201241037, T.Coupe@student.liverpool.ac.uk, Thomas Coupe

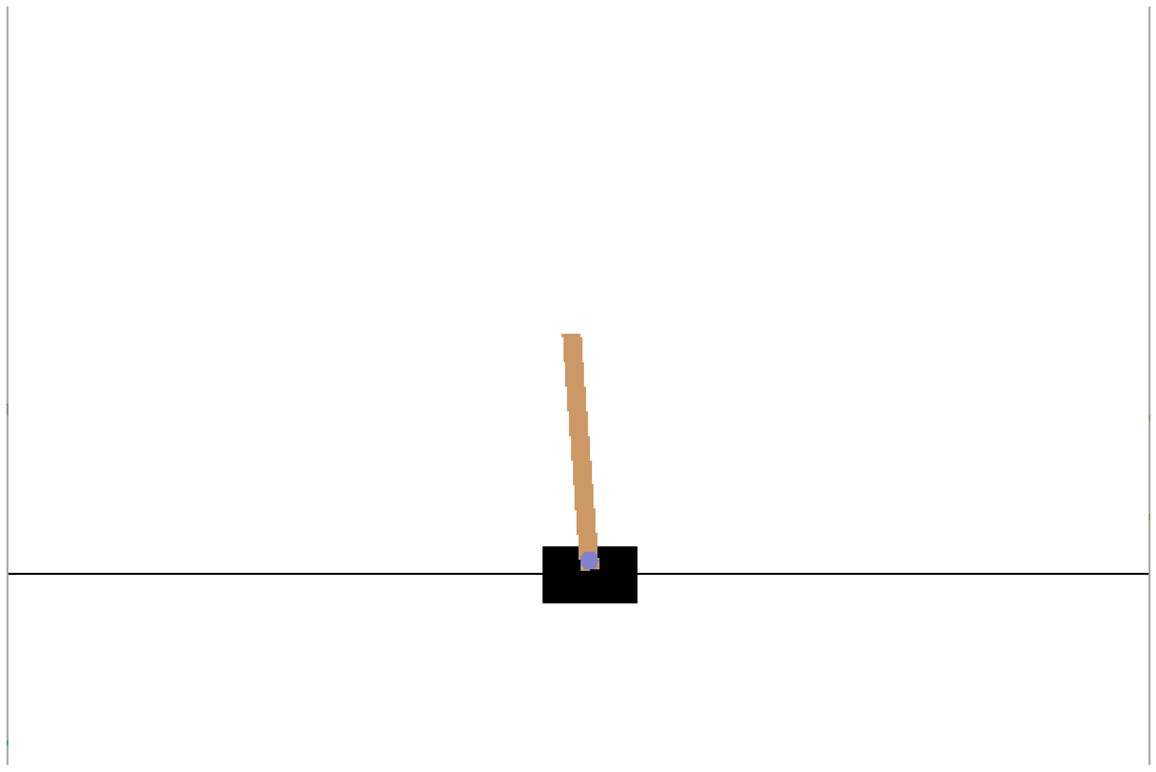
Assignment 02 – Advanced AI

Task checklist:

- Imported a game from the OpenAI gym library
- Created a neural network
- Connected the gym game enviorment to the neural network
- Implemented a deep reinforcement learning model (Deep Q Learning)
- Experimental results

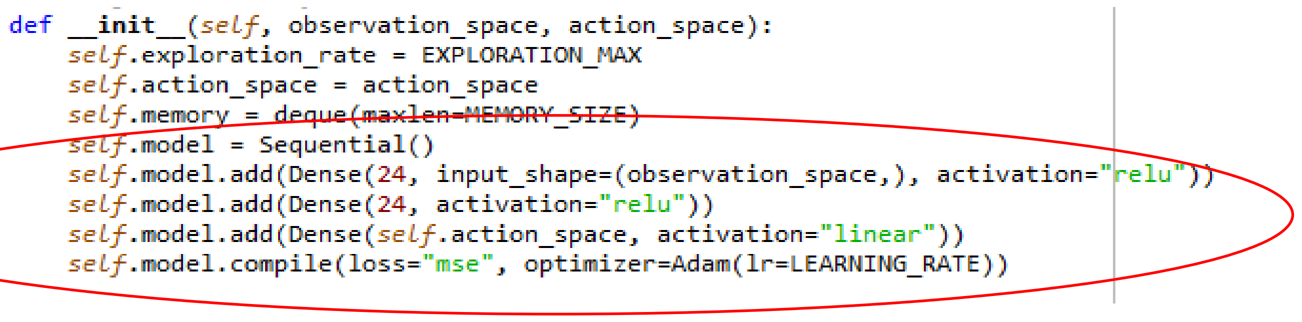
Importing Open AI gym game

Firstly we needed a game from the OpenAI gym library to apply a reinforcement leaning model to. The game that I chose for this task was "CartPole_v1" from the gym library. In this game it involves a vertical pole sitting on top of a cart. The goal is to move the cart from the centre to the edge without the pole tilting more than 15 degrees either way. By using a reinforcement model we can teach the agent to move the cart correctly without knocking over the pole.



Creating a Neural Network

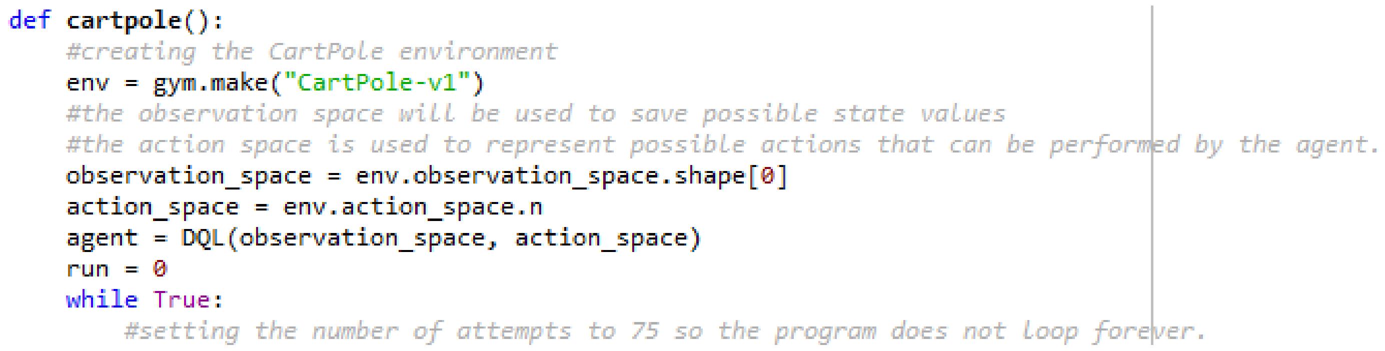
When creating the neural network we first declared out net architecture under the _init_ method within the DQL class. This is where the structure and parameters of the neural net were defined.



```
def __init__(self, observation_space, action_space):
    self.exploration_rate = EXPLORATION_MAX
    self.action_space = action_space
    self.memory = deque(maxlen=MEMORY_SIZE)
    self.model = Sequential()
    self.model.add(Dense(24, input_shape=(observation_space,), activation="relu"))
    self.model.add(Dense(24, activation="relu"))
    self.model.add(Dense(self.action_space, activation="linear"))
    self.model.compile(loss="mse", optimizer=Adam(lr=LEARNING_RATE))
```

By using the keras library we can set the model of our network to whatever we want. In the highlighted section you can see I have set the model to a sequential model. The sequential model is a linear stack of layers, it needs to know what input shape it should be expecting and it needs to know information about its input shape. As you can see I have set the input shape to the observation_space as that will be the input given by the agent. I have also set the activation to "relu", relu stands for Rectified Linear Unit.

Connecting the game to the neural network



```
def cartpole():
    #creating the CartPole environment
    env = gym.make("CartPole-v1")
    #the observation space will be used to save possible state values
    #the action space is used to represent possible actions that can be performed by the agent.
    observation_space = env.observation_space.shape[0]
    action_space = env.action_space.n
    agent = DQL(observation_space, action_space)
    run = 0
    while True:
        #setting the number of attempts to 75 so the program does not loop forever.
```

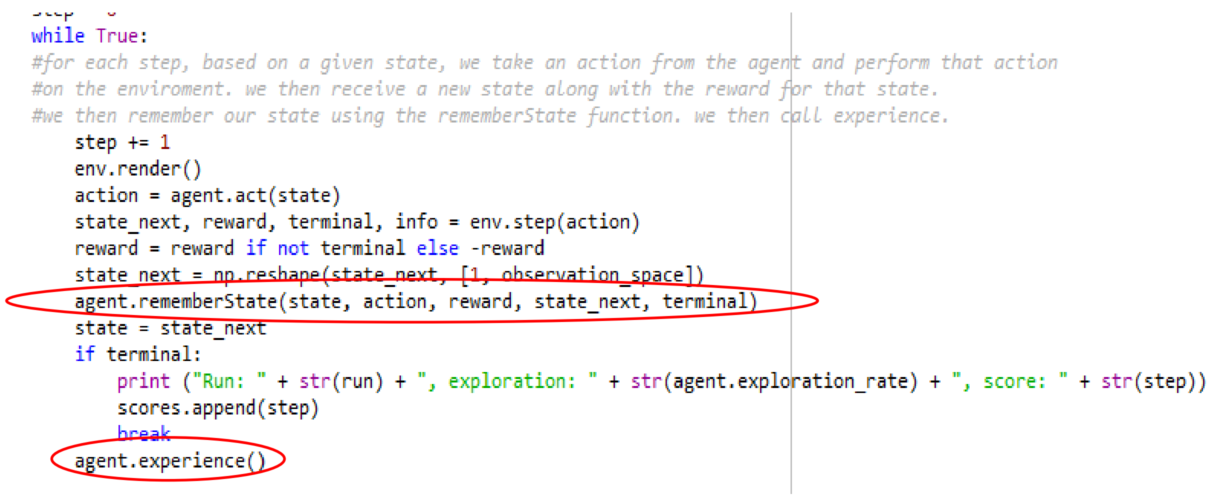
Our cartpole method is where we connect the game to the neural network. We do this by interacting with the observation space and the action space. The observation space is what used to stored observations of states made by the agent. With the values from the observation space, we can then use our action space to make possible actions that our agent will perform.

201241037, T.Coupe@student.liverpool.ac.uk, Thomas Coupe

Deep Reinforcement learning model

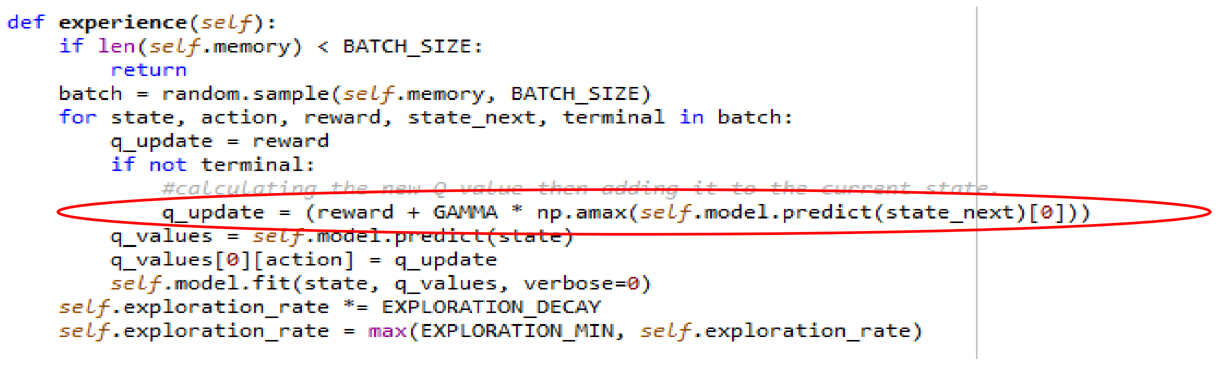
For this task the reinforcement learning model that I implemented was the Deep Q Learning model. Deep Q Learning works by choosing the best action for a given observation. Each possible action for each possible observation has a given Q value, a Q value represents the quality of a given move performed by an agent. To ensure that we end up with accurate Q values we need to interact with our neural network along with some linear algebra.

This while loop loops throughout each step the agent takes until the agent either fails or



```
while True:
#for each step, based on a given state, we take an action from the agent and perform that action
#on the enviroment. we then receive a new state along with the reward for that state.
#we then remember our state using the rememberState function. we then call experience.
    step += 1
    env.render()
    action = agent.act(state)
    state_next, reward, terminal, info = env.step(action)
    reward = reward if not terminal else -reward
    state_next = np.reshape(state_next, [1, observation_space])
    agent.rememberState(state, action, reward, state_next, terminal)
    state = state_next
    if terminal:
        print ("Run: " + str(run) + ", exploration: " + str(agent.exploration_rate) + ", score: " + str(step))
        scores.append(step)
        break
    agent.experience()
```

completes the task. For each state that our agent experiences it is important that we remember it. This is where the rememberState method is called, it is used so that our agent can remember the previous states. Once the agent either fails or completes the task we call the experience method. The experience method is used to sample experiences from the memory and for each entry to update the Q value.



```
def experience(self):
    if len(self.memory) < BATCH_SIZE:
        return
    batch = random.sample(self.memory, BATCH_SIZE)
    for state, action, reward, state_next, terminal in batch:
        q_update = reward
        if not terminal:
            #calculating the new Q value then adding it to the current state.
            q_update = (reward + GAMMA * np.amax(self.model.predict(state_next)[0]))
        q_values = self.model.predict(state)
        q_values[0][action] = q_update
        self.model.fit(state, q_values, verbose=0)
    self.exploration_rate *= EXPLORATION_DECAY
    self.exploration_rate = max(EXPLORATION_MIN, self.exploration_rate)
```

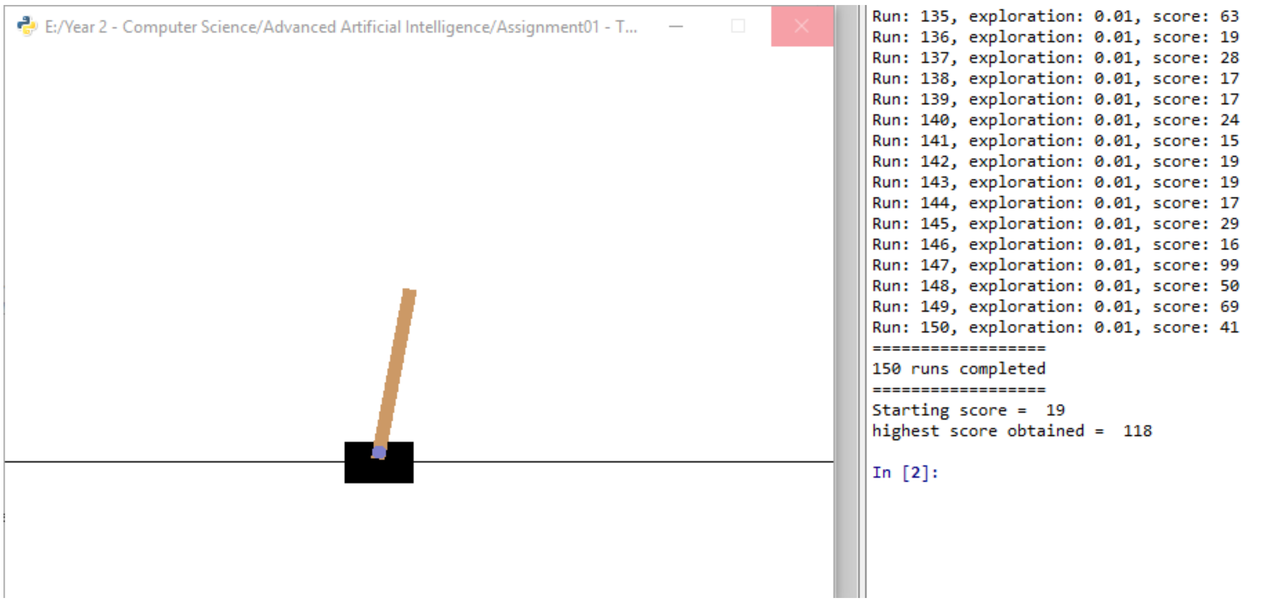
In the highlighted section of the experience method we are calculating the new Q value by taking the maximum Q for a given action and then multiplying it by the GAMMA variable which is the discount factor. Once we have done this we then add it to the current state reward.

201241037, T.Coupe@student.liverpool.ac.uk, Thomas Coupe

Experimental results

In the program the reliability of the learning model is represented in the terminal with a score. The score is increased based upon how slow and controlled the agent moves the cart. If the agent gets a low score it means that it was rushed and not controlled however if the agent gets a higher score it means it was much more controlled meaning that the learning model for that run was reliable.

First run:

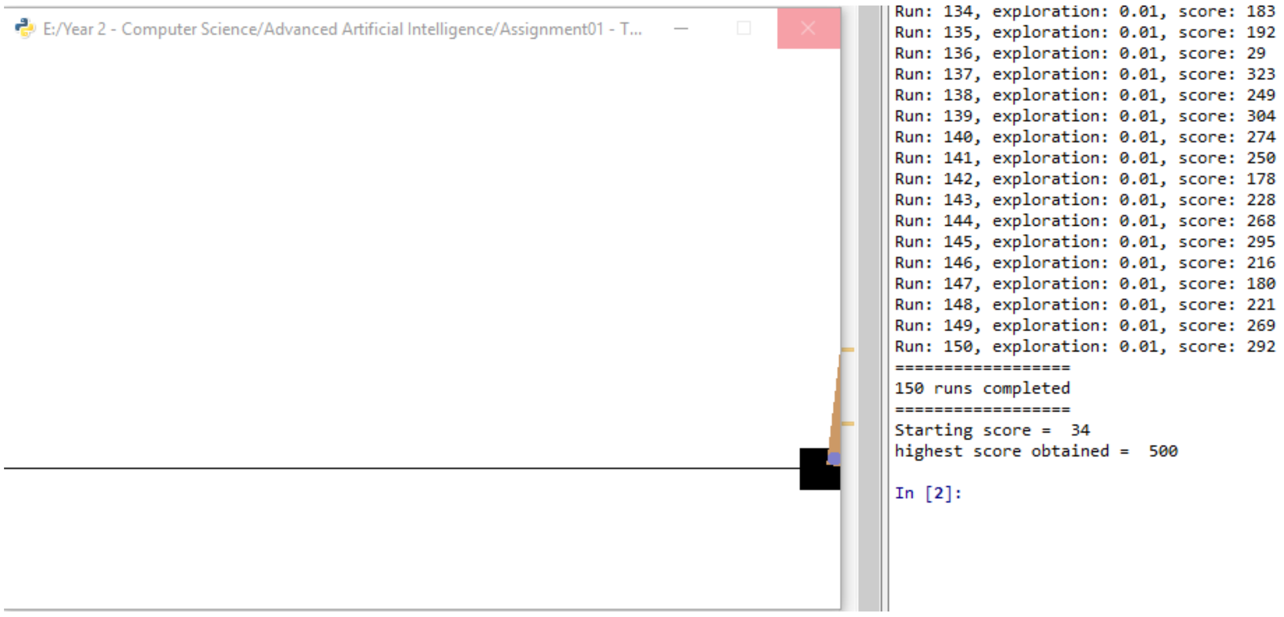

E:/Year 2 - Computer Science/Advanced Artificial Intelligence/Assignment01 - T...

```
Run: 135, exploration: 0.01, score: 63
Run: 136, exploration: 0.01, score: 19
Run: 137, exploration: 0.01, score: 28
Run: 138, exploration: 0.01, score: 17
Run: 139, exploration: 0.01, score: 17
Run: 140, exploration: 0.01, score: 24
Run: 141, exploration: 0.01, score: 15
Run: 142, exploration: 0.01, score: 19
Run: 143, exploration: 0.01, score: 19
Run: 144, exploration: 0.01, score: 17
Run: 145, exploration: 0.01, score: 29
Run: 146, exploration: 0.01, score: 16
Run: 147, exploration: 0.01, score: 99
Run: 148, exploration: 0.01, score: 50
Run: 149, exploration: 0.01, score: 69
Run: 150, exploration: 0.01, score: 41
==================
150 runs completed
==================
Starting score =  19
highest score obtained =  118

In [2]:
```

On the first run of the program there wasn't really a huge increase in score. The starting score of the agent was 19 and the highest score achieved was 118. This shows an increase in score however the agent did not manage to complete the task.

Second Run:


E:/Year 2 - Computer Science/Advanced Artificial Intelligence/Assignment01 - T...

```
Run: 134, exploration: 0.01, score: 183
Run: 135, exploration: 0.01, score: 192
Run: 136, exploration: 0.01, score: 29
Run: 137, exploration: 0.01, score: 323
Run: 138, exploration: 0.01, score: 249
Run: 139, exploration: 0.01, score: 304
Run: 140, exploration: 0.01, score: 274
Run: 141, exploration: 0.01, score: 250
Run: 142, exploration: 0.01, score: 178
Run: 143, exploration: 0.01, score: 228
Run: 144, exploration: 0.01, score: 268
Run: 145, exploration: 0.01, score: 295
Run: 146, exploration: 0.01, score: 216
Run: 147, exploration: 0.01, score: 180
Run: 148, exploration: 0.01, score: 221
Run: 149, exploration: 0.01, score: 269
Run: 150, exploration: 0.01, score: 292
==================
150 runs completed
==================
Starting score =  34
highest score obtained =  500

In [2]:
```

In the second run we had much more successful results as we started with a score of 34 and finished with a high score of 500. This shows a huge improvement in the learning of the agent.

Due to the huge differences in results from the agent I believe that this algorithm may have some problems. I don't think the Deep Q Learning algorithm is very reliable as the agent may achieve a score of 500 and then on the next run achieve a score of 25. This is a huge drop in performance and I believe that this is due to the Deep Q Learning model over-estimating some of the Q values which causes anomalies in the data. I think a more reliable model would be a Double Q Learning model as two values are used to measure and calculate the Q values instead of just one value. Overall though throughout the model there is a clear improvement in the agent's performance suggesting that Deep Q Learning can be a useful reinforcement learning model.