For the AI agent implementation, I used the Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems book (Reinforcement Learning Chapter), particularly the training step function and sample experiences are based on the implementation from the book official github repo that can be found here <https://github.com/ageron/handson-ml2>. However, in the book implementation, the deep learning model does not take an image as input to analyze the game and output the q-values. Rather, it uses simple state variables for a very simple gym built-in environment. I mentioned that I used this book clearly in my document. I used this tutorial in this link https://blog.paperspace.com/creating-custom-environments-openai-gym/ to learn how to make a custom environment. This tutorial is based on a very different game and my code implementation and game rule are completely different. The ShapedObject, AgetnSHip, EnemyShip, AgentMissile, and EnemyMissile are completely my own implementation.

# Libraries:

Many of the libraries are imported based on the book implementation. However, I use extra libraries for my game including shapely geometry library that I used for collision detection.

Import tensorflow and give it the alias name tf. Also import keras library. Tensorflow is used in implementing the gradient descent RMSprop variant and keras is used to define the structure of the deep learning model. Check that tensorflow can use the GPU since deep learning models requires gpu to run faster.

Text

Description automatically generated with medium confidence

Import numy library and give the alias np and import the os library



This is used to ensure that the



Import matplotlib that is used for plotting figures

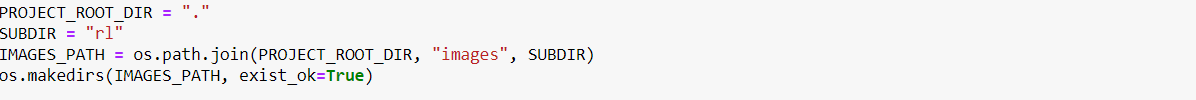
A picture containing shape

Description automatically generated

Import the deque which is used to implement the replay buffer in the AI agent



Set directory to save images. Not used in my game



Import libraries from the shapely library (affinity, point and polygon). Also import gym OpenAI library that contains the Env class.

A picture containing background pattern

Description automatically generated

# ShapeObject class:

Define the shaped object class which is a subclass of the class object



Define the init method of the class shaped object which is basically the constructor. It takes the object name as input and self is just to reference the object.



Initialize the x and y attributes to zero. These attributes correspond to the center of the object.



Initialize the angle to zero, the angle will define the direction of the object. The angle zero means that the direction is pointing to the right



Initialize the structure to have a single (0,0) point. The structure attribute stores the polygonal points representing the margin of the object and they are crucial to detect collision between 2 shaped objects by using the intersects function.



Initialize the name of the shaped object. This is important to distinguish between different object in my game (agent ship or enemy ship)



Define the set\_shape function. This function takes a list of 2D points representing the margin of the object. These points are stored in the structure attribute. The self in the argument list is to reference the object and is not used when invoking the method.



Define the rotate method. The input to this function is the amount of increment that we want to apply to the angle attribute. This increment is in degrees not radian. Again, self is to reference the object and is not used when invoking the method.



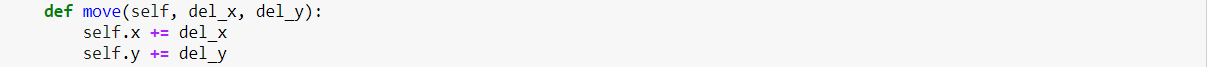
Apply the increment to the angle attribute.



Apply a rotation to the polygon points stored in the structure attribute based on the angle increment. This rotation will be around the center of the object.



Define the move method that will take as input the difference in the x and y attributes. It will add the differences to the x and y attributes. This version of the move function is not used in my code. I am using another version of the move method that will be discussed next.



Define another version of the move method, this method takes the object velocity as input. It then applies a translation to the polygon points and the center along the directional angle using the sine and cosine functions.



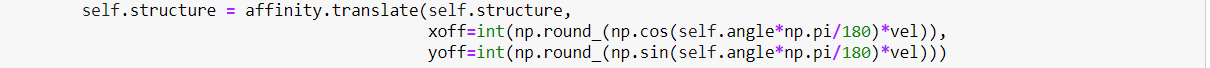
Update the attribute x value by adding the cosine of the angle attribute (transformed to radians) and multiply it with the velocity. The value is rounded to nearest integer since we are deadling with pixel positions.



Update the attribute y value by adding the sine of the angle attribute (transformed to radians) and multiply it with the velocity. The value is rounded to nearest integer since we are dealing with pixel positions.



Update the polygon points of the object by applying a translation operation. This operation will add the sine of the angle in radians multiplied by velocity to y values and the cosine of the angle multiplied by velocity to the x values of each point in the structure attribute. Also, these values are rounded to the nearest integer since we are dealing with pixel positions.



# AgentShip class:

Define the AgentShip class. This class is a subclass of the class ShapedObject.



Define the init method (constructor). This will take as input, the name of the object, the initial x and y position of the ship center, the initial angle, and the velocity which by default takes the value 0.



Invoke the super class init method and pass the “name” input variable.



Initialize the velocity attribute based on the input velocity.



Initialize the x and y attributes of the super class. These represents the center of the object relative to the polygon points.



Read the image file representing the agent ship and assign it to attribute icon. The pixel values are divided by 255 so that they take values between 0 and 1.



Set the side length of the minicanvas that will be containing the ship image to 200.



Set the width of the ship object to 40.



Set the height of the ship object to 80.



Define the top left position of the ship image relative to its containing minicanvas.



Resize the ship image based on the icon\_w and incon\_h attributes (80,40)



Define the minicanvas as a matrix of size 200x200x3 and set all the pixel values to one. So that the minicanvas represents a white image. I used the minicanvas as it will help when rotating the image in a larger white background image.



Copy the pixel values of the agentship image (stored in icon) to their corresponding locations in the minicanvas.



Store the final canvas after copying pixel values in the icon attribute.



Define the polygon points representing the AgentShip margins and store them in the structure attribute in the superclass using the set\_shape function.

A picture containing background pattern

Description automatically generated

Rotate both the polygon points and the image using the rotate method based on the angle attribute.



Define the speedup method which increments the velocity attribute by a factor of 1. The maximum velocity is 10.



Define the speedup method which decrements the velocity attribute by a factor of 1. The minimum velocity is 0.



Define the move method which invokes the move method in the superclass and pass the velocity attribute.



Define the rotate method which takes as input the change in angle in degrees.



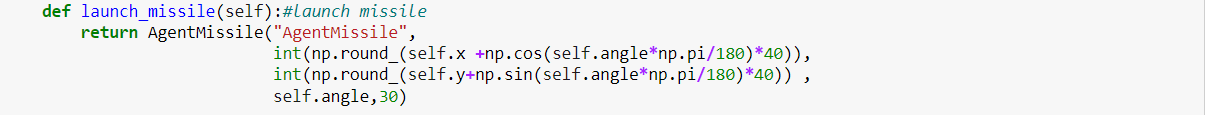
Invoke the rotate method in the super class which will rotate the polygonal points based on the change in angle. This will also update the angle attribute value in the super class.



Rotate the agentship image (represented in the icon attribute) by computing the rotation matrix and applying it to the image.



Define the launch missile method. This method returns a new AgentMissile object with name “AgentMissile”, positioned in front of the ship head (based on its directional angle). It will have the same angle as the current angle of the ship and a velocity of 30.



# EnemyShip class:

Define the EnemyShip class. This class is a subclass of the class ShapedObject.



Define the init method (constructor). This will take as input, the name of the object, the initial x and y position of the ship center, the initial angle, and the velocity which by default takes the value 0.



Invoke the super class init method and pass the “name” input variable.



Initialize the velocity attribute based on the input velocity.



Initialize the x and y attributes of the super class. These represents the center of the object relative to the polygon points.



Read the image file representing the enemy ship and assign it to attribute icon. The pixel values are divided by 255 so that they take values between 0 and 1.



Set the side length of the minicanvas that will be containing the ship image to 200.



Set the width of the ship object to 40.



Set the height of the ship object to 80.



Define the top left position of the ship image relative to its containing minicanvas.



Resize the ship image based on the icon\_w and incon\_h attributes (80,40)



Define the minicanvas as a matrix of size 200x200x3 and set all the pixel values to one. So that the minicanvas represents a white image. I used the minicanvas as it will help when rotating the image in a larger white background image.



Copy the pixel values of the enemy ship image (stored in icon) to their corresponding locations in the minicanvas.



Store the final canvas after copying pixel values in the icon attribute.



Define the polygon points representing the EnemyShip margins and store them in the structure attribute in the superclass using the set\_shape function.

A picture containing shape

Description automatically generated

Rotate both the polygon points and the image using the rotate method based on the angle attribute.



Define the speedup method which increments the velocity attribute by a factor of 1. The maximum velocity is 10.



Define the speedup method which decrements the velocity attribute by a factor of 1. The minimum velocity is 0.



Define the move method which invokes the move method in the superclass and pass the velocity attribute.



Define the rotate method which takes as input the change in angle in degrees.



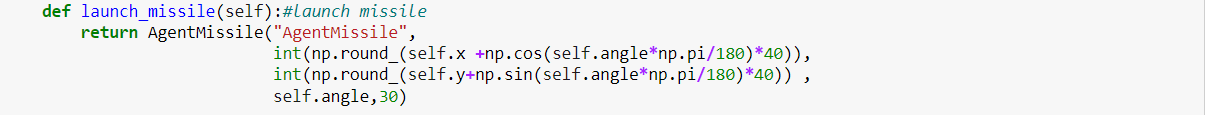
Invoke the rotate method in the super class which will rotate the polygonal points based on the change in angle. This will also update the angle attribute value in the super class.



Rotate the enemyship image (represented in the icon attribute) by computing the rotation matrix and applying it to the image.



Define the launch missile method. This method returns a new EnemyMissile object with name “EnemyMissile”, positioned in front of the ship head (based on its directional angle). It will have the same angle as the current angle of the ship and a velocity of 30.



# AgentMissile class:

Define the AgentMissile class. This class is a subclass of the class ShapedObject.



Define the init method (constructor). This will take as input, the name of the object, the initial x and y position of the ship center, the initial angle, and the velocity which by default takes the value 0.



Invoke the super class init method and pass the “name” input variable.



Initialize the velocity attribute based on the input velocity.



Initialize the x and y attributes of the super class. These represents the center of the object relative to the polygon points.



Read the image file representing the agent missile and assign it to attribute icon. The pixel values are divided by 255 so that they take values between 0 and 1.



Set the side length of the minicanvas that will be containing the ship image to 200.



Set the width of the ship object to 15.



Set the height of the ship object to 30.



Define the top left position of the ship image relative to its containing minicanvas.



Resize the ship image based on the icon\_w and incon\_h attributes (80,40)



Define the minicanvas as a matrix of size 200x200x3 and set all the pixel values to one. So that the minicanvas represents a white image. I used the minicanvas as it will help when rotating the image in a larger white background image.



Copy the pixel values of the enemy ship image (stored in icon) to their corresponding locations in the minicanvas.



Store the final canvas after copying pixel values in the icon attribute.



Define the polygon points representing the AgentSHip margins and store them in the structure attribute in the superclass using the set\_shape function.

A picture containing shape

Description automatically generated

Rotate both the polygon points and the image using the rotate method based on the angle attribute.



Define the speedup method which increments the velocity attribute by a factor of 1. The maximum velocity is 10.



Define the speedup method which decrements the velocity attribute by a factor of 1. The minimum velocity is 0.



Define the move method which invokes the move method in the superclass and pass the velocity attribute.



Define the rotate method which takes as input the change in angle in degrees.



Invoke the rotate method in the super class which will rotate the polygonal points based on the change in angle. This will also update the angle attribute value in the super class.



Rotate the agent missile image (represented in the icon attribute) by computing the rotation matrix and applying it to the image.



# EnemyMissile class:

Very similar to agent missile

# BattleShipGame class:

Define the BattleSHipGame class which is a subclass of gym.Env class.



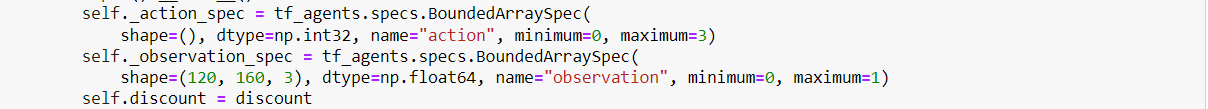
Define the constructor which takes the discount factor as input.



Invoke the super class init method.



These attributes are not used in the game



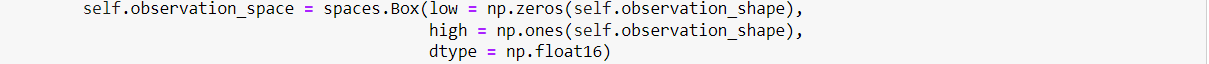
Define the shape of the observation (game canvas) which is 600x800x3 pixels.



Define the corner points to represent the margin of the game canvas to detect if any object is getting outside the game canvas.



Define the observation space which corresponds to the possible values of each pixel (between 0 and 1)



Define the action space which takes one of the values {0,1,2,3,4}

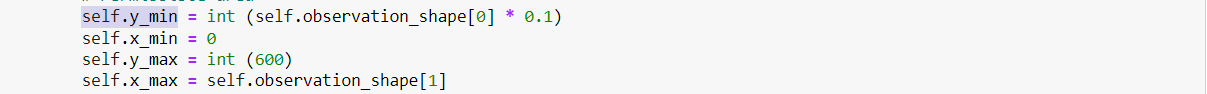


Initialize the game canvas as a white image of size 600x800x3

Initialize the elements list which contains the different game objects (agent ship, enemy ship, agent missile and enemy missile.



These attributes are not used in my game.



Define the draw\_elements\_on\_canvas method. This method draws all game objects in the game canvas.



Initialize the game canvas as a white image. All pixel values are equal to one.



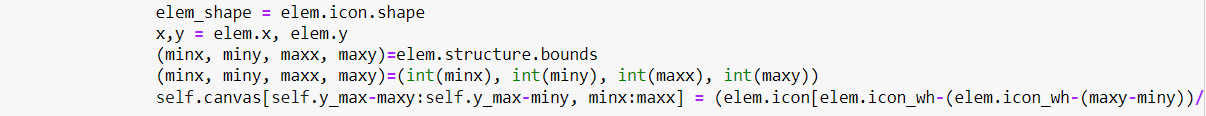
Loop over all elements (objects) in the game list.



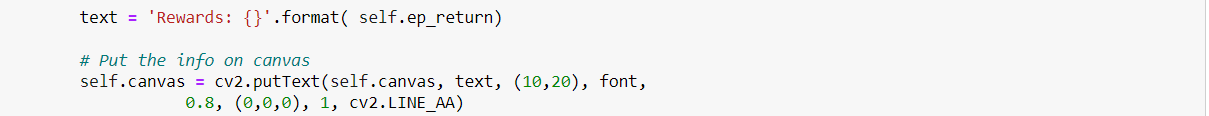
Check if the element is inside the game border using the shapely covers function



Copy the elements pixels to their appropriate positions in the game canvas.



Show the episodic accumulated reward on the top left of the game canvas.



Define the reset method. This method is used to reset the game to its initial state. It returns a reduced-size game canvas of size 120x160x3 pixels. This is to reduce the computations for deep learning training.



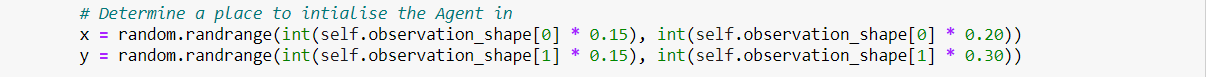
Ep\_return attribute represents the accumulated rewards in the current episode.



Number of enemy ships currently in the game canvas. Used to detect whether the enemy fleet was destroyed.



These variables are not used.



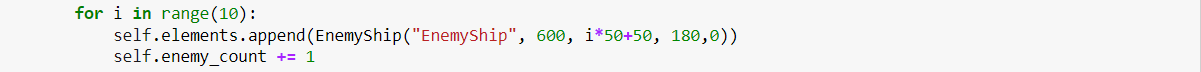
The ship attribute corresponds to the agent ship. We create a new agent ship object and place at position 200 200 in the canvas with an initial angle of 0 and an initial velocity of 2.



Add the agent ship to the game elements list



Create 10 enemy ships and add them to the game elements list using the append function. The enemy ships are positioned in the right side of the game canvas with a velocity of zero and angle of 180 degrees (pointing towards the left.



Initialize the canvas so that it has pixel values of 1 (white image).



Draw the game elements on the game canvas.



Return the observation as a reduced size image of the canvas (120x160x3 pixels). This is to reduce the computations of the deep learning model.



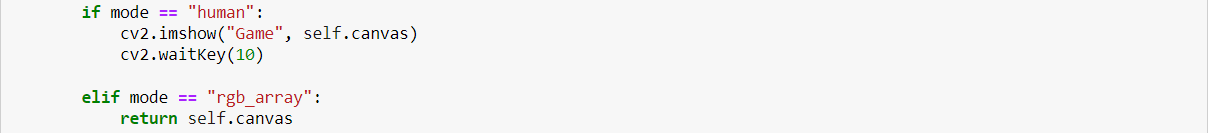
Define the render method. This method takes as input the mode of the rendering either “human” in which, the full-size (600x800x3 pixels) game canvas will be rendered in a separate window and update the scene each 10 milliseconds or “rgb\_array” in which the h=game canvas will be returned as an image.



Check that the input has valid values



“human” in which, the full-size (600x800x3 pixels) game canvas will be rendered in a separate window and update the scene each 10 milliseconds or “rgb\_array” in which the h=game canvas will be returned as an image.



Define the close method,, which closes all windows opened by the app.



This method gives the meaning of action values.



Define the step method. This function takes an action (the values of which is one from {0,1,2,3,4}) as input and outputs the new observation, the reward value, the done flag which indicates whether the episode has ended or not and an extra information string which is empty in our case.



Assign the done flag to false



Check if the action value is valid otherwise print an error message.



Create the reward variable and assign the value zero to it. The reward variable gives the reward gained after applying the action into the current state.



If the action value was 0, rotate the agent ship by 6 degrees to the left.



If the action value is 1, rotate the agent ship 6 degrees to the right.



If the action value is 2 launch an agent missile of velocity 30 and add it to the game elements list.



If the action value is 3, apply the accelerate function to the ship



If the action value is 5, apply the decelerate function to the agent ship.



Loop over all elements in the game



Apply the move method to that element. If the element was an enemy ship, it won’t move since the velocity attribute is set to 0.

If the element was an enemy ship



Fire a missile with probability of 0.08. The higher the probability value the more intense the firing would be.



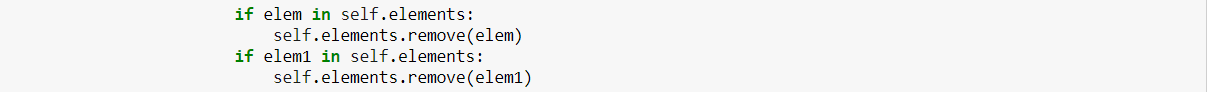
Check other elements in the list to check if the enemy ship has collided.



If the enemy ship collided with agent missile. Increment the reward variable by a factor of 30.



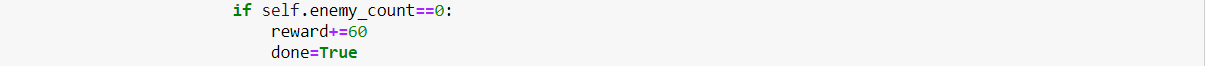
Delete both elements from the game elements list. Not doing so might result in a run time error



Decrease the number of enemy ships by 1.



Check if the number of enemy ships is 0 (fleet destroyed) in this case, increment the reward by 60 and set the done flag to True (the episode is finished).



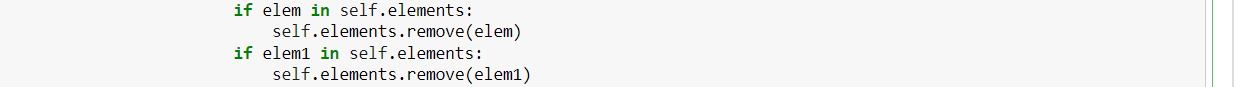
Else if the enemy ship collided with the agent ship.



Decrement the reward by 100 as a penalty.



Remove both elements from the game elements list.



Set the done flag to true.



If the element is agent ship

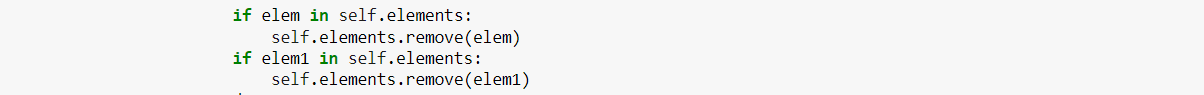


Loop over other elements. 

If the agent ship was hit by enemy missile, decrement the rewards by 100 points as a penalty.



Remove both elements from the game elements list



Set the done flag to true



If the element is an agent missile or enemy missile



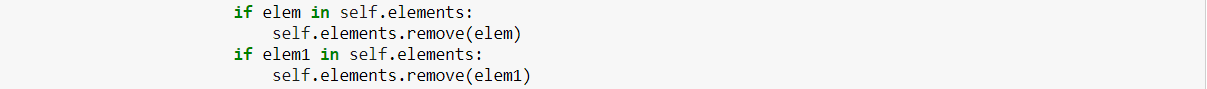
Loop over the other elements



If the agent missile hits an enemy missile, reward the agent extra 2 points.



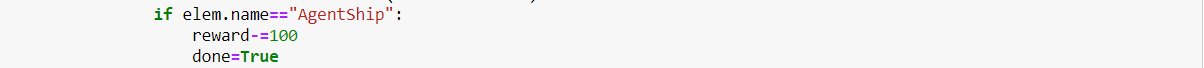
Remove both elements from the list



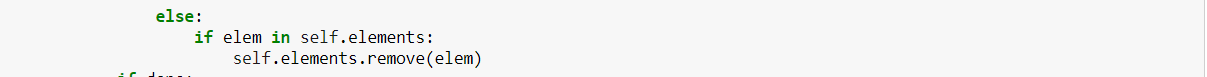
Check if the element is outside the game border



If the element was the agent ship, remove 100 points from the reward variable and set done to true



Otherwise remove the element



If the done flag is true stop the outer for loop



Accumulate the episodic return by adding the current reward



Draw all elements on canvas



Return the reduced-size canvas of the new observation, the reward, the done flag and an empty list



Define the has\_collide method to check if the two input elements has collided. Returns true if the two polygons intersects.



# Test a random agent using 20 episodes:

Create a new BattleSHipGame environment and assign it to variable env



Reset the environment and assign the initial state (observation) to variable obs



Set n\_episodes counter to zero.



While the episode counter is less than 20



Take a random action by generating a random number from {0,1,2,3,4}



Apply the action to the environment and return the new observation, the reward, the done flag and the info which is an empty list in my case.



Render the game canvas in a separate window



If the done flag is equal to true reset the game and increment the episode counter



Close al windows after resetting the game



# DQAgent class:

Define the DQAgent class



The constructor takes a gym compatible environment



Initialize the environment attribute



Define the memory buffer (replay buffer) that will save recent game experiences in a queue data structure. Each experience will contain the observation, the action, the reward, the next observation, and the done flag. The maximum queue length is 4,000 experiences.



Reset the environment and set the shape of the convolutional neural network (deep learning model), which will be same as the reduced size canvas (120x160x3)



Set the number of outputs of the deep learning model (5 outputs corresponding to the q-values of the ation at a given state)



The model attribute corresponds to the deep learning model. This will consist of 3 convolutional layers and 2 dense layers. The structure is based on the book as it is very hard to know the optimal structure. The size of the input is the same as the size of the game reduced-size canvas (120x160x3) and the outputs are basically the q-value of each corresponding action. Given a game observation the model will return the q-value of each action. We choose the action that has the highest q-value.

A picture containing text

Description automatically generated

Set the batch size which is the number of experiences used in each training step to teach the model estimating the q-values using the gradient descent algorithm.



Set the discount factor attribute which is used in computing the q-value formula for each action at a given state



Define the optimizer which is the RMSprop gradient descent variant the parameter values are based on the book



Define the loss function which is the mean squared error. The loss function measure how well did the deep learning model learnt the task on hand.



Define the epsilon gready policy method. This method takes the observation and a probability (epsilon). Based on this probability it will decide whether the action will be decided by the deep learning model or by the random agent. If the epsilon value is one, the random agent will decide the action for the given state. If the epsilon value is zero (default value) the action will be decided using the deep learning model based on the given state.



If a random variable is below epsilon.



Take a random action from the action space.



Else let the deep learning model compute the q-values of each action based on the given state



Take the action that gives the highest q-value



Define the playe\_one\_step method, this method uses the epsilon greedy policy to decide the action based on the given state and the it saves the experience in the replay memory.



Apply the epsilon greedy policy



Save the experience in the replay buffer. The experience is basically the given state, the action, the reward, next state and the done flag



Return the next state, the reward, the done flag and the info which is empty list



Define the sample experiences method. This method takes random experiences from the replay buffer to be used in the training step of the deep learning model. The number of experiences is based on the batch size which is 128 in my case.



Generate 128 random numbers less or equal to the replay buffer length. These number are basically the indices in the replay buffer of the random batch that will be picked for the training step.



Extract the experiences based on the random indices of the batch



Return the experiences



Define the training step method. This method takes a random experiences using the sample experiences method and then apply the gradient descent RMSprop to train the model predicting the q-values.



Extract the state, taken actions, rewards, next state (observation) and the done flags



The q-value formula is q-value of the current state is equal to the reward added to the maximum q-values of the next state multiplied by the discount rate.

To do that we first predict the q-values of the next states from the current batch



Take the maximum q-value



Multiply these values by the discount rate and add the reward. Execlude samples that has done equal to true (since there is no next state of the episode is terminated).



Apply the gradient descent to minimize the mean square error between the q-value and target q-values (computed previously). The gradient descent is based on the RMSprop variant

Scatter chart

Description automatically generated

Define the train method. This will train the model based on a number of episodes (12000 episodes by default.



These commands are used to ensure that the experiment results are reproducible.



Define a variable that stores the accumulated reward for each corresponding episode.



This variable is not used



This variable gives the total number of steps played



For each episode



Reset the environment.



Kee playing the episode until the done flag is true t which the while loop will be broken



Increment the number of steps



Compute the epsilon value. This function gives value close to 1 at the beginning of the training phase and a value close to zero (0.01) at the end of the training phase.



Use the play\_one\_step method to take an action using the random agent or deep learning model and the apply this action to the current training step and store the experience in the replay buffer



If the number of steps played is greater than 200 then apply the training step method at each 4th step



If the done flag is equal to true break the whil loop and start a new episode



Append the episodic return (total rewards) to the rewards list



Print the episode number and its corresponding episodic return



# Training the model:

Create a new BattleShipGame environment



Create a new DQAgent class and pass the game environment to the constructor



Invoke the train method to train the agent using 12000 episodes



# Plot the episodic return (total reward for each training episode):

A picture containing shape

Description automatically generated

# Record the performance of the random agent based on 5 episodes:

Reset the environment and set the initial state



This variable store all the frames from the game episode



Set the variable n\_episodes to zero



While n\_episodes<5



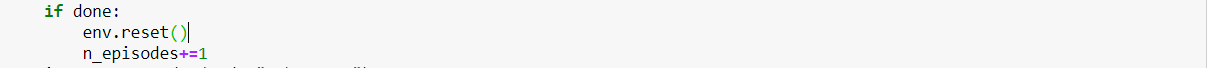
Decide the action based on the random agent since the epsilon is 1.



Apply the action and return the new state, the reward, the done flag and the info variable which is empty list in my case



If the done flag is True, reset the environment and increment the number of episodes



Get the current game canvas as an rgb image and add the image to the frames list



Plot the frames as a video using the plot animation function



# Record the performance of the AI agent based on 5 episodes:

Reset the environment and set the initial state



This variable store all the frames from the game episode



Set the variable n\_episodes to zero



While n\_episodes<5



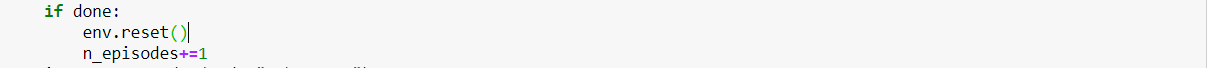
Decide the action based on the AI agent since the epsilon is 0 (default value) in this case.



Apply the action and return the new state, the reward, the done flag and the info variable which is empty list in my case



If the done flag is True, reset the environment and increment the number of episodes



Get the current game canvas as an rgb image and add the image to the frames list



Plot the frames as a video using the plot animation function



# Save the trained model weights

Save the model weight in the Models directory

