Solving Subway Surfers using Reinforcement Learning

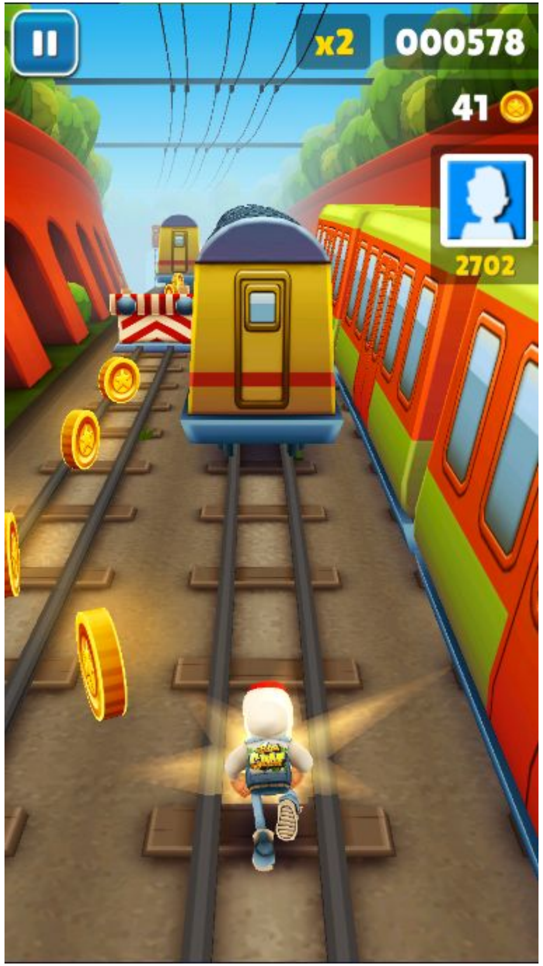
CS 386 Project : Group 6

Manan Jain (180010017), Rajnish Saraswat (180010023) & Yuvraj Agrawal (180010031)

Problem description

The problem is to train an agent to play the Subway Surfers game using DQN-CNN based Reinforcement Learning.

About Subway surfers - A user controlled agent runs on railway tracks and a cop chases it. It can't stop and needs to run endlessly otherwise it will get caught. It has 4 possible moves - to go left, right, jump up and roll down to avoid upcoming obstacles including moving/static trains and barriers.

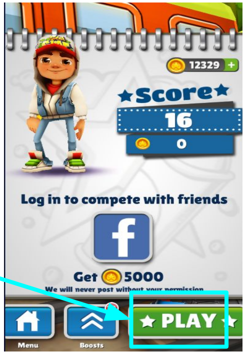


Let’s first understand the environment

Detecting the location of game on screen using two symbols and PyAutoGUI. The location include top, left, width and height of box in terms of pixels of screen.

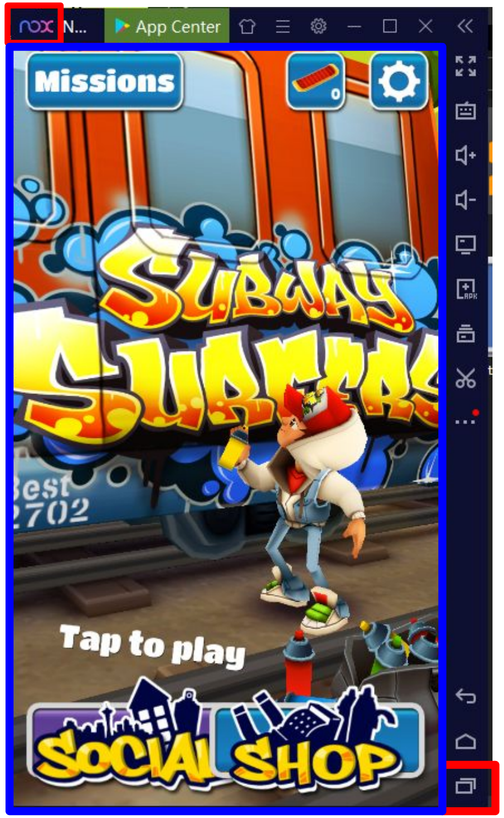
The agent can perform five actions - left swipe, right swipe, swipe up, swipe down & no-operation (i.e. Remain idle).

To detect the end of the game PyAutoGUI along with OpenCV search for presence of PLAY image in the screen


12329
Score
16
0
Log in to compete with friends
Get 5000
We will never post without your permission
PLAY
Menu
Boosts

The agent takes some time to detect the end of the game and performs some extra actions even after dying.

For each state/image if the agent does not die then reward of +2 is given, else reward of -10 is given.



Our Solution

Our project emphasizes on training the agent (runner) to learn and play the game. Agent detects the obstacles with the use of the input image and tries to make suitable moves, just like an ordinary human player would based on the environment.

We came to the conclusion to use Q-learning variants, in particular, due to the fact that the game we are trying to learn had discrete action spaces with near-zero tolerance for errors.

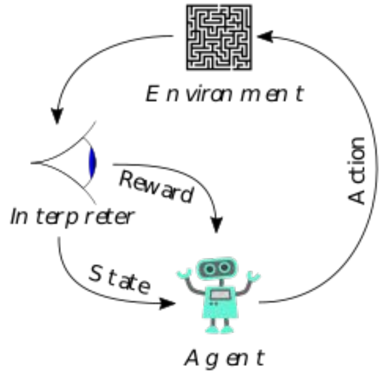
We have used DQN based reinforcement learning algorithm along with CNN with some modifications (Experience replay, LSTM & Eligibility Trace).

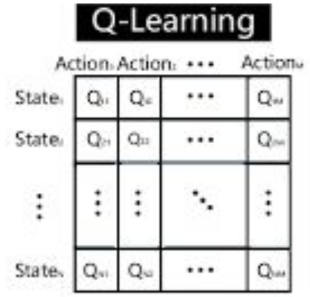
RL, Q Learning & DQN

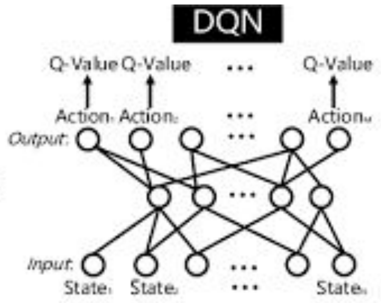
Reinforcement Learning is the branch of machine learning that permits systems to learn from the outcomes of their own decisions. It solves a particular kind of problem where decision making is sequential, and the goal is long-term.

Q Learning is a value-based RL algorithm which attempts to learn the value of being in a given state, and taking a specific action there.

Deep Q Learning uses a deep neural network to approximate and assign values to all possible moves taken by the agent (aka Q values). The goal is to calculate accurate Q values and hence take correct decisions. For example - In this game, we have 5 possible moves for every state, hence we try to calculate 5 Q-values, each corresponding to an action.







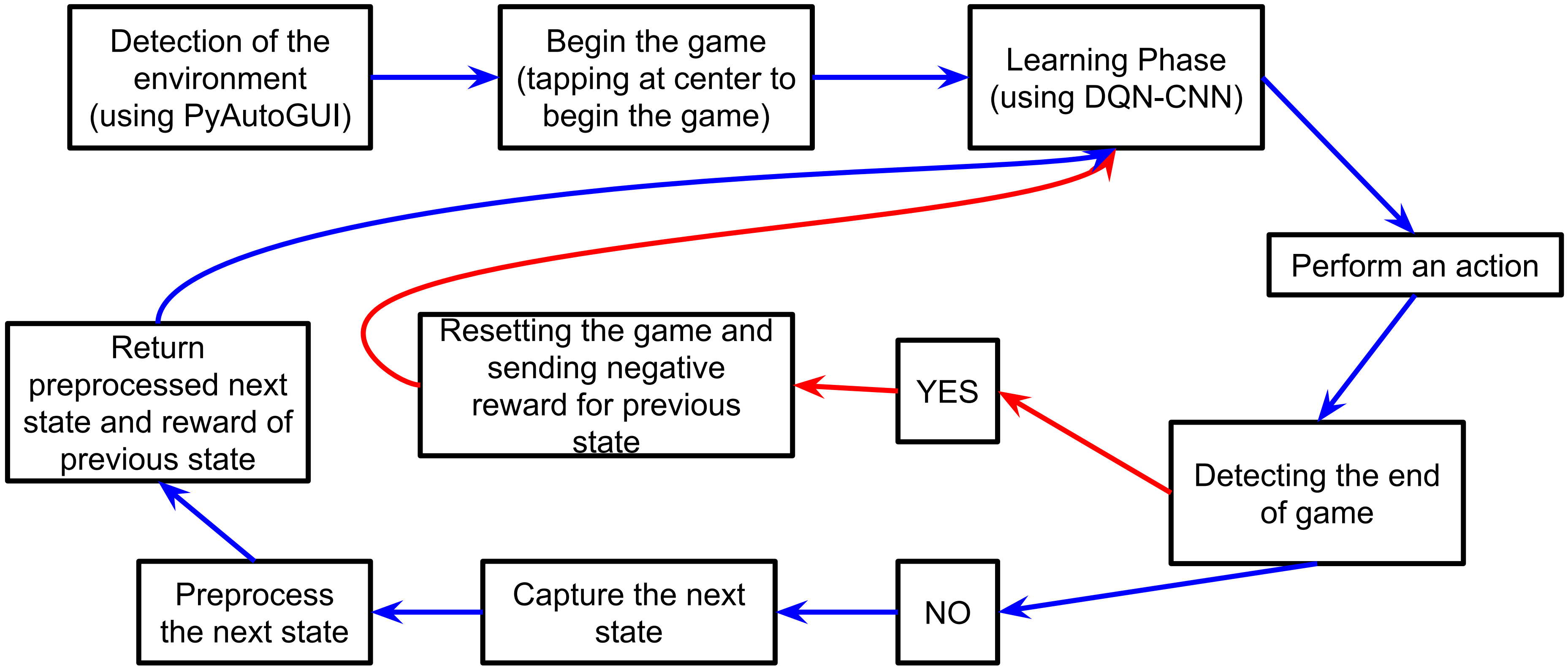
CNN & LSTM

Convolutional Neural Networks - These are generally used for image recognition, image classification or object detection tasks. In CNN, we take an image as an input, assign filters to detect important features/aspects of the image. Here, it is used to identify obstacles and find suitable path.

LSTM - Long-Short-Term Memory models have the ability to learn and remember over long sequences of input data through the use of “gates” which regulate the information flow of the network.

Here, it is used to identify relation between consecutive states and find the direction of the agent.

Overview of the model


Detection of the environment (using PyAutoGUI)
Begin the game (tapping at center to begin the game)
Learning Phase (using DQN-CNN)
Perform an action
Detecting the end of game
YES
Resetting the game and sending negative reward for previous state
NO
Capture the next state
Preprocess the next state
Return preprocessed next state and reward of previous state

The Network

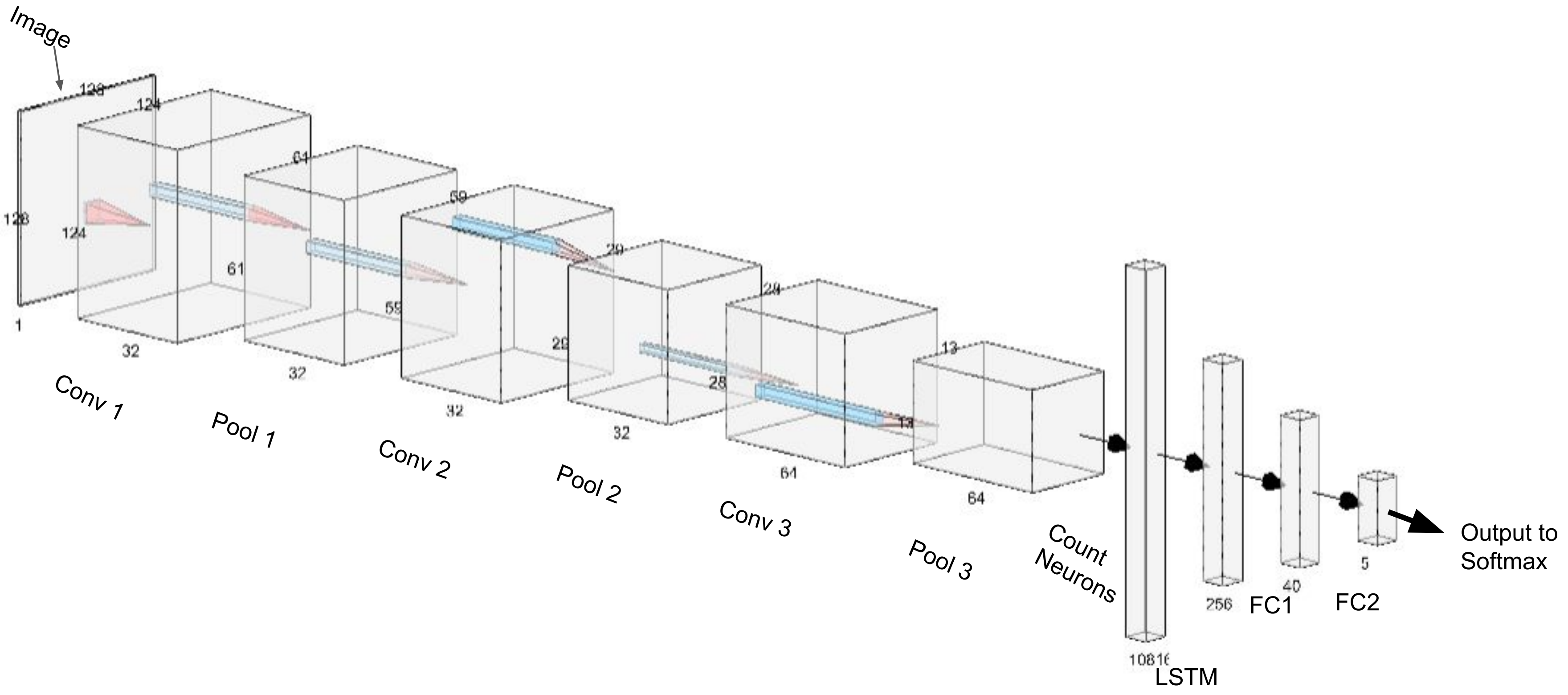
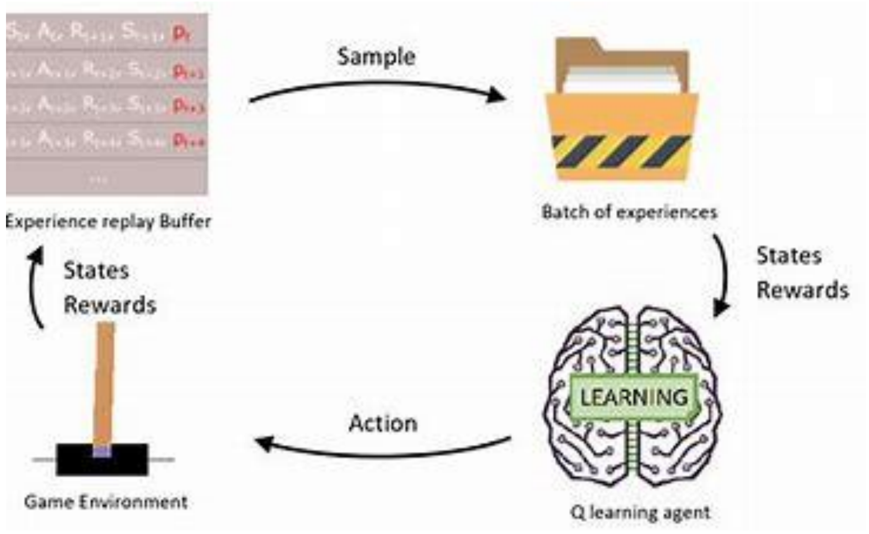


Image
128
128
1
124
124
32
Conv 1
64
61
32
Pool 1
59
59
32
Conv 2
29
29
32
Pool 2
28
28
64
Conv 3
13
13
64
Pool 3
Count Neurons
10816
LSTM
256
FC1
40
FC2
5
Output to Softmax

Functions used in the model

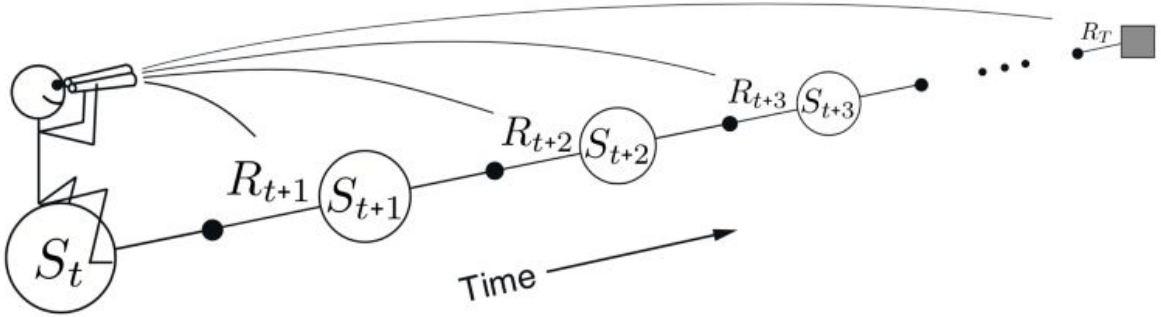
Replay Memory: We store the agent's experiences at each time step in a data set called the replay memory. Then random experiences from this memory are used for training. A key reason for this is to break the correlation between consecutive samples



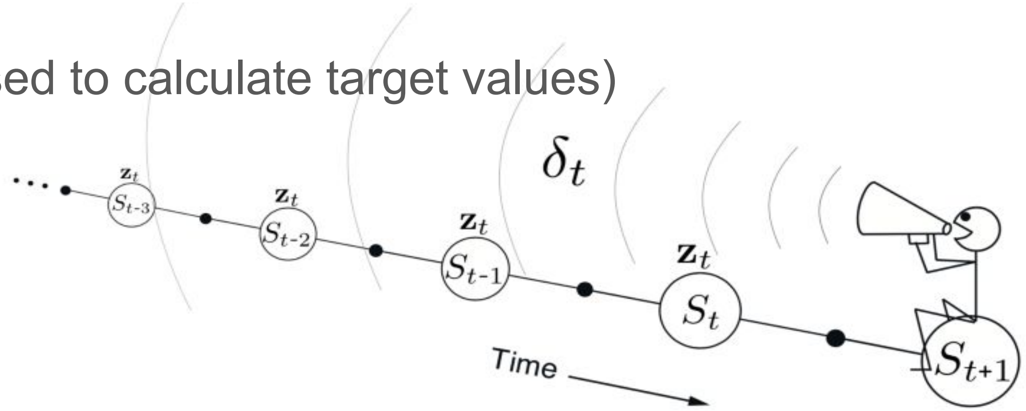
Moving Average: Average of last n-values.
In our case, we take the average of the last 100 rewards received by the agent to gauge the performance.

Functions used in the model

N-Step Progress



Eligibility Trace (used to calculate target values)



Now let’s understand the code of the Algorithm

Contributions - Manan

- Understanding the environment of Subway Surfers and finding out the best mode to run the game on PC. We had two options to run the game i.e. either using the .exe version or using the .apk version(Subway Surfers v1.0) on an android emulator. Here we are using the .apk file.
- Researched on Open-AI Universe (https://openai.com/blog/universe/) and read about it to figure out whether it could be used to implement algorithms on subway surfers. But, I was unable to use this implementation because of a lack of documentation.
- Contacted Opher Lieber, author of the blog (https://towardsdatascience.com/reinforcement-learning-for-mobile-games-161a62926f7e) to gain his valuable advice to create a model for this AI. But, unfortunately, according to him this project required a lot of expertise, experience, resources, and time than what we had. Although, he directed us to use VNC and to create a GYM wrapper.
- Created the preprocessing of the image screenshot, first grayscale the image(https://en.wikipedia.org/wiki/Grayscale#Luma_coding_in_video_systems) and then scaled the image to 128X128 using the matplotlib and numpy libraries. (Submitted as preprocess_image.py)
- We jointly researched a lot of reinforcement learning algorithms and looked at comparative studies of them to find a suitable algorithm for our problem. We came to the conclusion to use q-learning variants, in particular, due to the fact that the game we are trying to learn had discrete action spaces with near-zero tolerance for errors.

Contributions - Manan

- We finally decided to go with DQN with some modifications.
- We combined our codes to generate an environment to play our game. (Submitted as env.py)
- After the environment and research part was over, we inserted the algorithm at the heart of this environment and trained the model. (Submitted as ai.py, ma.py and n_step.py)
- Worked on the algorithm mainly the network part with Rajnish to finalize the network. Including the number of layers, convolution, pooling layers, and fully connected layers.
- Worked on understanding and implementing the experience replay in our algorithm as it was very useful in our game.(submitted as replay_memory.py)
- As our environment uses the screen and takes a screenshot and sends back the actions so running multiple instances was not feasible, so for final evaluation we build a model that is trained on a single environment using the DQN algorithm.

Contributions - Rajnish

- Tried out numerous android emulators to run the game keeping in mind the minimum computation power available to us and finalized the Nox Player (https://www.bignox.com/)
- Researched Open-AI baselines (https://github.com/openai/baselines) and stable-baselines (https://stable-baselines.readthedocs.io/en/master/) and figured out the algorithm implementation that we used in our game. But, all of these implementations used GYM wrapper so we were unable to use them.
- I also found Serpent AI (http://serpent.ai/), an alternative to the Open-AI Universe, but creating the environment in this requires a lot of time and expertise which makes it out of our current scope.
- Created the code to perform actions on the game using swiping gestures. The file has 5 actions- left, right, up, down, and no-operation. Used the pyautogui library. (Submitted as action.py)
- We jointly researched a lot of reinforcement learning algorithms and looked at comparative studies of them to find a suitable algorithm for our problem. We came to the conclusion to use q-learning variants, in particular, due to the fact that the game we are trying to learn had discrete action spaces with near-zero tolerance for errors.

Contributions - Rajnish

- We finally decided to go with DQN with some modifications.
- We combined our codes to generate an environment to play our game. (Submitted as env.py)
- After the environment and research part got over, we then inserted the algorithm at the heart of this environment and train the model. (Submitted as ai.py, ma.py and n_step.py)
- Worked on the algorithm mainly the network part with Manan to finalize the network. Including the number of layers, convolution, pooling layers, and fully connected layers.
- I also tried to find resources to figure out whether LSTM was useful in our model and after getting the answer as yes, we researched about the implementation.(submitted as nn.py)
- As our environment uses the screen and takes a screenshot and sends back the actions so running multiple instances was not feasible, so for final evaluation we build a model that is trained on a single environment using the DQN algorithm.

Contributions - Yuvraj

- Researched a lot of implementations of Reinforcement Learning Algorithms on games. And tried to figure out a way to implement reinforcement-learning in our game. (https://medium.com/@SmartLabAI/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc) (https://towardsdatascience.com/an-exploration-of-neural-networks-playing-video-games-3910dcee8e4a) (https://www.linkedin.com/pulse/understanding-intelligence-one-pixel-time-aamir-mirza/)
- Researched Open-AI Retro (https://openai.com/blog/gym-retro/) and read about it to figure out whether it can be used to implement algorithms on subway surfers. But, the implementation was quite complicated and requires a lot of time.
- Explored the possibility of using VNC (also used by Open-AI Universe) for our project but they require huge computational power along with a good understanding of Networks and expertise in using Android SDK (in Swiftshader mode) to share information using ports. All this requires a lot of time to first learn and apply so this was also not a feasible option for us.
- Created the code to automatically detect the location of NoxPlayer and store the coordinates of the game to take screenshots and train the model. Used pyautogui to detect the game and take the screenshot. (Submitted as start_game.py)

Contributions - Yuvraj

- We jointly researched a lot of reinforcement learning algorithms and looked at comparative studies of them to find a suitable algorithm for our problem. We came to the conclusion to use q-learning variants, in particular, due to the fact that the game we are trying to learn had discrete action spaces with near-zero tolerance for errors.
- We finally decided to go with DQN with some modifications.
- We combined our codes to generate an environment(basic idea was to create something like a Gym wrapper) to play our game. (Submitted as env.py)
- After the environment and research part was over, now we inserted the algorithm at the heart of this environment and train the model. (Submitted as ai.py, ma.py and n_step.py)
- I tried to incorporate eligibility trace in the implementation of our model as it helps in improving the results.(submitted as eligibility_trace.py)
- Mainly worked on creating a wrapper(taking ideas from GYM wrapper) for the game through which we can control the game and use the observation to train our model on general implementation of algorithms which were made for the GYM environment. Thus, treating both the algorithm and environment as black boxes and creating a method to connect both of them. (submitted as env.py)
- As our environment uses the screen and takes a screenshot and sends back the actions so running multiple instances is not feasible currently, so till final evaluation we believe we will be able to build a model that is trained on a single environment using the DQN algorithm.

Results

Despite of very less training our model learnt to dodge some obstacles and achieve an average score of around 250 (high score of 512).

During training for some epochs it learnt to keep jumping or rolling as the agent is not penalized for these actions and sometimes it helps to dodge some obstacles.

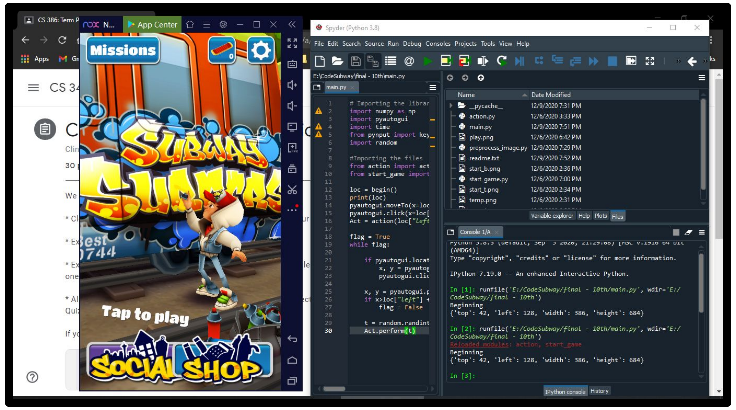
For better results the model require more training.

Requirements

- NoxStudio
- PyAutoGUI
- Subway Surfers v1.0 (Install the given APK in the android emulator)
- OpenCV
- Matplotlib
- Numpy
- sklearn
- PyTorch

Instructions to replicate our model:

- Download the .zip file and extract the contents to a folder.
- Follow the readme present in install folder.
- Make the folder as your current directory in cmd or any other console that you are using. (For this project we prefer to use pyTorch)
- Launch Nox and run subway surfers in it.
- Then run the code(ai.py) and Subway Surfer should be running on Nox Player such that the whole screen of Nox Player is visible while running the code.
- Don't alter the folder structure.
- Don't disturb the mouse once you run the code.
- To exit the game repeatedly try to bring the cursor out of the game window.


Missions
Subway Surfers
Tap to play
Social
Shop
Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Variable explorer Help Plots Files
Console 1/A
IPython console History

E:\CodeSubway\final - 10th\main.py

```
# Importing the librar
import numpy as np
import pyautogui
import time
from pynput import key
import random

#Importing the files
from action import act
from start_game import

loc = begin()
print(loc)
pyautogui.moveTo(x=loc
pyautogui.click(x=loc[
Act = action(loc["left

flag = True
while flag:

    if pyautogui.locat
        x, y = pyautog
        pyautogui.clic

    x, y = pyautogui.p
    if x>loc["left"] +
        flag = False

    t = random.randint
    Act.perform(t)
```

Name	Date Modified
__pycache__	12/9/2020 7:31 PM
action.py	12/6/2020 3:33 PM
main.py	12/9/2020 7:51 PM
play.png	12/6/2020 6:42 PM
preprocess_image.py	12/9/2020 7:29 PM
readme.txt	12/9/2020 7:52 PM
start_b.png	12/6/2020 2:36 PM
start_game.py	12/6/2020 7:00 PM
start_t.png	12/6/2020 2:34 PM
temp.png	12/6/2020 2:31 PM

```
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('E:/CodeSubway/final - 10th/main.py', wdir='E:/
CodeSubway/final - 10th')
Beginning
{'top': 42, 'left': 128, 'width': 386, 'height': 684}

In [2]: runfile('E:/CodeSubway/final - 10th/main.py', wdir='E:/
CodeSubway/final - 10th')
Reloaded modules: action, start_game
Beginning
{'top': 42, 'left': 128, 'width': 386, 'height': 684}

In [3]:
```

References:

- https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
- https://www.linkedin.com/pulse/understanding-intelligence-one-pixel-time-aamir-mirza/
- https://towardsdatascience.com/an-exploration-of-neural-networks-playing-video-games-3910dcee8e4a
- https://www.superdatascience.com/pages/artificial-intelligence
- https://towardsdatascience.com/reinforcement-learning-for-mobile-games-161a62926f7e
- https://github.com/opherlieber/rltime/issues/1
- https://medium.com/@SmartLabAI/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc
- https://www.udemy.com/course/artificial-intelligence-az/
- https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2