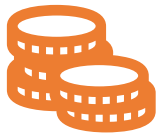
Implementation of PONG Game using Deep Q Networks (DQN)

Akshatha Thonnuru Swamygowda
(017532175)

Sarveshwaran Sampathkumar
(017387654)



In 2014 Google acquired a small London based start-up called DeepMind for 500 Million Dollars :O

It was a simple bot for Atari Games

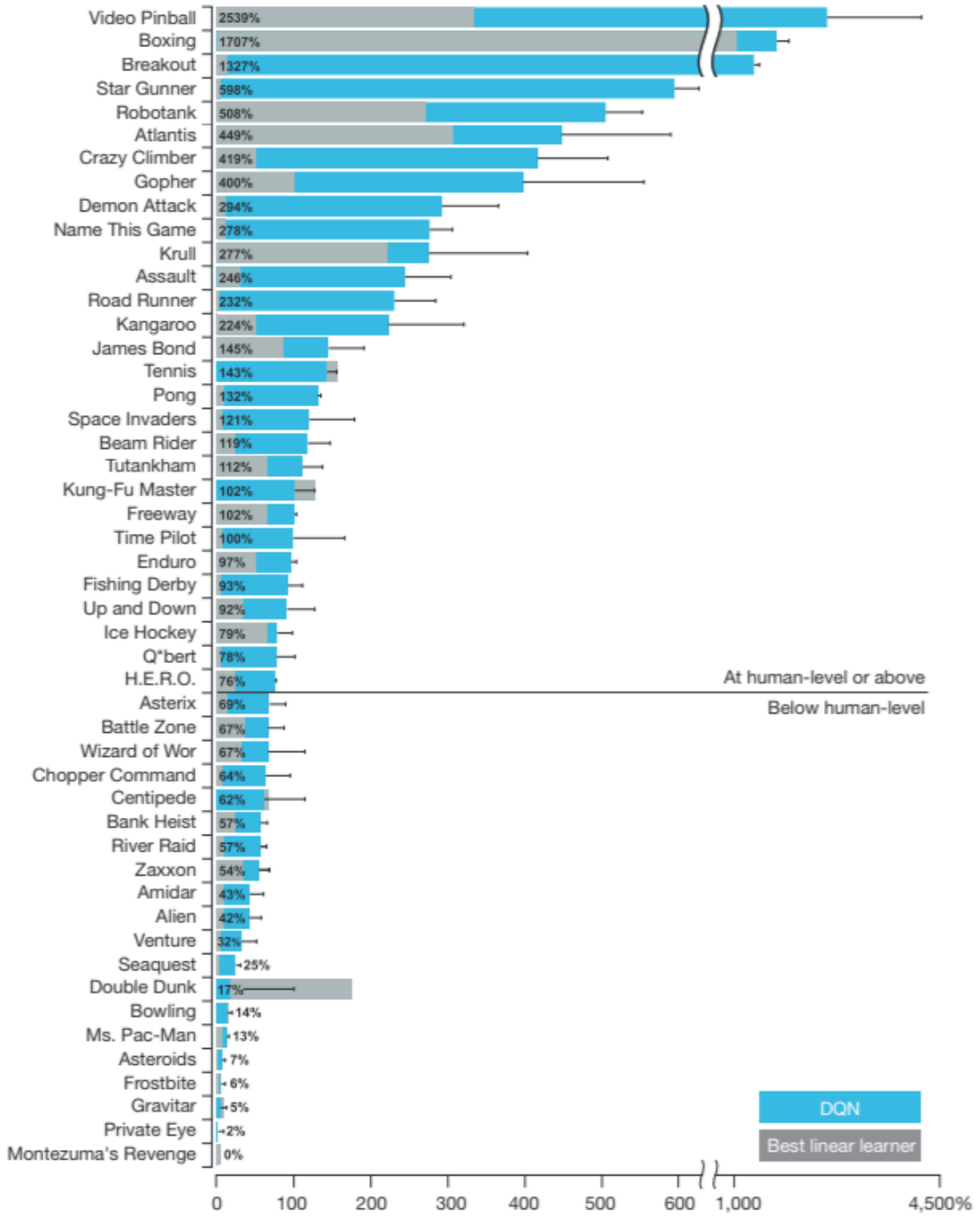
But the reason they paid so much for it was because it was one of the first step towards General Artificial Intelligence

This AI can excel in not one but a variety of tasks

Their Algorithm could be applied to 50 different Atari Games

Background & Motivation

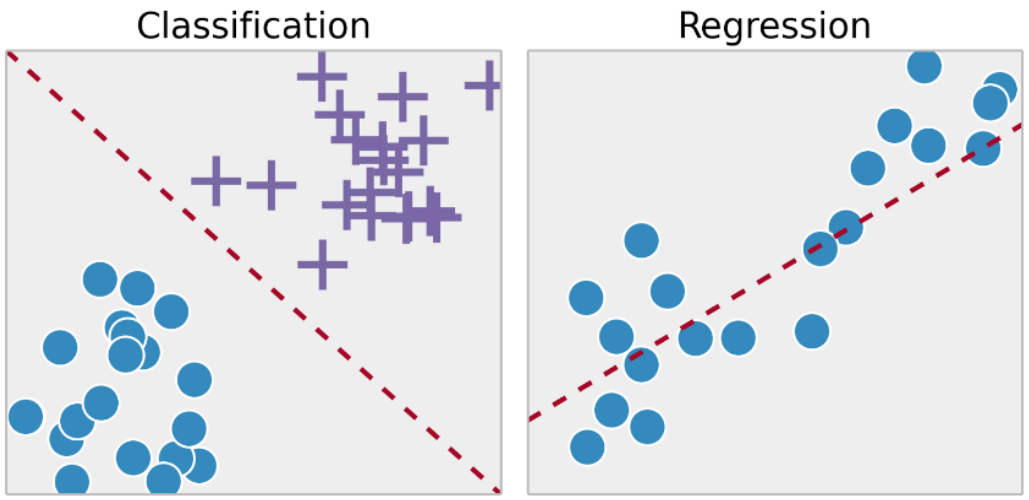
Background & Motivation



Reinforcement Learning

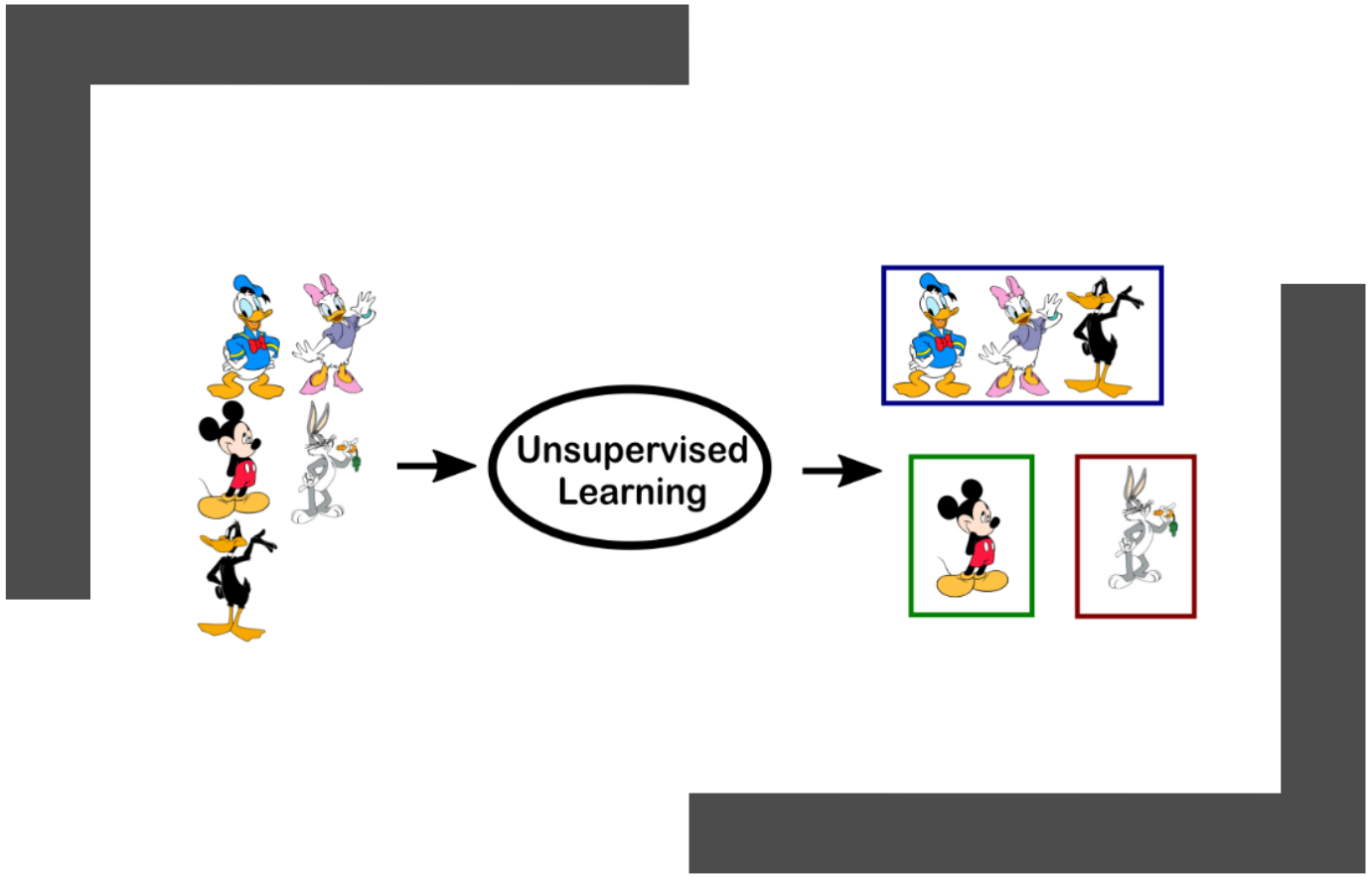
Before we dive into Reinforcement Learning lets glance over Supervised and Unsupervised Learning

Supervised Learning

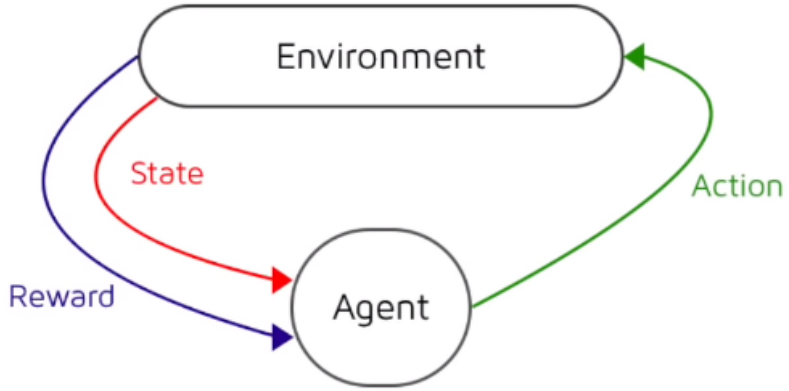


- Supervised learning is typically done in the context of classification, when we want to map input to output labels, or regression, when we want to map input to a continuous output.

Unsupervised Learning



- The most common tasks within unsupervised learning are clustering, representation learning, and density estimation. In all these cases, we wish to learn the inherent structure of our data without using explicitly-provided labels.



Reinforcement Learning

- Reinforcement learning is an important type of Machine Learning where an agent learn how to behave in an environment by performing actions and seeing the results

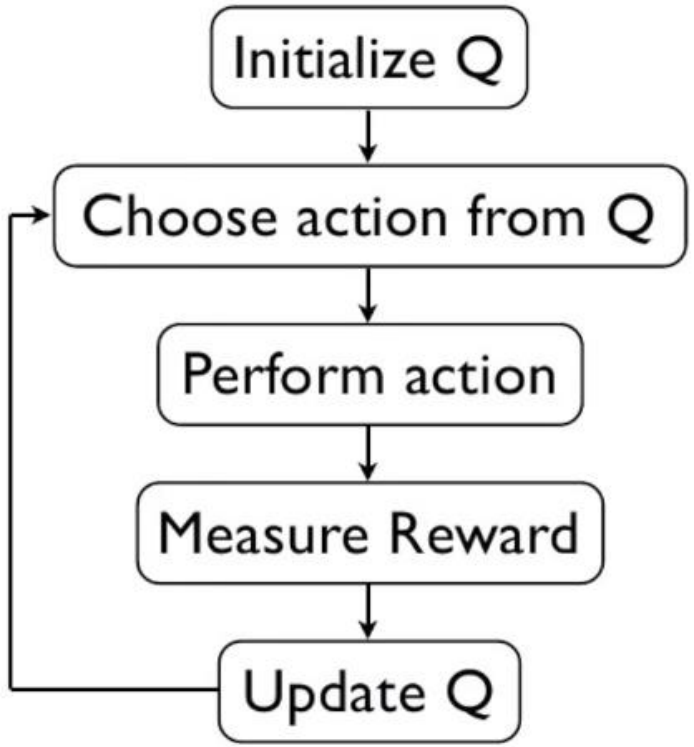
Markov Decision Processes (MDP)

- A mathematical representation of a complex decision-making process is "**Markov Decision Processes**". The goal of the MDP is to find a policy that can tell us, for any state, which action to take.
- The optimal policy is the one that maximizes the long-term expected reward.

Markov Decision Processes (MDP)

There are two properties on which the Markovian process is based on:

- Only the present matters; which means that the transition function only depends on the current state S and not any of the previous states.
- Things are stationary; therefore rules do no change over time.



Idea – Q Learning

- For the agent to be smart, it shouldn't just play for short term rewards but for long term rewards as well
- Whenever a player is deciding between several possible actions, the solution is picking up the action with the highest Q (Quality) value
- Computing the Q function is where the learning process comes in
- Maximum future reward for a present state/action is the sum of the reward and the maximum reward of the future state/action

Idea – Q Learning

- Q function - Matrix with States as rows and Actions as columns
- Start by initialising the Matrix with random values
- Observe the initial state of the game
- Then approximate using a training process where we first pick up an action, execute it and then see if we receive a reward for that action and the resulting state
- This is called Q Learning

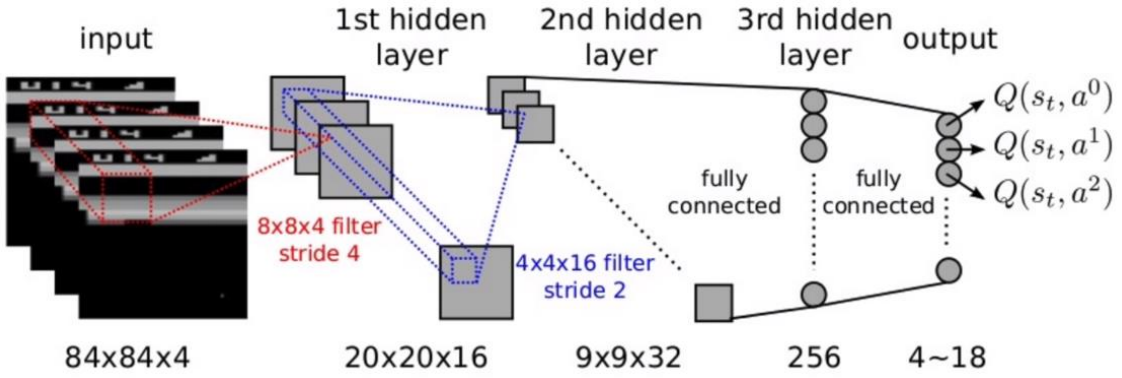
How do we Generalise this

- One thing that all games share are the pixels
- If we can convert pixels from game screen into action, we can feed in any game and it would learn the optimal policy for that game
- Consider we use Convolutional Neural Network (CNN), we can use game screen as input and output the Q value for each possible action

How do we Generalise this

- DQN – Consists of 3 Convolution Layers and 2 Fully Connected Layers
- Input – 4 different game screens, so that we have a way of representing speed and direction of the game characters
- Output – Q Values of all the possible actions which can performed by the player
- Since we are using deep network to help approximate the Q Function we can call this process as **Deep Q Learning**

Deep Q Network



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

The Game of PONG

Libraries Used

- TensorFlow
- CV2
- Numpy
- Random
- Pygame

Methods

- definePingPongBall
- defineLeftPaddle
- defineRightPaddle
- updatePingPongBall
- updateLeftPaddle
- updateRightPaddle
- returnPresentFrameInformation
- returnNextFrameInformation
- tensorFlowGraph
- deepQNetworkGraphTraining

Flow of the code

Initialise the libraries, parameters and network

- Experiences are stored in Deque
- Parameters (OBSERVE, EPSILON, GAMMA, BATCH_SIZE, etc) are initialized
- Create a network

Observing what each action does

- Start the game
- Perform a random action
- Observe the new state, the reward received for that particular action

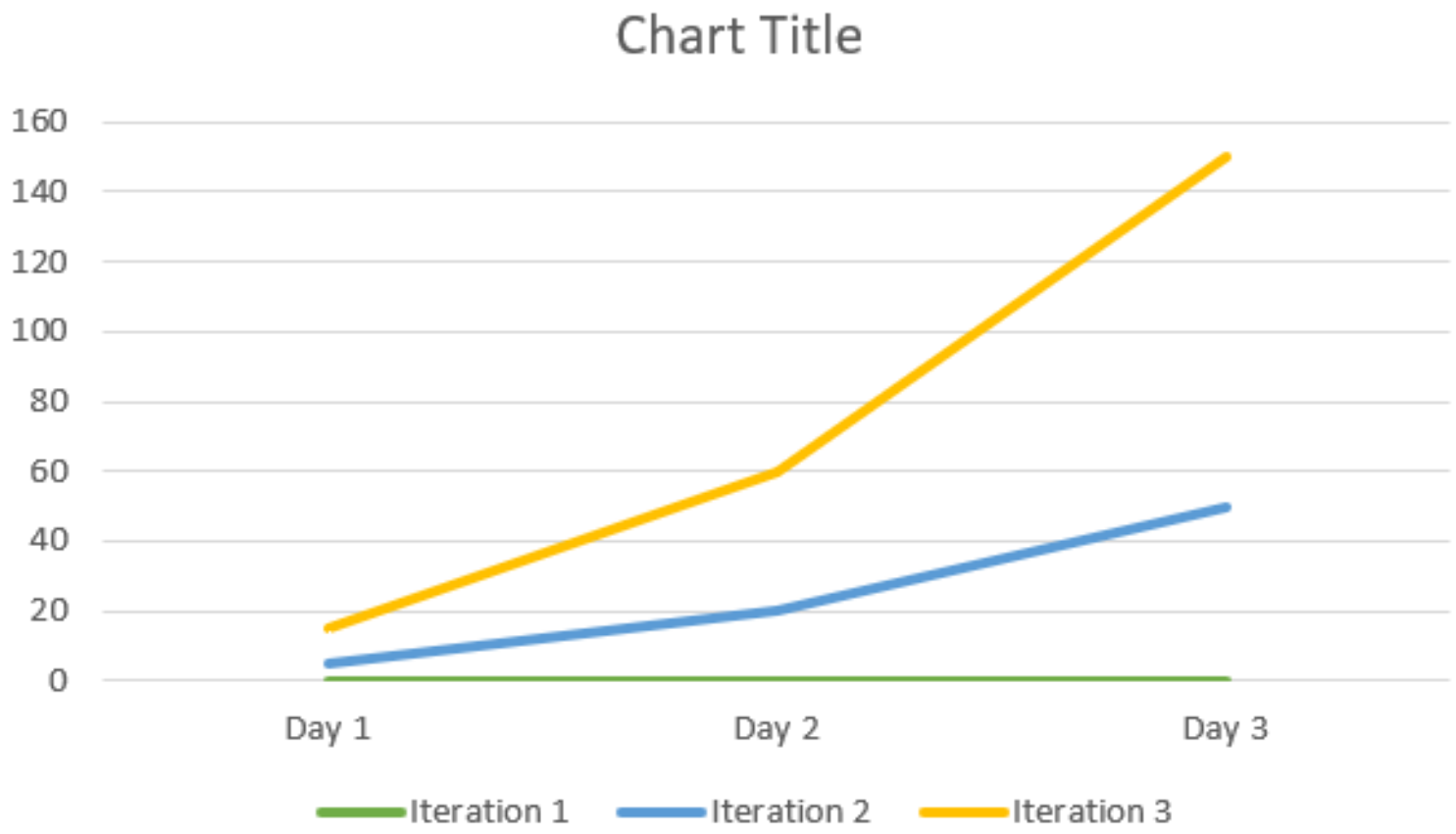
Experience Delay – Learning from the observations

- Bellman Equation is built for the Q Function

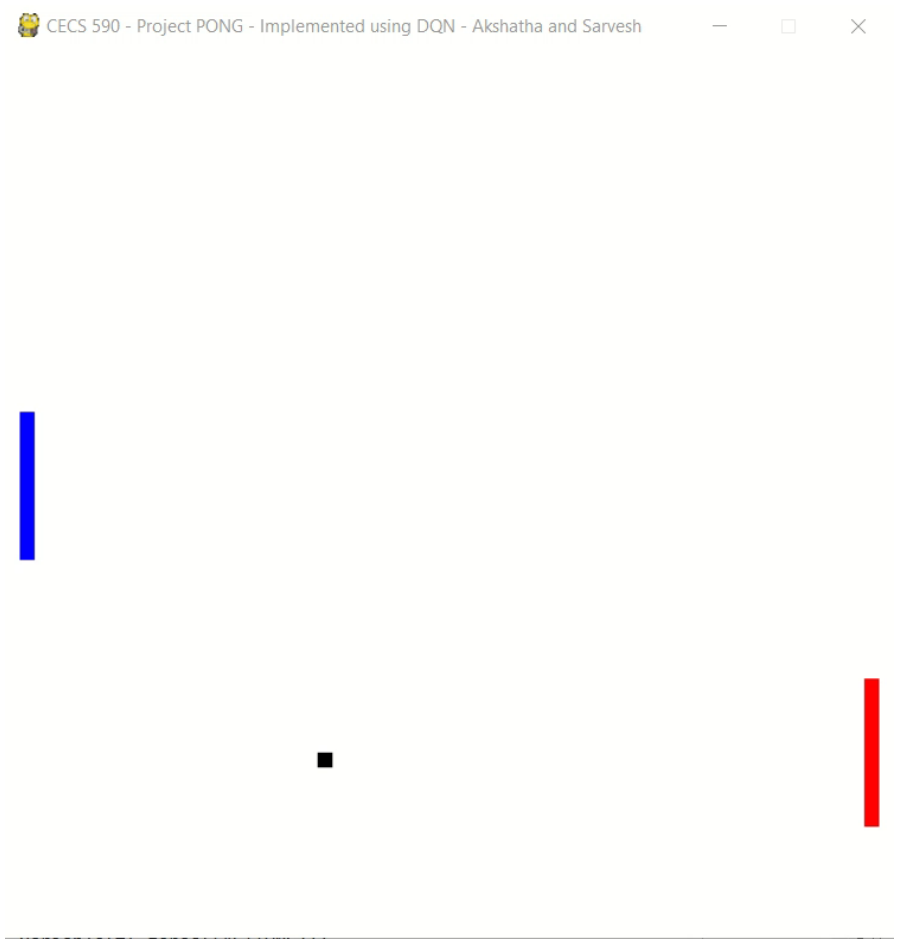
Challenges Faced

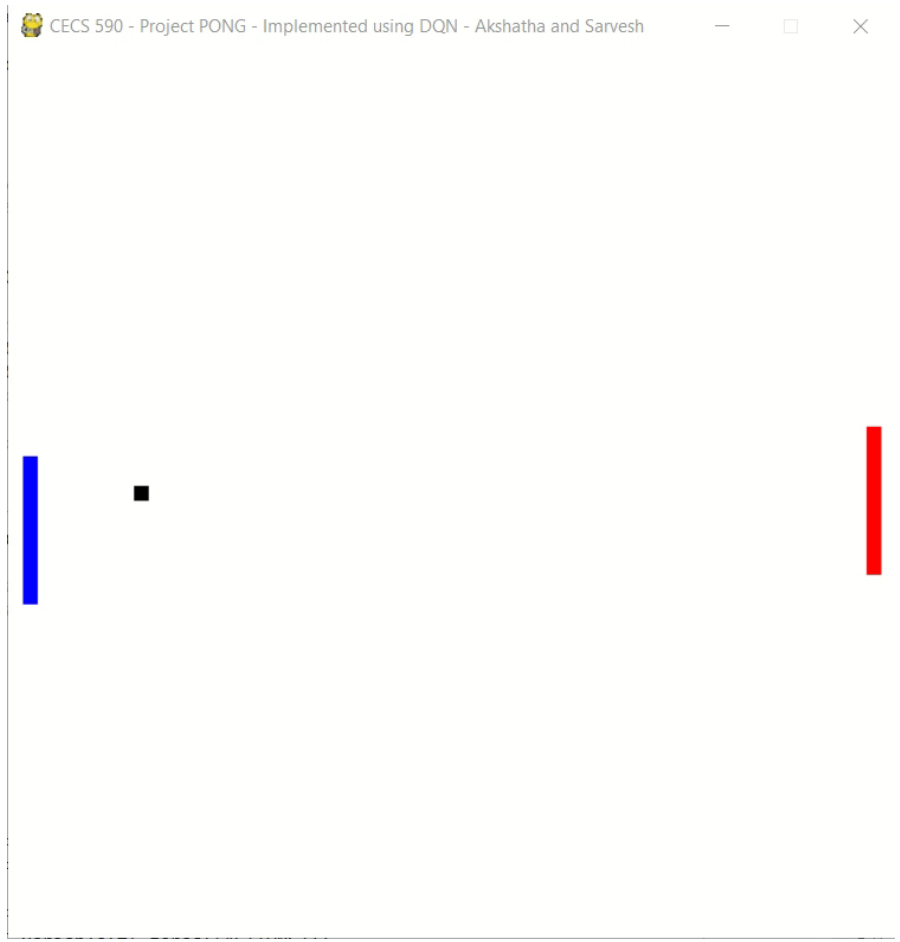
- We didn’t think of following scenarios:
  - Paddle going out of the window
  - Ball moving in wrong direction
- Playing around to find the right set of values for the parameters
- Training the model

Results of Analysis



Demo


CECS 590 - Project PONG - Implemented using DQN - Akshatha and Sarvesh


CECS 590 - Project PONG - Implemented using DQN - Akshatha and Sarvesh

Reference

- “Human-level control through deep reinforcement learning” – DeepMind Reference Paper
- Website- https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf