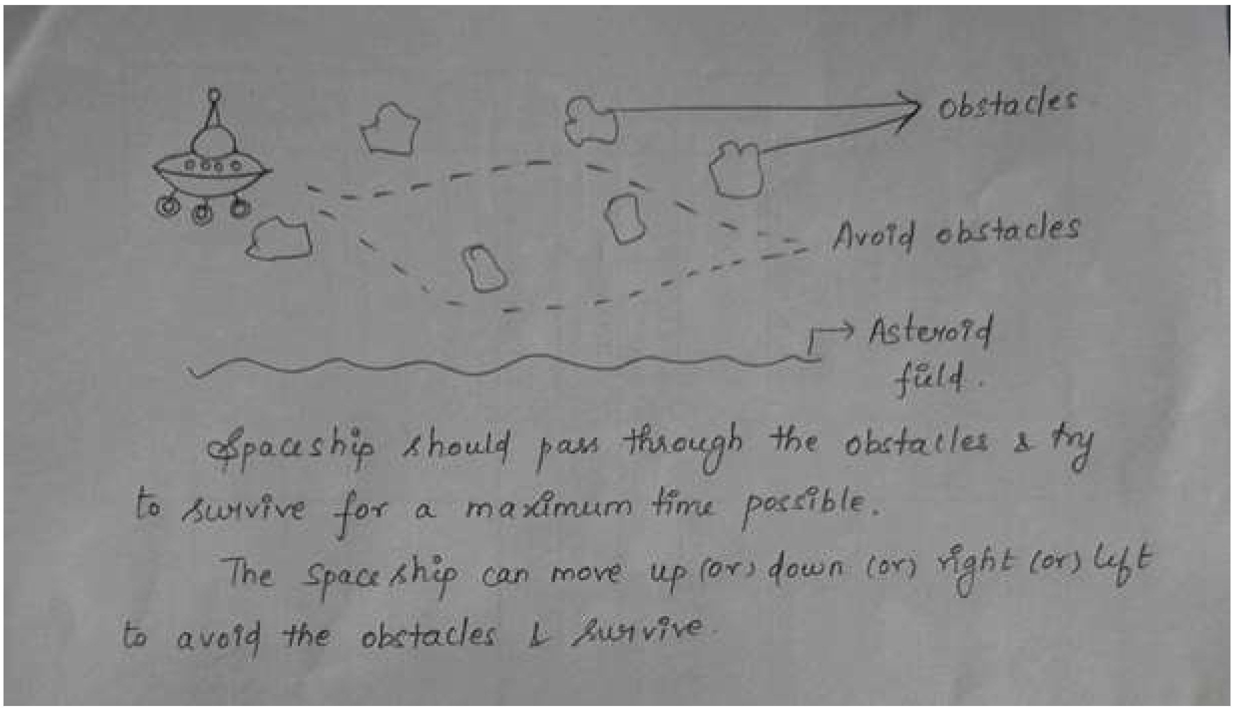
Assignment 2 - Part A - Spaceship Survival Game

Group 46

Satya Prakash Pandit - 2022AC05040
Pushkar Kumar Verma – 2022AC05272
G Krishna Sameera – 2022AC05407

Problem Statement: You are developing a mobile game where players control a spaceship navigating through an asteroid field. The objective is to avoid collisions with the asteroids for as long as possible. The game environment is represented as a 2D grid, where the spaceship can move up, down, left, or right. You can refer the following picture for an idea.


obstacles
Avoid obstacles
Asteroid field.
Spaceship should pass through the obstacles & try to survive for a maximum time possible.
The Spaceship can move up (or) down (or) right (or) left to avoid the obstacles & survive.

Objective: Design a deep neural network that takes the current state of the game environment (i.e., the positions of the spaceship and asteroids on the grid) as input and outputs the optimal action (i.e., move up, down, left, or right) to maximize the spaceship's survival time.

Additional Information:

- The game environment is dynamic, with asteroids moving randomly across the grid.
- The spaceship's movement speed and agility are constant.
- The reward system is based on the survival time, with higher rewards for longer survival durations.

- The neural network should use function approximation to learn the optimal policy for navigating the spaceship through the asteroid field.

Requirements and Deliverables:

1. Elaborate on, how the described problem could be solved using deep neural network and explain the action plan to create a gaming environment.**(1 Mark)**
2. Prepare a Colab sheet with outputs saved satisfying the following requirements. Implementation should be in OpenAI gym with python.

> (a) Develop a deep neural network architecture and training procedure that effectively learns the optimal policy for the spaceship to avoid collisions with asteroids and maximize its survival time in the game environment.

> (i) **Environment Setup** - Define the game environment, including the state space, action space, rewards, and terminal conditions (e.g., when the spaceship is destroyed). **(1 Mark)**

> (ii) **Replay Buffer** - Implement a replay buffer to store experiences (state, action, reward, next state, terminal flag). **(1 Mark)**

> (iii) **Deep Q-Network Architecture:** Design the neural network architecture for the DQN using Convolutional Neural Networks. The input to the network is the game state, and the output is the Q-values for each possible action. **(2 Marks)**

> (iv) **Epsilon-Greedy Exploration:** Implement an exploration strategy such as epsilon-greedy to balance exploration (trying new actions) and exploitation (using learned knowledge). **(1 Mark)**

> (v) **Training Loop:** Initialize the DQN and the target network (a separate network used to stabilize training). In each episode, reset the environment and observe the initial state. **(2 Marks)**

```
While the episode is not done (e.g., spaceship is not destroyed)
    Select an action using the epsilon-greedy strategy.
    Execute the action in the environment and observe the next state and reward.
    Store the experience in the replay buffer.
    Sample a batch of experiences from the replay buffer.
    Compute the target Q-values using the Bellman equation.
    Update the Q-network using back propagation and the loss between predicted and target Q-valu
    Periodically update the target network weights with the Q-network weights.
```

> (vi) **Testing and Evaluation:** After training, evaluate the DQN by running it in the environment without exploration (set epsilon to 0). Monitor metrics such as

> average reward per episode, survival time, etc., to assess the performance. **(1 Mark)**

Reference: https://colab.research.google.com/drive/1qTI0VeFfSq3yTQtm8a1l3SEc7Gp8EEyG?usp=sharing

Implementation

1. Action Plan

For the purpose **Deep Q-Network (DQN)** algorithm will be used.

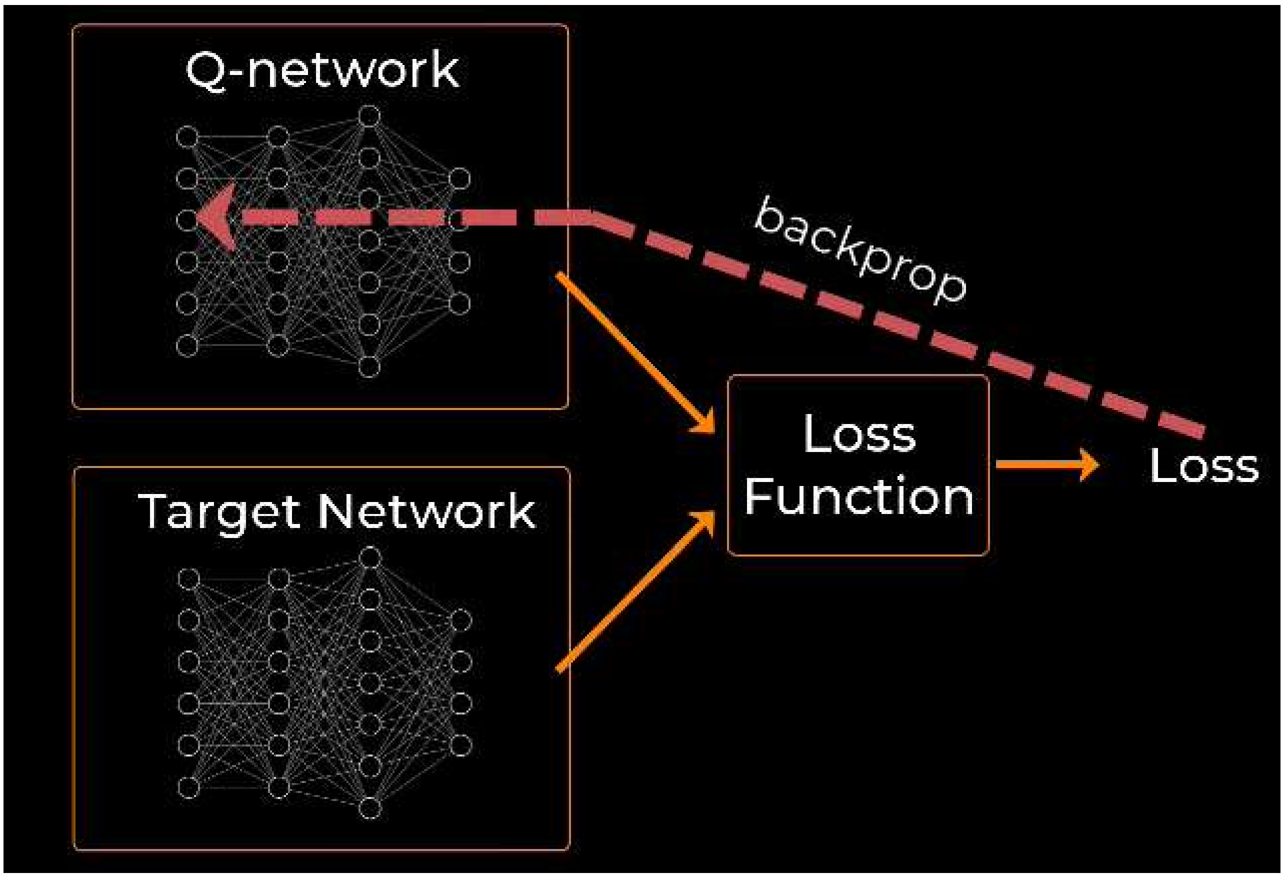
Environment:
We will create an OpenAI Gym environment representing the game. For we'll define the
state space,
action space,
rewards, and
terminal conditions.

Game Dynamics:
The spaceship can move up, down, left, or right. The asteroid's position will be updated randomly in each step.

Neural Network Model:
To impplement Neural Network Model we use **convolutional neural network (CNN)** model using TensorFlow/Keras. The model will approximate the Q-values for each action given a state.

Experience Replay Buffer:
state, action, reward, next state, done flag will be stored in a replay buffer to do correlations between consecutive experiences.

Exploration and Exploitation:
Epsilon-greedy strategy will be used to balances exploration and exploitation.



2. Python code

Block 1 to 6: Class definition

```
#Block 1: Environment Definition
#Definition of the custom Gym environment for the spaceship game.
#Initialization of observation space, action space, and other variables.

import gym
from gym import spaces
import numpy as np
class SpaceshipEnv(gym.Env):
    global observation_space
    observation_space = spaces.Box(low=0, high=255, shape=(64, 64, 3), dtype=np.uint8)
    print("observation_space",observation_space.shape)
    global action_space
    action_space = spaces.Discrete(4)
    global spaceship_position
    spaceship_position = [0, 0]
    global asteroid_position
    asteroid_position = [10, 10]
    global steps_beyond_done
    steps_beyond_done = None
    global max_steps
    max_steps = 100
    def _init_(self):
        super(SpaceshipEnv, self)._init_()
        self.action_space = spaces.Discrete(4) # Up, Down, Left, Right
        self.observation_space = spaces.Box(low=0, high=255, shape=(64, 64, 3), dtype=np.
        print("observation_space",observation_space.shape)
        self.spaceship_position = [0, 0]
        self.asteroid_position = [10, 10]
        self.steps_beyond_done = None
        self.max_steps = 100 # Define maximum steps per episode

#Block 2: Environment Step Function
#Define the step function for the environment, which takes an action and returns the next

    def step(self, action):
        steps_beyond_done = 0
        #assert self.action_space.contains(action), f"Invalid action {action}"
        if action == 0:
            self.spaceship_position[1] -=1 # Move Up
        elif action == 1:
            self.spaceship_position[1] += 1 # Move Down
        elif action == 2:
            self.spaceship_position[0] -=1 # Move Left
        elif action ==3:
            self.spaceship_position[0] += 1 # Move Right

# Update asteroid position (Example: random movement)
        self.asteroid_position[0] += np.random.randint(-1, 2)
        self.asteroid_position[1] += np.random.randint(-1, 2)
# Calculate reward
        reward = self.calculate_reward()
        # Check if episode is done
        steps_beyond_done += 1
        done = steps_beyond_done >= max_steps
        # Update observation
        observation = self.get_observation()
        return observation, reward, done, {}

#Block 3: Environment Reset Function
#Define the reset function for the environment, which resets the environment to its initi
    def reset(self):
        self.spaceship_position = [0, 0]
        self.asteroid_position = [10, 10]
        self.steps_beyond_done = 0
        return self.get_observation()

#Block 4: Environment Rendering Function
#Define the rendering function for the environment, which is used for visualizing the env

    def render(self, mode='human'):
    # Render the environment -- will be developed later
        pass

#Block 5: Observation Function
#Define a function to generate the observation based on the positions of the spaceship an
    def get_observation(self):
    # Example: Concatenate spaceship and asteroid positions in the observation
        observation = np.zeros((64, 64, 3), dtype=np.uint8)
        observation[self.spaceship_position[0], self.spaceship_position[1]] = [255, 255,
        observation[self.asteroid_position[0], self.asteroid_position[1]] = [255, 0, 0] #
        return observation

#Block 6: Reward Calculation Function
#Define a function to calculate the reward based on the distance between the spaceship an
    def calculate_reward(self):

    # Example: Reward based on distance between spaceship and asteroid
        distance = np.sqrt((self.spaceship_position[0] - self.asteroid_position[0])**2 +
        if distance < 5:
            return -10 # Penalty for being close to asteroid
        else:
            return 1 # Constant reward for survival
```

```
observation_space (64, 64, 3)
```

Block 7: Main Code

Create an instance of the custom environment. Define the state size and action size. Import required libraries, define constants, create the DQN model, initialize the replay buffer, and run the training loop to train the DQN. Evaluate the trained DQN on the environment. Each block of code serves a specific purpose in implementing the spaceship environment and training a DQN model to interact with it.

```
env = SpaceshipEnv()
state_size = observation_space.shape
action_size = action_space.n

import tensorflow as tf
from tensorflow.keras import layers, models
from collections import deque
import random

# Define constants
MEMORY_SIZE = 1000000
BATCH_SIZE = 64
GAMMA=0.95

LEARNING_RATE=0.001
EXPLORATION_MAX =1.0
EXPLORATION_MIN =0.01
EXPLORATION_DECAY = 0.995
EPISODES= 5
MAX_STEPS = 20

# Create DQN model
def create_q_network(state_size, action_size):
    model= models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=state_size),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(action_size, activation='linear')
    ])
    model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_R
    return model

# Initialize replay buffer
replay_buffer = deque(maxlen=MEMORY_SIZE)

# Initialize environment and DQN model
#env = SpaceshipEnv()
#state_size = env.observation_space.shape
#action_size = env.action_space.n
#q_network = create_q_network(state_size, action_size)
q_network = create_q_network(state_size, action_size)

# Training loop
epsilon= EXPLORATION_MAX
for episode in range(EPISODES):
    state= env.reset()
    state= np.expand_dims(state, axis=0)
    total_reward = 0

    for step in range(MAX_STEPS):
        if np.random.rand() <= epsilon:
            action= action_space.sample() # Explore
        else:
            q_values = q_network.predict(state)
            action=np.argmax(q_values[0]) # Exploit
        next_state, reward, done,_= env.step(action)
        next_state = np.expand_dims(next_state, axis=0)
        total_reward += reward

        replay_buffer.append((state, action, reward, next_state, done))

# Sample minibatch and train Q-network
        if len(replay_buffer) >= BATCH_SIZE:
            minibatch = random.sample(replay_buffer, BATCH_SIZE)
            for state, action, reward, next_state, done in minibatch:
                target= reward
                if not done:
                    target= reward+ GAMMA* np.amax(q_network.predict(next_state)[0])
                target_f = q_network.predict(state)
                target_f[0][action] = target
                q_network.fit(state, target_f, epochs=1, verbose=0)
        state= next_state
        if done:
            break

    epsilon= max(EXPLORATION_MIN, epsilon* EXPLORATION_DECAY)
    print(f'Episode: {episode+ 1}, Total Reward: {total_reward}')

# Evaluate trained DQN
total_rewards = []
for _ in range(500):
    state= env.reset()
    state= np.expand_dims(state, axis=0)
    total_reward = 0

    for _ in range(MAX_STEPS):
        q_values = q_network.predict(state)
        action= np.argmax(q_values[0])

        next_state, reward, done,_= env.step(action)
        next_state = np.expand_dims(next_state, axis=0)
        total_reward += reward

        state= next_state



        if done:
            break
    total_rewards.append(total_reward)
average_reward = np.mean(total_rewards)
print(f"Average Reward per Episode: {average_reward}")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarn
  and should_run_async(code)
```

Streaming output truncated to the last 5000 lines.

```
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 33ms/step
```

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 35ms/step
```

```
print(f"Average Reward per Episode: {average_reward}")
```

```
Average Reward per Episode: 19.098
```