Comp 7720-001
Artificial Intelligence
Demon Attack

Akhila Bachu(U00884957)
Sainath Reddy(U00865655)

Table of Contents

Chapter 1. INTRODUCTION

Demon Attack AI project is an initiative to develop an artificial intelligence system that can play the classic video game Demon Attack with a high level of proficiency. The goal of this project is to develop an AI agent that can not only compete with human players but also surpass them in terms of performance and strategy.

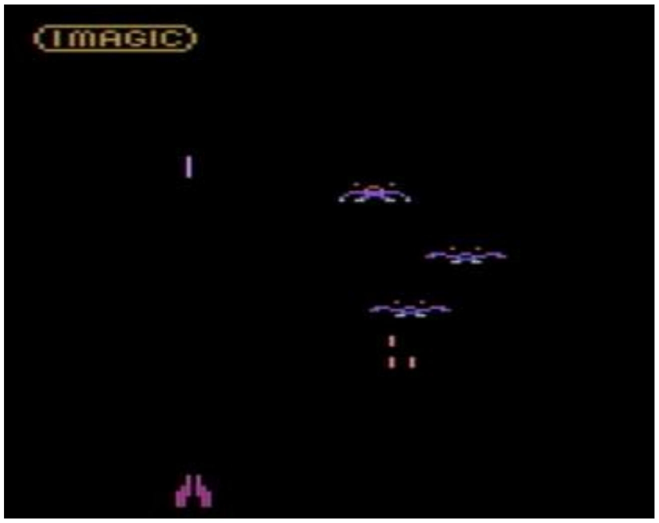


Fig: Demon Attack Link

There are several reasons why this project is worth pursuing. Firstly, Demon Attack is a challenging and complex game that requires a high level of skill and strategy to master. Developing an AI system that can play the game at a high level would be a significant achievement in the field of artificial intelligence and video game research.

The development of an AI system that can play Demon Attack could have practical applications in the development of autonomous systems and robotics. The ability to navigate and interact with complex environments and make decisions in real-time is a crucial aspect of many autonomous systems, and developing an AI system that can play Demon Attack could help researchers and engineers to better understand and develop these systems.

The Demon Attack AI project is a worthwhile initiative that could have significant implications for the fields of artificial intelligence, robotics, and entertainment.

Motivation:

Motivation of Developing the Demon Attack using artificial intelligence (AI). One of the main reasons is to advance the field of AI and demonstrate its ability to solve complex problems in the domain of video games. Demon Attack is a challenging game that requires a high level of strategy and decision-making to succeed. Developing an AI system that can play the game at a high level would be a significant achievement and could help researchers and developers to better understand and improve AI algorithms.

Contributions to the work:

- The development of an AI-based Demon Attack game using TensorFlow provides a challenging and engaging gaming experience for players.
- The use of deep learning and reinforcement learning algorithms allows the agent to learn from large datasets of gameplay experiences and make informed decisions in real-time.
- The research contributes to the advancement of AI, gaming, and robotics by providing a benchmark for evaluating the effectiveness of various AI algorithms and techniques.
- The project's exploration of transfer learning and ensemble methods has the potential to improve the agent's performance and efficiency.
- The development of an intelligent agent capable of playing Demon Attack at a high level can inspire the development of more advanced gaming platforms and autonomous systems.
- The project's findings can provide valuable insights into the design and implementation of AI-based gaming systems and contribute to the refinement and improvement of existing algorithms and techniques.

Chapter 2. RELATED WORK

There have been several related works to Demon Attack game using artificial intelligence. For instance, researchers at OpenAI used reinforcement learning techniques to train an AI agent to play a variety of Atari games, including Demon Attack. The AI agent was able to achieve superhuman performance on Demon Attack and other games, surpassing the performance of even the best human players.

Another related work is a project called MarLo, which uses reinforcement learning to teach AI agents to play the popular video game Minecraft. The project aims to develop AI agents that can navigate complex environments, solve problems, and work together to achieve goals.

Novelties of the project:

- The research project employs Tensor Flow, a state-of-the-art machine learning framework, to develop and train complex AI algorithms for playing Demon Attack.
- A deep reinforcement learning model is being developed to learn to play Demon Attack from scratch, without prior knowledge or handcrafted features.
- Transfer learning is being explored to bootstrap the learning process and improve the agent's performance.
- The project investigates ensemble methods, where multiple models are combined to make more accurate predictions and decisions.
- These novelties make the project an exciting and challenging endeavor with great potential for advancing the state-of-the-art in AI and gaming research.

Chapter 3. RL ENVIRONMENT

Reinforcement Learning

Reinforcement learning is a subfield of machine learning that involves training an artificial agent to learn and make decisions by interacting with an environment. In reinforcement learning, an agent receives feedback in the form of rewards or penalties based on its actions in the environment. The goal of the agent is to learn a policy that maximizes its expected long-term reward.

Reinforcement learning algorithms use trial-and-error learning to improve the agent's policy over time. The agent explores the environment and learns from the feedback it receives, adjusting its policy accordingly. The feedback it receives is used to update the value function, which estimates the expected long-term reward for a given state-action pair.

There are two main approaches to reinforcement learning:

- model-based
- model-free

Model-based approaches involve learning a model of the environment, such as a transition function or a reward function, and using this model to plan the agent's actions.

Model-free approaches, on the other hand, do not use a model of the environment and instead learn the policy directly from the feedback it receives.

Libraries

gym library:

The gym library is a Python toolkit for developing and comparing reinforcement learning algorithms. It provides a standardized set of environments for reinforcement learning experiments, making it easy to evaluate and compare the performance of different algorithms.

It provides a suite of tools for evaluating the performance of reinforcement learning algorithms. These tools include metrics for tracking the agent's progress, visualization tools for monitoring the agent's behavior, and benchmarking tools for comparing the performance of different algorithms. The gym library is widely used in the research community and has become a standard tool for evaluating and comparing reinforcement learning algorithms. It provides a simple and standardized interface for working with reinforcement learning environments, making it easy to get started with reinforcement learning research.

tensorflow:

TensorFlow is an open-source machine learning framework developed by Google. It is designed to make it easy to build, train, and deploy machine learning models. TensorFlow provides a variety of tools and libraries for working with a wide range of machine learning tasks, including deep learning, reinforcement learning, and computer vision.

TensorFlow has become a popular choice for machine learning researchers and practitioners due to its flexibility, ease of use, and wide range of applications. It has been used to build and train models for a wide range of tasks, including image recognition, natural language processing, and speech recognition, among others.

Convolution Neural Network:

Convolutional neural networks (CNNs) are a type of neural network commonly used for image and video processing tasks. CNNs are designed to recognize spatial patterns in images by using multiple convolutional layers.

There are three common layers used in deep learning models, particularly in convolutional neural networks (CNNs) for image processing tasks.

- Conv2D layer is used to apply a convolution operation to the input image. It takes as input a 3D tensor representing an image (height, width, channels), and applies a set of filters to the image to extract features. The output of a Conv2D layer is another 3D tensor representing the output feature maps.
- Dense layer is used for creating a fully connected neural network layer. It takes the flattened output from the previous layer (or input data) as input and applies a linear transformation to it, followed by a non-linear activation function. The output of a Dense layer is a tensor representing the probability of each class in a classification problem, or the predicted value in a regression problem.
- Flatten layer is used for flattening the output from a previous layer or input data into a 1D tensor. It takes the 3D tensor representing the feature maps from the previous Conv2D layer and flattens it into a 1D tensor, which can be fed into a Dense layer for classification or regression.

Together, these layers form the building blocks for a CNN model for image processing tasks. The Conv2D layer extracts features from the input image, the Flatten layer flattens the features, and the Dense layer applies a linear transformation to the flattened features to produce the final output.

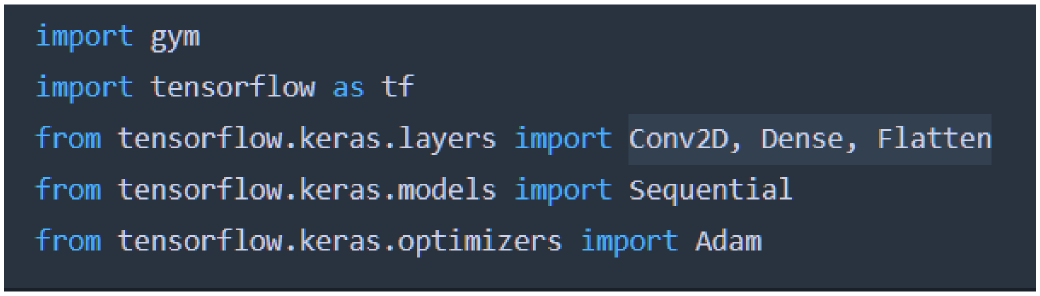
Sequential Model

The Sequential model class in Keras allows you to create a model by stacking layers in a linear fashion, where the output of one layer is fed as input to the next layer, and so on. The Sequential class provides a simple and intuitive way to build deep learning models, particularly for beginners who are new to deep learning.

Imports the Sequential model class from the Keras API within the TensorFlow library.

Adam Optimizer:

Optimizers are algorithms used to update the parameters (weights and biases) of a neural network during training. The Adam optimizer is a type of optimization algorithm that is commonly used for training deep neural networks.



```python
import gym
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
```

Chapter 4. METHODOLOGY

The implementation is carried on by two different researchers' individual roles are described below

The first researcher's role is to:

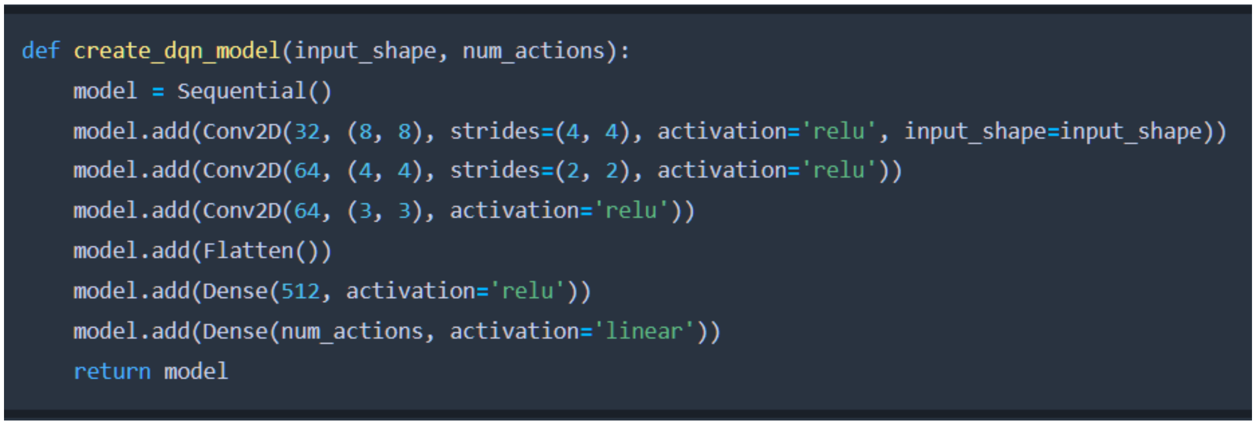
- Define and create the DQN model using the Keras API of TensorFlow.
- Compile the model with the Adam optimizer and mean squared error loss.
- Define hyperparameters for training, such as the number of episodes, maximum steps per episode, epsilon for epsilon-greedy exploration, and the replay buffer's size.
- Implement the main training loop, which consists of interacting with the environment, adding the transition to the replay buffer, and updating the model by sampling a batch from the replay buffer and computing the target and predicted Q-values using the DQN model.
- Decay epsilon for epsilon-greedy exploration and print the total reward for each episode.
- Test the model on the DemonAttack-v5 environment and record a video of the agent playing the game.

The second researcher's role involves:

- Installing dependencies required to record environment render
- Using the Recorder class from the colabgymrender package to record the environment and save it as a video
- Using the trained DQN model to choose actions in the DemonAttack-v5 environment
- Printing the chosen action at each step
- Updating the current state and checking if the episode is done
- Playing the recorded video of the game using the play() method from the Recorder class

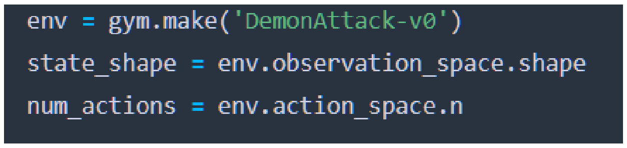
Code Implementation:

Model Building



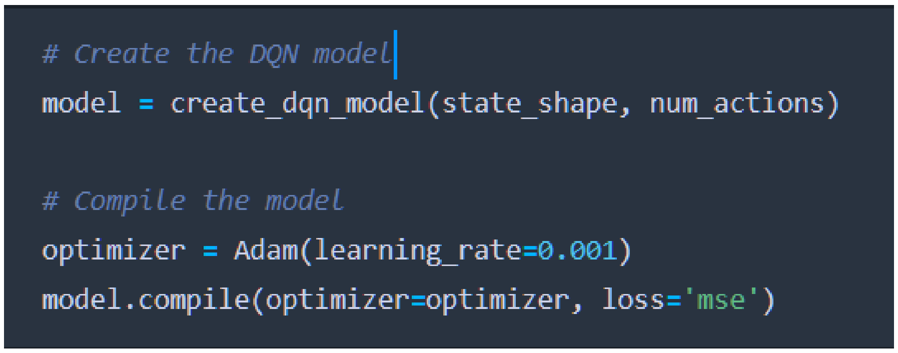
```
def create_dqn_model(input_shape, num_actions):
    model = Sequential()
    model.add(Conv2D(32, (8, 8), strides=(4, 4), activation='relu', input_shape=input_shape))
    model.add(Conv2D(64, (4, 4), strides=(2, 2), activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(num_actions, activation='linear'))
    return model
```

Define the DemonAttack-v0 Gym environment



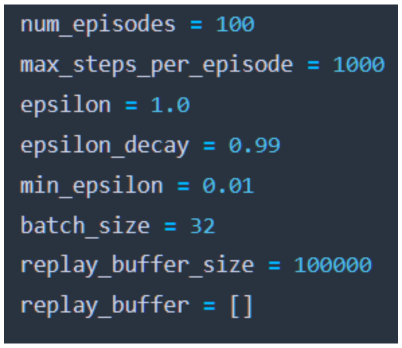
```
env = gym.make('DemonAttack-v0')
state_shape = env.observation_space.shape
num_actions = env.action_space.n
```

Creation and compilation of DQN model



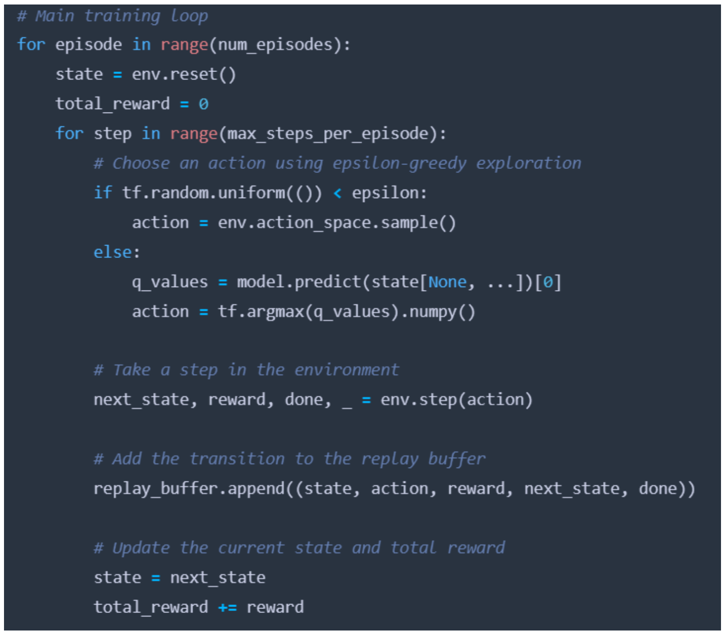
```
# Create the DQN model
model = create_dqn_model(state_shape, num_actions)

# Compile the model
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse')
```

Defining the hyperparameters for training of the model

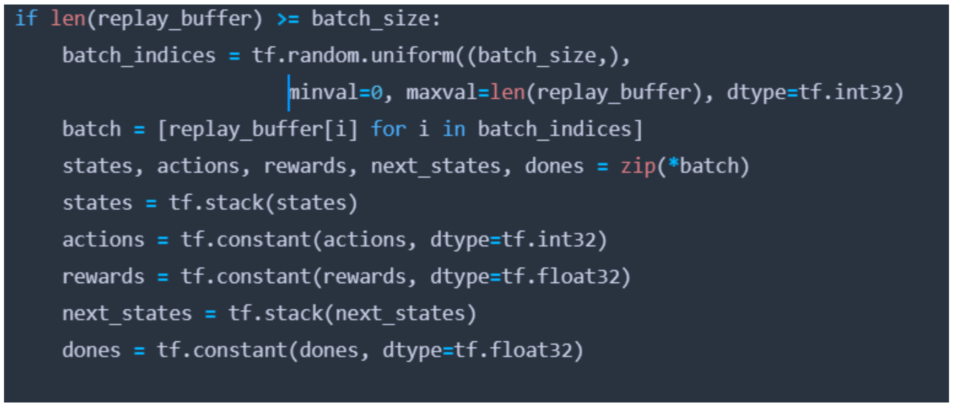


```
num_episodes = 100
max_steps_per_episode = 1000
epsilon = 1.0
epsilon_decay = 0.99
min_epsilon = 0.01
batch_size = 32
replay_buffer_size = 100000
replay_buffer = []
```

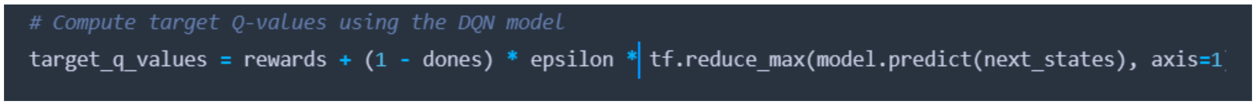
Training the model:



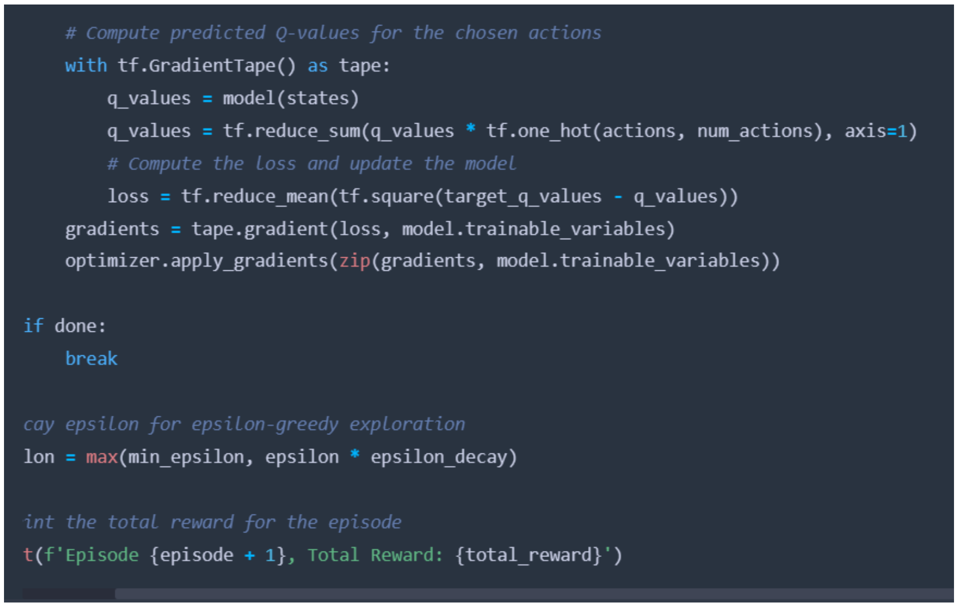
```python
# Main training loop
for episode in range(num_episodes):
    state = env.reset()
    total_reward = 0
    for step in range(max_steps_per_episode):
        # Choose an action using epsilon-greedy exploration
        if tf.random.uniform(()) < epsilon:
            action = env.action_space.sample()
        else:
            q_values = model.predict(state[None, ...])[0]
            action = tf.argmax(q_values).numpy()

        # Take a step in the environment
        next_state, reward, done, _ = env.step(action)

        # Add the transition to the replay buffer
        replay_buffer.append((state, action, reward, next_state, done))

        # Update the current state and total reward
        state = next_state
        total_reward += reward
```



```python
if len(replay_buffer) >= batch_size:
    batch_indices = tf.random.uniform((batch_size,),
                                      minval=0, maxval=len(replay_buffer), dtype=tf.int32)
    batch = [replay_buffer[i] for i in batch_indices]
    states, actions, rewards, next_states, dones = zip(*batch)
    states = tf.stack(states)
    actions = tf.constant(actions, dtype=tf.int32)
    rewards = tf.constant(rewards, dtype=tf.float32)
    next_states = tf.stack(next_states)
    dones = tf.constant(dones, dtype=tf.float32)
```

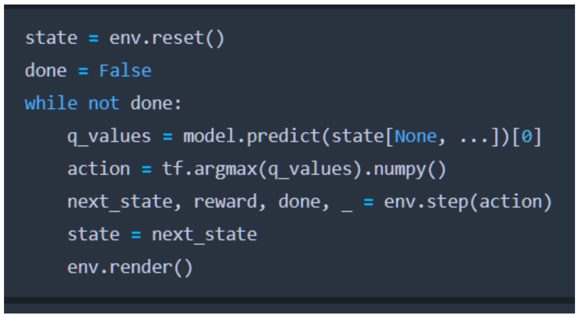


```python
 # Compute target Q-values using the DQN model
 target_q_values = rewards + (1 - dones) * epsilon * tf.reduce_max(model.predict(next_states), axis=1
```



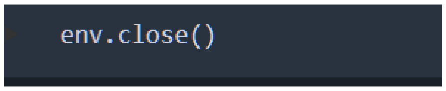
```
    # Compute predicted Q-values for the chosen actions
    with tf.GradientTape() as tape:
        q_values = model(states)
        q_values = tf.reduce_sum(q_values * tf.one_hot(actions, num_actions), axis=1)
        # Compute the loss and update the model
        loss = tf.reduce_mean(tf.square(target_q_values - q_values))
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

if done:
    break

cay epsilon for epsilon-greedy exploration
lon = max(min_epsilon, epsilon * epsilon_decay)

int the total reward for the episode
t(f'Episode {episode + 1}, Total Reward: {total_reward}')
```

Reset the environment



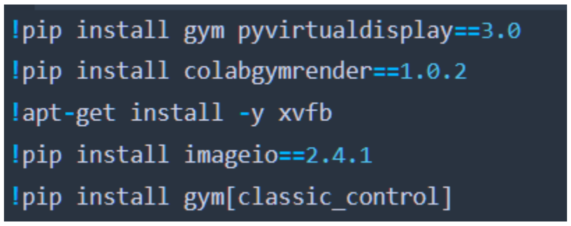
```
state = env.reset()
done = False
while not done:
    q_values = model.predict(state[None, ...])[0]
    action = tf.argmax(q_values).numpy()
    next_state, reward, done, _ = env.step(action)
    state = next_state
    env.render()
```

Close the environment:

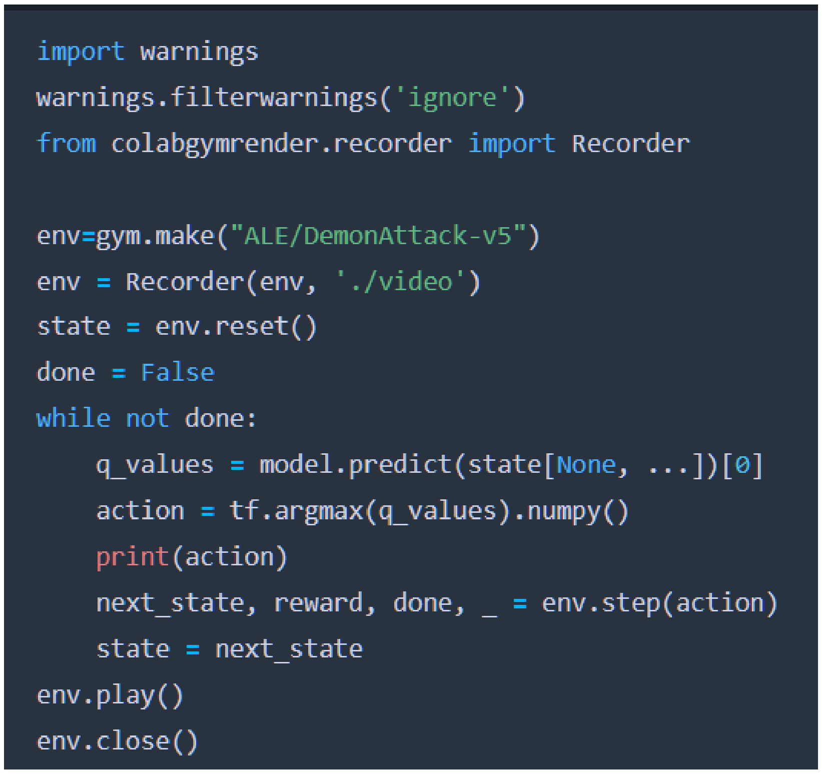


```
env.close()
```

Installing of some dependencies to record environment render



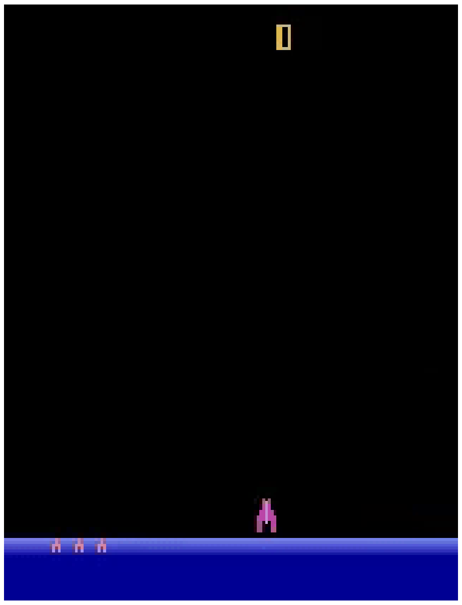
```
!pip install gym pyvirtualdisplay==3.0
!pip install colabgymrender==1.0.2
!apt-get install -y xvfb
!pip install imageio==2.4.1
!pip install gym[classic_control]
```



```python
import warnings
warnings.filterwarnings('ignore')
from colabgymrender.recorder import Recorder

env=gym.make("ALE/DemonAttack-v5")
env = Recorder(env, './video')
state = env.reset()
done = False
while not done:
    q_values = model.predict(state[None, ...])[0]
    action = tf.argmax(q_values).numpy()
    print(action)
    next_state, reward, done, _ = env.step(action)
    state = next_state
env.play()
env.close()
```

Chapter 5. EXPERIMENTS AND RESULTS

Results obtained on implementing the above code



Chapter 6. CONCLUSION

Demon Attack game can serve as a benchmark for evaluating the effectiveness of various AI algorithms and techniques. Developing an AI system that can achieve a high score in the Demon Attack game would be a significant achievement and could serve as a standard for evaluating and comparing the performance of different AI systems.

The use of TensorFlow to develop an artificial intelligence system capable of playing the Demon Attack game presents an exciting research area with significant potential for advancing the fields of AI, gaming, and robotics. Leveraging deep learning and reinforcement learning algorithms, through this project created an intelligent agent which is capable to play the game automatically.

Through the use of deep learning techniques, the agent can learn from large datasets of gameplay experiences and develop a robust understanding of the game's dynamics and challenges. Reinforcement learning algorithms enabled the agent to make informed decisions and adapt to new challenges.

Chapter 7. FUTURE WORK

Developing Demon Attack using AI can have educational value. This project can be used as a teaching tool to help young learners to better understand artificial intelligence and machine learning concepts. It can also be used as a platform for researchers and developers to experiment with new AI techniques and algorithms and share their findings with the wider community.

The development of an AI-based Demon Attack game using TensorFlow presents an exciting research area with significant potential for advancing the fields of AI, gaming, and robotics. The integration of deep learning and reinforcement learning techniques can enable the creation of intelligent agents capable of achieving high scores and providing human players with a challenging and engaging gaming experience.

With respect to the project, create a challenging opponent for human players. Human players can become bored or disinterested when playing against opponents that are too easy or too difficult. By creating an AI opponent that can adapt to the player's skill level and provide a challenging and engaging experience, the game can become more entertaining and enjoyable for players.

References:

[1] https://www.retrogames.cz/play_081-Atari2600.php?language=EN

[2] Robert Moni, “Reinforcement Learning algorithms — an intuitive overview”, https://smartlabai.medium.com/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc

[3] OpenAI SPinning up, “Part 2: Kinds of RL Algorithms”, https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

[4] Mohit Pandey, “Top Reinforcement Learning Algorithms”, https://analyticsindiamag.com/top-reinforcement-learning-algorithms/

[5] Chao De-Yu, “Deep Q-Network, with PyTorch”, https://towardsdatascience.com/deep-q-network-with-pytorch-146bfa939dfe