Engineering Design Document

Fireboy and Watergirl (ZED)

GOAL OF THE PROJECT:

The goal of this project is to apply deep learning concepts, viz. reinforcement learning over an already existing game named 'Zed' to train the character to play the game by itself without any intervention of the user/player. Through various reinforcement learning techniques, the character will learn to play the game. The game has various levels/stages but this project at this point of time would only be utilizing the 1st stage of the game for training and testing purposes.

BACKGROUND AND OVERVIEW:

Zed is an android. But unlike other androids, he has always had a very special dream, to sport a space suit entirely made of gold. The objective is to guide Zed through ten particularly vicious levels and collect enough gold pieces to fulfil his dream. The quest begins in the gold mines, deep down in the underworld. Collect good stuff. Avoid bad stuff. Three gold pieces are need to unlock a golden door at each level to get through to next level.

It's a 2D platformer adventure game owned by MiniClip (world's largest privately owned online gaming website). We found this game to be near-perfect for this project and our purposes.

To control Zed:

Left arrow key: to move left
Right arrow key: to move right
Up arrow key: to jump

RESEARCH:

Our research prior to attempting the project and while doing the project:

- Read plethora of papers related to 'Reinforcement learning for games' online.
- Went through a lot of articles and videos tutorials.
- Since our project involves applying reinforcement learning over an already existing game, it makes it quite obvious that we won't have access to the source code of the game. Therefore, achieving this task involved a lot of research on how to implement learning on already built games. This involved research on:
    - Simulating key presses through python.
    - Simulating mouse controls through python.
    - Grabbing real time images to feed to the CNN for training.
    - Finding out the ideal size of these images that would translate essential details on the game screen (specifically for a 2D platformer game like ours) and also would be relatively easier for the CNN to train on.
    - Reading text from images through Pytesseract (for rewards).
    - Computer vision libraries for image manipulation.
- Read and learnt about Keras, Tensorflow and Pytorch to implement neural networks and deep learning in general.
- Studied Q-Learning and how it works.
- Researched and learnt about Deep Q-Learning and how to implement it using neural networks.
- Explored Open AI Gym and its various environments.
- Read about MobileNets in Keras and how to use them for feature extraction from an image.

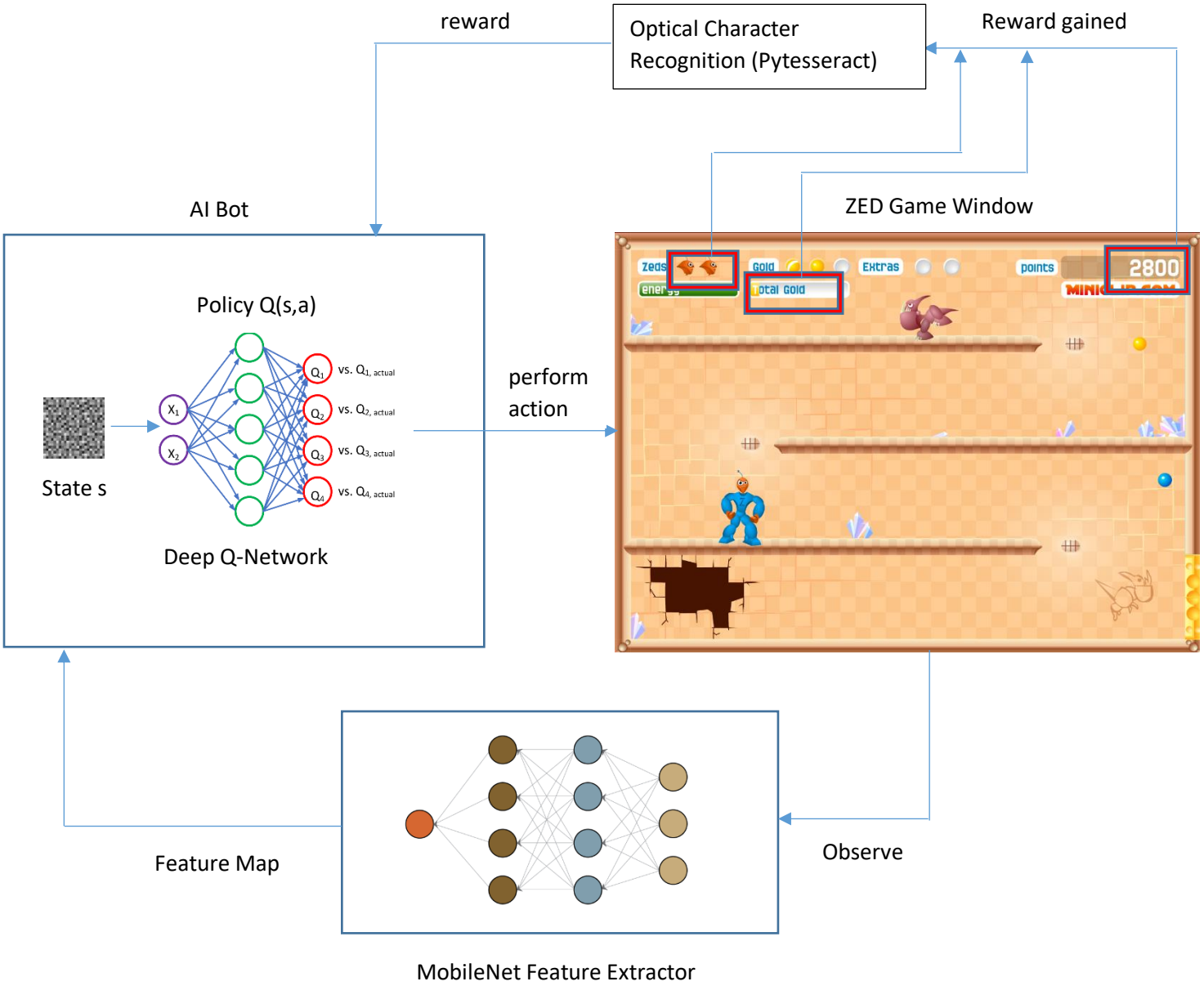
METHODOLOGY:

Basic flow:

- The game is kept running in a 800 x 600 window.
- Python then uses ImageGrab to take a screenshot of the game screen and feeds it to MobileNet which is a CNN feature extractor.
- The feature map extracted from this CNN represents the current state of the game environment.
- A dense neural network takes this state as an input and decides what action to take (output). This network is essentially a Q-Learning model.
- This Deep Q-Network takes one of four actions. [left, right, jump left, jump right].
- Once an action is taken to interact with the game, an outcome is observed through the points meter on the top-right corner of the game screen. We have used pytesseract to parse and read text from the screen and assign rewards for an action accordingly. Also the part of the screen towards the top-

left which represents the number of lives remaining, and the gold bar are also used to assign a reward.

- This information (reward) is fed back to the Deep Q-Network. The network adjusts its policy accordingly based on thousands of such interactions with the game. While observing such feedbacks, it will learn over time.



The nitty-gritty:

- Our code uses experience replay memory. This memory is used to store past experiences i.e <s,a,r,s'> is one experience where:
  s-> current state
  a-> action taken in current state
  r-> reward obtained for this action
  s'-> new state entered due to the action
- A game over Boolean is also stored along with an experience for the network to know if the game got over due to an experience or not. […, (<s, a, r, s'>, <game_over>), …]
- During training, batches of randomly drawn experiences are used to generate the input and target for training.
- Our Q-Learning model utilizes an epsilon-greedy strategy. A random number between 0 and 1 is generated. If the epsilon-value is greater than this random number, a random action is taken. Else, our Deep Q-Network predicts an action for us. This epsilon-value starts from 1 and decays gradually according to a decay-rate, as the training progresses.
- Our code is set to read the game window's coordinates assuming that the screen will open at the middle of the system's screen. It thus grabs screenshots of the game window and a lot of other information using these coordinates.
- Game is restarted every 100 steps to reset the environment. This goes on for 100 games/epochs.

Libraries used:

- Keras
- TensorFlow
- Grabscreen
- NumPy
- Pytesseract
- Matplotlib
- cv2
- PIL
- pynput
- ctypes
- time
- os

REFERENCES:

- https://www.google.com/
- https://www.youtube.com/
- https://www.freecodecamp.org/news/deep-reinforcement-learning-where-to-start-291fb0058c01/
- https://github.com/Sentdex/pygta5
- https://blog.paperspace.com/dino-run/
- https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a
- https://pathmind.com/wiki/deep-reinforcement-learning