for episode 6 is 43
Total Reward for episode 7 is 27
Total Reward for episode 8 is 54
Total Reward for episode 9 is 141
Total Reward for episode 10 is 73
```

Figure 8 - Test 80,000 Model

```
[40]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_70000')
[40]: <stable_baselines3.dqn.dqn.DQN at 0x1fcc3961e90>
[41]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 201
Total Reward for episode 2 is 92
Total Reward for episode 3 is 81
Total Reward for episode 4 is 45
Total Reward for episode 5 is 58
Total Reward for episode 6 is 85
Total Reward for episode 7 is 25
Total Reward for episode 8 is 86
Total Reward for episode 9 is 35
Total Reward for episode 10 is 40
```

Figure 9 - Test 70,000 Model

```
[42]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_60000')
[42]: <stable_baselines3.dqn.dqn.DQN at 0x1fcc4fb0390>
[43]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 26
Total Reward for episode 2 is 32
Total Reward for episode 3 is 72
Total Reward for episode 4 is 80
Total Reward for episode 5 is 25
Total Reward for episode 6 is 24
Total Reward for episode 7 is 163
Total Reward for episode 8 is 202
Total Reward for episode 9 is 86
Total Reward for episode 10 is 122
```

Figure 10 - Test 60,000 Model

```
[45]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_50000')
[45]: <stable_baselines3.dqn.dqn.DQN at 0x1fcc45a6b10>
[46]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 226
Total Reward for episode 2 is 64
Total Reward for episode 3 is 179
Total Reward for episode 4 is 83
Total Reward for episode 5 is 28
Total Reward for episode 6 is 182
Total Reward for episode 7 is 55
Total Reward for episode 8 is 121
Total Reward for episode 9 is 109
Total Reward for episode 10 is 68
```

Figure 11 - Test 50,000 Model

```
[48]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_40000')
[48]: <stable_baselines3.dqn.dqn.DQN at 0x1fcc44ea3d0>
[49]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 83
Total Reward for episode 2 is 48
Total Reward for episode 3 is 86
Total Reward for episode 4 is 93
Total Reward for episode 5 is 57
Total Reward for episode 6 is 67
Total Reward for episode 7 is 181
Total Reward for episode 8 is 68
Total Reward for episode 9 is 74
Total Reward for episode 10 is 49
```

Figure 12 - Test 40,000 Model

```
[50]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_30000')
[50]: <stable_baselines3.dqn.dqn.DQN at 0x1fcc470ca50>
[51]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 96
Total Reward for episode 2 is 146
Total Reward for episode 3 is 65
Total Reward for episode 4 is 77
Total Reward for episode 5 is 74
Total Reward for episode 6 is 26
Total Reward for episode 7 is 44
Total Reward for episode 8 is 34
Total Reward for episode 9 is 27
Total Reward for episode 10 is 83
```

Figure 13 - Test 30,000 Model

```
[52]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_20000')
[52]: <stable_baselines3.dqn.dqn.DQN at 0x1fcc4729a10>
[53]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 74
Total Reward for episode 2 is 27
Total Reward for episode 3 is 57
Total Reward for episode 4 is 32
Total Reward for episode 5 is 82
Total Reward for episode 6 is 68
Total Reward for episode 7 is 185
Total Reward for episode 8 is 27
Total Reward for episode 9 is 78
Total Reward for episode 10 is 54
```

Figure 14 - Test 20,000 Model



```
[54]: model.load('C:\\Users\\Administrator\\Documents\\UOWD BCS CYBERSECURITY\\Year 4\\UOWD Sem 1 Autumn\\CSCI218\\project_backup\\DINO1\\train\\best_model_10000')

[54]: <stable_baselines3.dqn.dqn.DQN at 0x1fcc475a8d0>

[55]: #FINAL
for episode in range(10):
    observation = env.reset()
    done = False
    total_reward = 0
    while not done:
        action, _ = model.predict(observation)
        observation, reward, done, info = env.step(int(action))
        total_reward += reward
    print('Total Reward for episode {} is {}'.format(episode+1, total_reward))
    time.sleep(1)

Total Reward for episode 1 is 78
Total Reward for episode 2 is 98
Total Reward for episode 3 is 23
Total Reward for episode 4 is 35
Total Reward for episode 5 is 39
Total Reward for episode 6 is 35
Total Reward for episode 7 is 26
Total Reward for episode 8 is 64
Total Reward for episode 9 is 52
Total Reward for episode 10 is 145
```

Figure 15 - Test 10,000 Model

## Integration with Chrome Dino Game

The integration of the model with the Chrome Dino game involved the utilization of the pydirectinput library for simulating keypresses, specifically emulating the space bar and downward arrow key inputs. Simultaneously, the mss library was employed to capture and process screenshots of the game window. The captured image was converted into a suitable format for input into the reinforcement learning model. Initially, the screenshot was transformed into a NumPy array, retaining solely the first three color channels (BGR). Subsequently, the image underwent a conversion to grayscale. The resulting 2D array was reshaped into a 3D array, representing a single-channel grayscale image, which was then passed as input to the convolutional neural network (CNN) component of the model. This facilitated real-time interaction between the model and the Chrome Dino game, enabling the model to make decisions based on the processed game screen observations.

## Testing Methodology

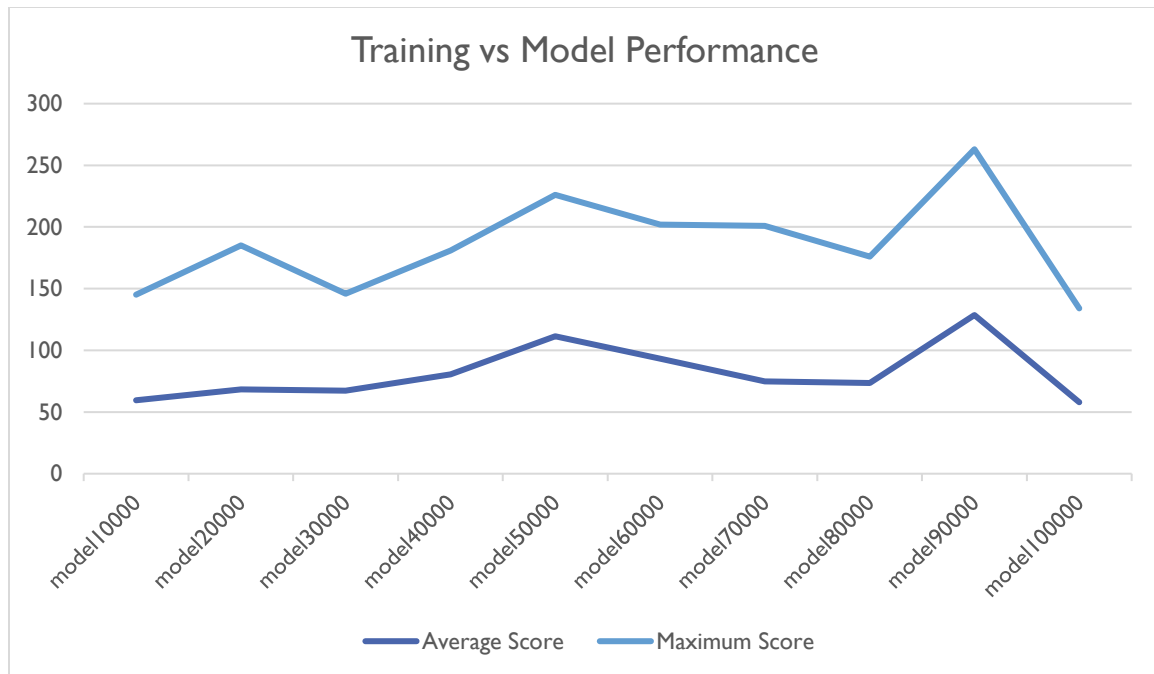
The model's performance was tested by running it on the Chrome Dino game for 10 episodes. The metrics we used to analyze the performance were the maximum and average score. Considering this model was trained 100 000 times, we were realistic with our expectations and did not expect to see scores in the 1000s range and kept our ideal score to be in the range of 300-400.

## Performance Metrics

As shown in the testing screenshots, 10 models were generated with the only difference being the number of executions. Starting at 10 000, going up to 100 000 in increments of 10 000, 10 models were created. Below is a table which shows the average score per run and the highest score achieved.

The model with the highest average score and the maximum score was the model90000 with the worst model being the model100000.

Model Name	Average Score (After 10 Executions)	Maximum Score (After 10 Executions)
model10000	59.5	145
model20000	68.4	185
model30000	67.2	146
model40000	80.6	181
model50000	111.5	226
model60000	93.2	202
model70000	74.8	201
model80000	73.6	176
model90000	128.6	263
model100000	57.9	134



## Encountered Issues

Several challenges were encountered, particularly surrounding training the model. We encountered an issue with the size of the game pixel range, as it would appear to be squished due to the rescaling. Through trial and error, we found the correct pixel size.

The next issue was the most significant one and was the lengthy training time. 100 000 steps completed after around 10 hours, meaning any training needed to be done overnight, and so any mistake would lead to valuable time being wasted. There were issues encountered with applications starting in the middle of the training and the model training pausing, which was mitigated by making sure all applications were turned off and closing the lid of the laptop to prevent any accidental keystrokes which would open an application or move the cursor.

## Experimental Adjustments

### Expanding the Dino's Field of View

We attempted to widen the observation space to grant the Dino more time to react to incoming obstacles. Adjusting the width was our focus yet altering the ratio of screen capture dimensions proved challenging. Consequently, we extended the width of the capture, leading to a quicker

reaction from the trained Dino due to the squeezed observation space, rendering this modification ineffective.

## **Mitigating Random Actions**

In a bid to reduce unnecessary actions, particularly random jumps, or crouches in empty areas, assigning a higher reward (2) to the no-operation action seemed promising. However, this adjustment inadvertently minimized other necessary actions, causing the Dino to repeatedly fail by colliding with obstacles.

## **Challenges with Overfitting**

Overfitting in reinforcement learning occurs when a model becomes too specialized in its training data, hindering its ability to generalize to new scenarios. Although we trained a model for 100,000 timesteps, it surprisingly performed poorer than a model trained for 90,000 timesteps. The latter model exhibited better adaptability and performance, showcasing the risk of overfitting in our training process.

## **Limitations**

Such a model takes up computing resources and time in order to train it adequately and has to work within the constraints of processing speed and memory of a computer and the model has to be in the same directory as the executable file to ensure the optimal execution of the Dino Bot Reinforcement Learning Agent.

## **Conclusion**

The project successfully demonstrates the application of reinforcement learning in a game environment. Future work can focus on refining the model for even better performance and exploring its application in more complex gaming scenarios.