CS698R - Deep Reinforcement Learning

Instructor: Prof. Ashutosh Modi

Foraging in a field

Group 3

Team members:

Dibyojyoti Sinha (180244)

Milind Nakul (180420)

Prakhar Pandey (180527)

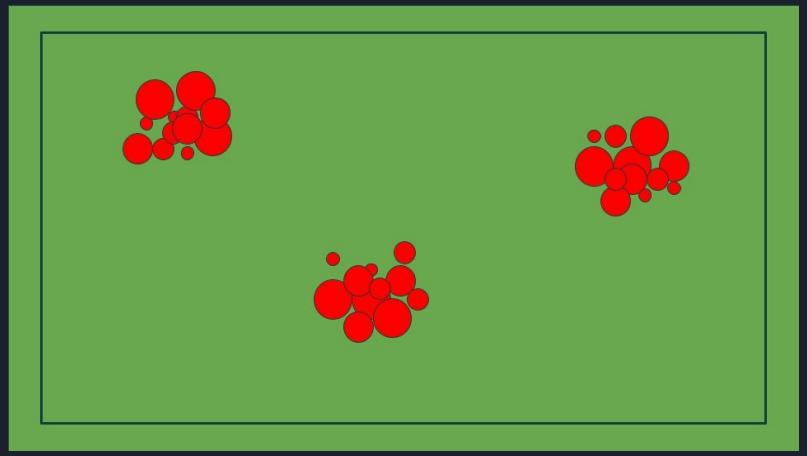
Samarth Varma (180655)

Mentors:

Adithya Anil

Ajita Shree

Problem Description



- Trying to find an OPTIMAL solution for collecting berries from a field
- Berries are distributed in patches
- There are 10 patches distributed randomly on the field
- Depending on their size, there are 4 kind of berries
- Each patch has a fixed number of berries of each size
- Our goal is to collect maximum amount of juice in a given time frame
- Moving on the field will drain our collected juice, while collecting berries will increase it
- The agent can move in 8 directions and can only see a part of the field at a particular point of time
- Bigger berries have more juice (linear relation between amount of juice and berry size)
- The environment is STATIC i.e. the location of the berries, patches will not change

Motivation

- Motivation to solve the problem can be different depending upon what you are trying to achieve
- For a neuroscientist, the motivation can be to:
  - Compare the animal behaviour with that of a machine, to see what are the differences and similarities between how a machine learns compared to an animal or a human. This can help not only to improve our understanding of how human learns and behave, but can also give us some motivation to come up with a completely new idea (like Neural Nets) or algorithms or improve the existing ones
- For an RL enthusiast, the motivation can be to:
  - Compare the learned solution with the optimal solution to see how good is a particular algorithm or how a particular algorithm performs in this problem
- This problem can also be the simplified version of many much more complex problems
  - One of the many problems can be to find the most optimal path to do some work

Environment Details

Overall
- Field size: 20000*20000
- Total time: 5 min

Berry
- Types: 4 depending upon size
- Size: 10, 20, 30, 40
- Shape: square

Patch
- Randomly generated
- Size: 2600*2600
- Count: 10
- 20 berries of each size
- Interpatch distance >= 5000

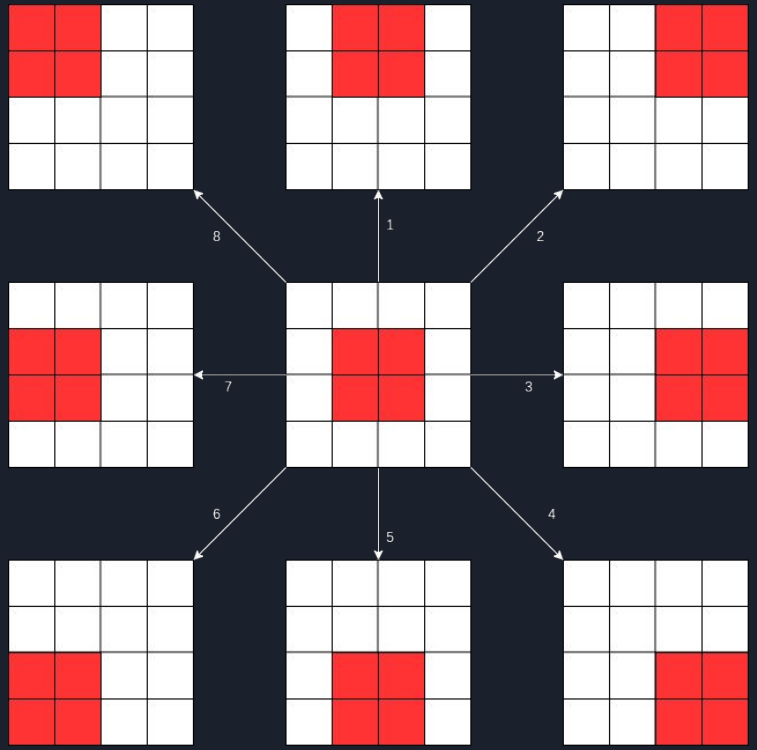
Agent
- Size: 10
- Speed: 400 pixels/sec
- Max steps: 400*(5*60) = 120000
- Action space size: 9
- Observation space size: 1080*1920

Reward
- Initial: 0.5
- Drain rate: 0.5*d/(120*400)
- Reward factor: size/10000

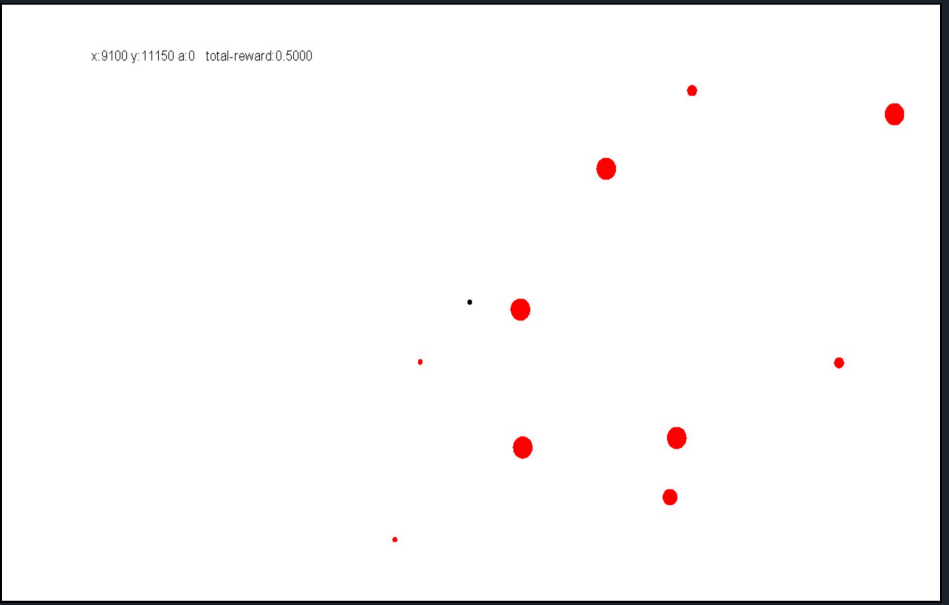
Environment Implementation Details

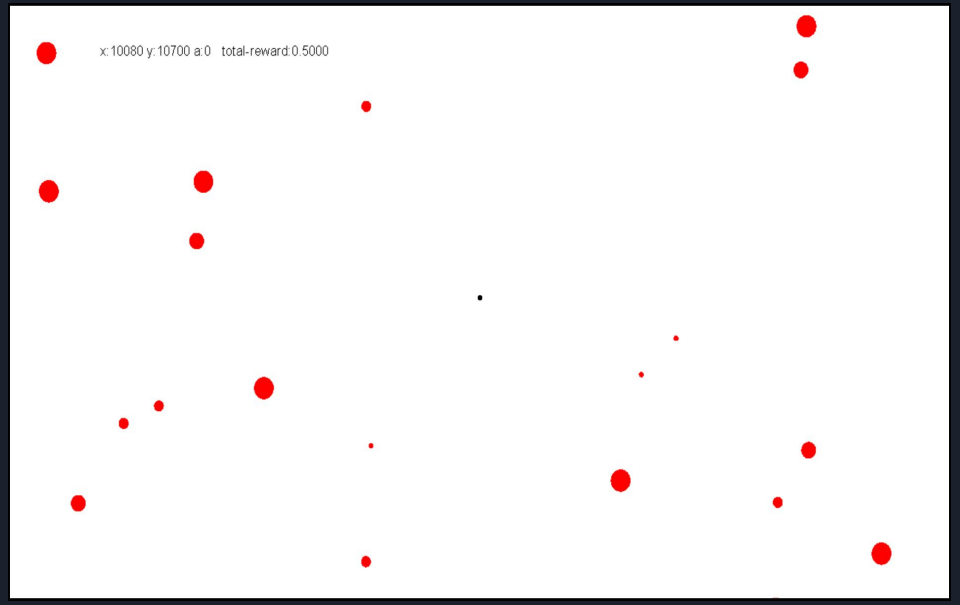
- Randomly generated the location of the patches and the berries inside the patches
- Generated a 20000*20000 int numpy array, which will remain in the memory and represents the pixels of the field
- Pixel values are either 0 or berry size
- After every step, check if the agent hitted any berry (or berries)
- If it has, we get the bounds of the berry using two interval trees and a map (mapping berry ID to its shape and position)
- Remove the berries the agent has hitted from the field and calculate the reward
- Get the observation for the user in form of a 35*5 matrix or an 8*2 matrix
- Instead of taking time, we are ending the task after max_steps = speed*time is reached

Action



Images of the Environment





Our Approach

- First we tried out the Deep Q Networks approach to solve the environment. However the CNN approach used in the DQN paper was computationally infeasible for us to implement since our environment and observation space were very vast
- We then replaced the raw pixels(1920*1080) by a simple (35*5) numpy array which represented the various properties of each observed berry such as the size, x-coordinate, y-coordinate etc.
- This helped us in reducing the computational cost of our model

Our Approach

- The DQN approach has previously been used to solve the Atari games
- Since the problem proposed here is similar in nature to the Atari games, using this approach in our problem makes sense
- We make certain tweaks in the DQN algorithm to make it compatible with our environment
- We also propose a new Neural Network architecture to make it computationally feasible for us to train our model
- The challenges with this approach and the further techniques proposed have been discussed in later slides

Structure of the observation

Structure 1 : A matrix of size 35*5 containing the following information-

Boolean (0/1) specifying if the info stored in this row corresponds to a berry	**Unit Vector**		**Float** specifying the distance from the agent to the berry	**Float** specifying the size of the berry
	Float specifying its component in the X direction	**Float** specifying its component in the Y direction		

Structure 2 : A matrix of size 8*2 containing the following information-

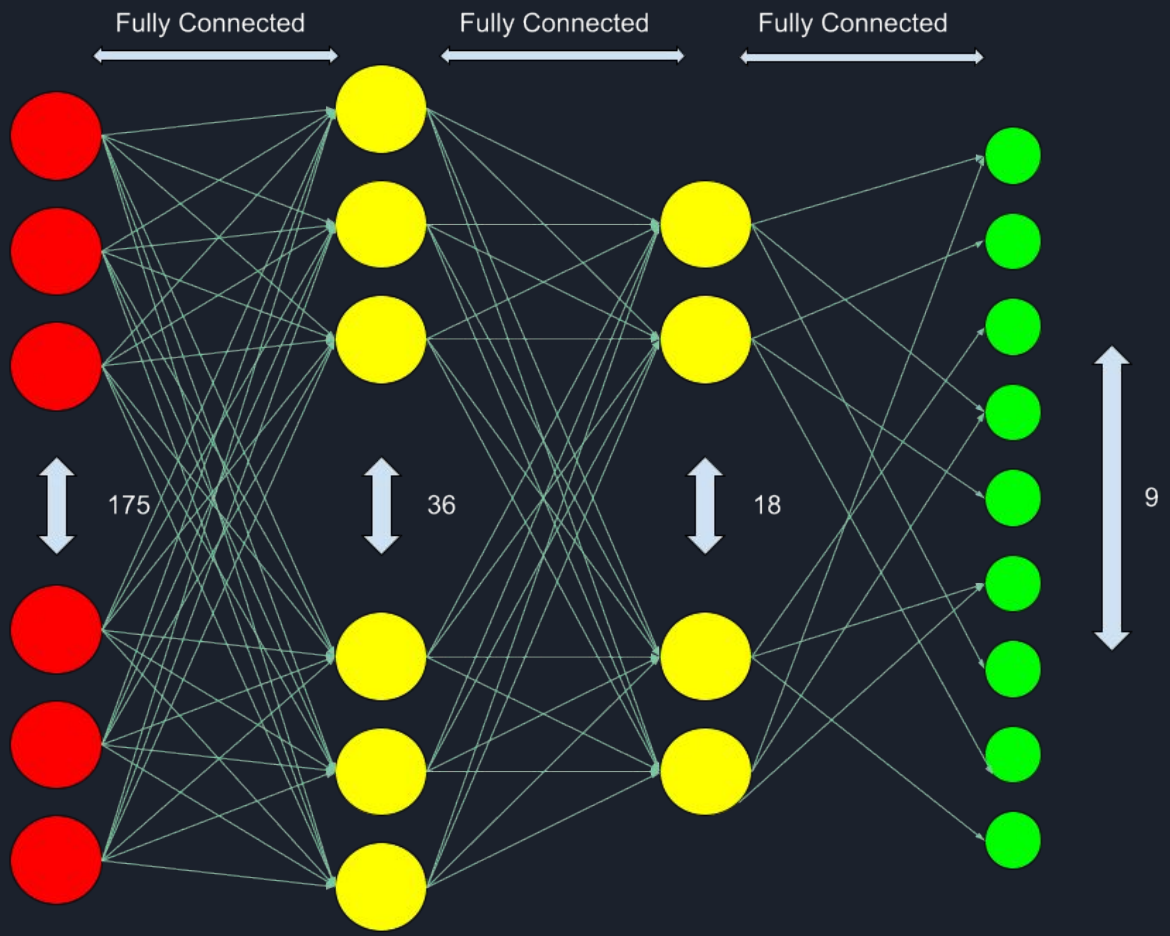
Float Average size of visible berries in the sector	**Float** Centroid of all the visible berries in the sector

Environment Challenges

- Observe that the rewards are very sparse, the berries are separated by a large number of steps, the agent sees lots of steps without positive rewards.
- “Lazy Cat”: trained agent chose to not move at all to maintain the initial 0.5 cumulative.
- “Where are the berries?” Vast empty space between patches. Difficult to explore due to the extremely rare rewards.

DQN Neural Network Architecture

- We pass our 35x5 to a simple feedforward neural network
- The number of hidden layers is 2
- The hidden layers are of sizes 36 and 18 which follows an output layer of size equivalent to the action space, which is 9
- The neural network is made up of simple linear model followed by a ReLu for both the hidden layers.


Fully Connected
Fully Connected
Fully Connected
175
36
18
9

Tackling the Sparse Reward Space

- We observed while rendering the environment that during the initial episodes the agent did not explore enough of the environment and was not able to learn much
- This was due to the fact that our reward space was very sparse and random actions were very unlikely to get the agent to any berry
- We used the following three approaches to tackle this problem:
    - Starting with a Heuristic Policy instead of Random Exploration
    - Using Closest Berry as Motivation
    - Curiosity Search Approach

Heuristic Policy for Exploration

- Segment the observation space into 8 sectors each subtending an equal angle at agent.
- These sectors correspond to the directions in which the agent can move.
- Compute a discounted average of the berry juice for each sector. Discounted by the distance of berry from the agent.
- Move towards the sector with the maximum amount of juice.
- If there is no berry in view take the last action taken.

Motivation Approach

- Augmenting Sparse Reward signal with additional reward signal
- This additional feedback or reward signal is related to the task of collecting the berries in an very optimized way.
- As the agent succeeds in these tasks, it learns about the environment better than in comparison from learning just from the sparse reward signals.
- We propose two motivational rewards - Curiosity Search and Closest Berry
- This approach motivates the agent to explore more states of the environment and thus prevents the agent from getting stuck at a local optimum

Closest Berry

- We find the berry closest that the agent can see.
- The additional reward is given by exp(k * closest_berry_distance)
- The hyperparameter k is taken as -0.001 to -0.01 during various simulations
- This motivates the agent to go closer to the berry and collect it, by ensuring more reward as the agent goes more closer to a berry.

Curiosity Approach

- In this approach in order to motivate the agent to explore new states and actions we keep a novelty reward
- We divide the environment into 100*100 grids
- There is a curiosity reward associated with exploring each grid for the very first time
- The reward function associated with this curiosity is:

$$R_c = \beta R + (1-\beta) I_c$$

- Here R_c represents the curiosity reward, R represents the game reward and I_c represents the indicator function denoting whether the current grid has already been explored. β is the hyperparameter which controls the preference between current game reward and the curiosity reward

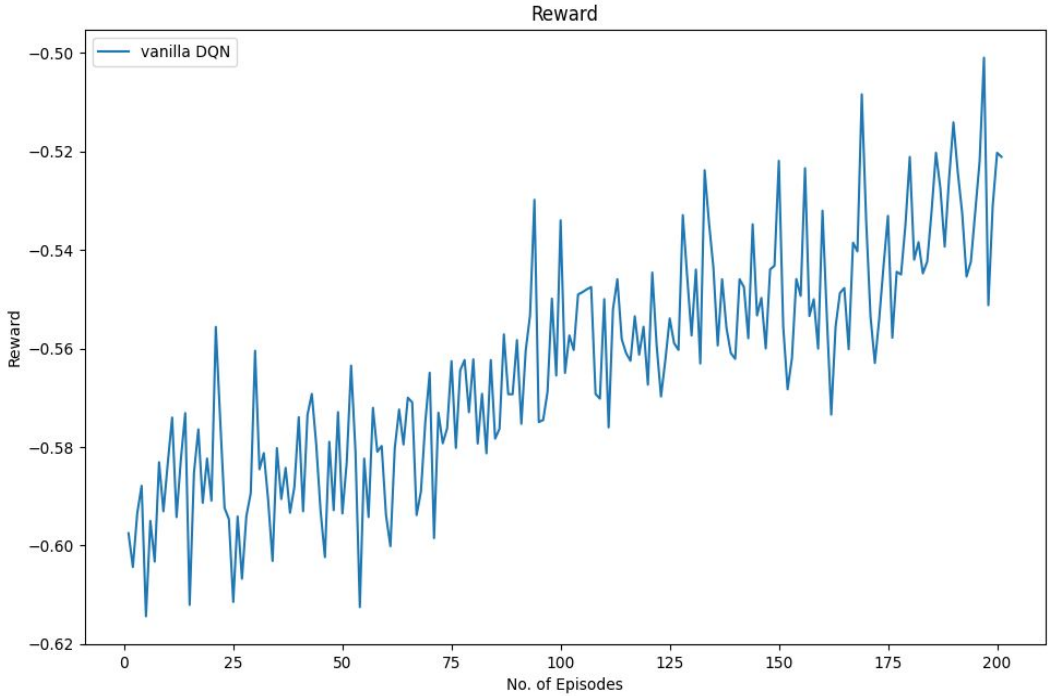
Result of DQN

- We used the DQN algorithm with the random exploration at the beginning and experience replay to train our model
- However this took too many episodes to train and hence the computational cost was very high
- One of the problems with DQN was that our environment was too big and the reward space was very sparse
- The agent did not explore the entire environment and stopped after exploring just a patch of berries

Vanilla DQN

Observations

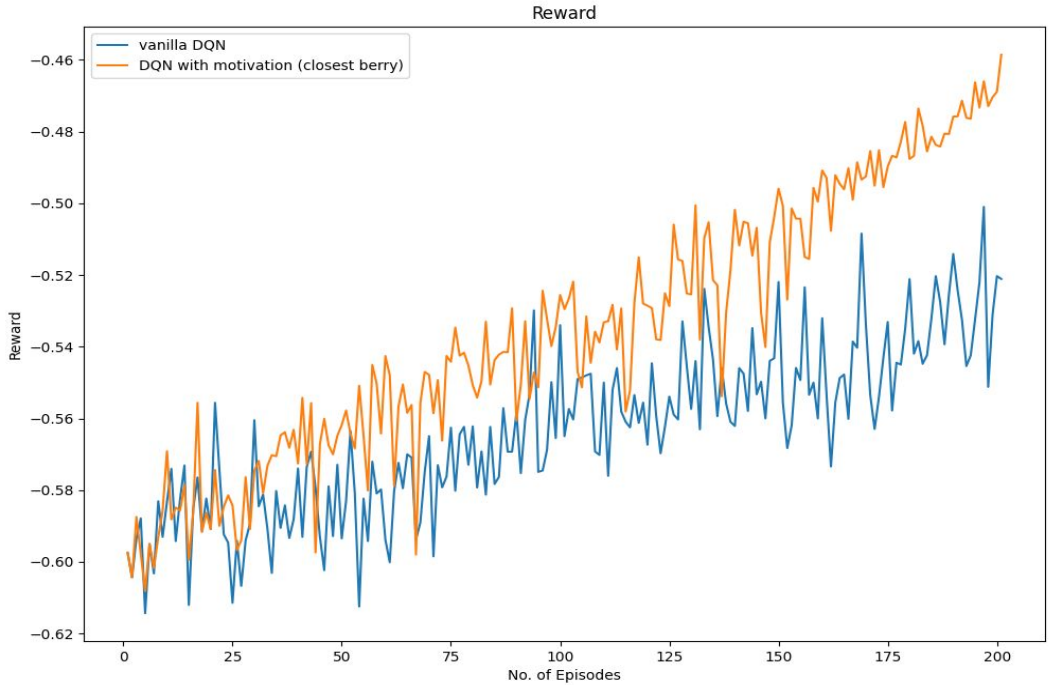
- Vanilla DQN was iterated for 200 episodes.
- The reward improves with every episode.
- Computationally Costly Overall.
- Hyperparameters: $\gamma = 0.999$, Update Frequency = 10, Batch Size = 64, Epsilon Decay from 0.99 to 0.05 in 500 decay rate



DQN with motivation

Observations

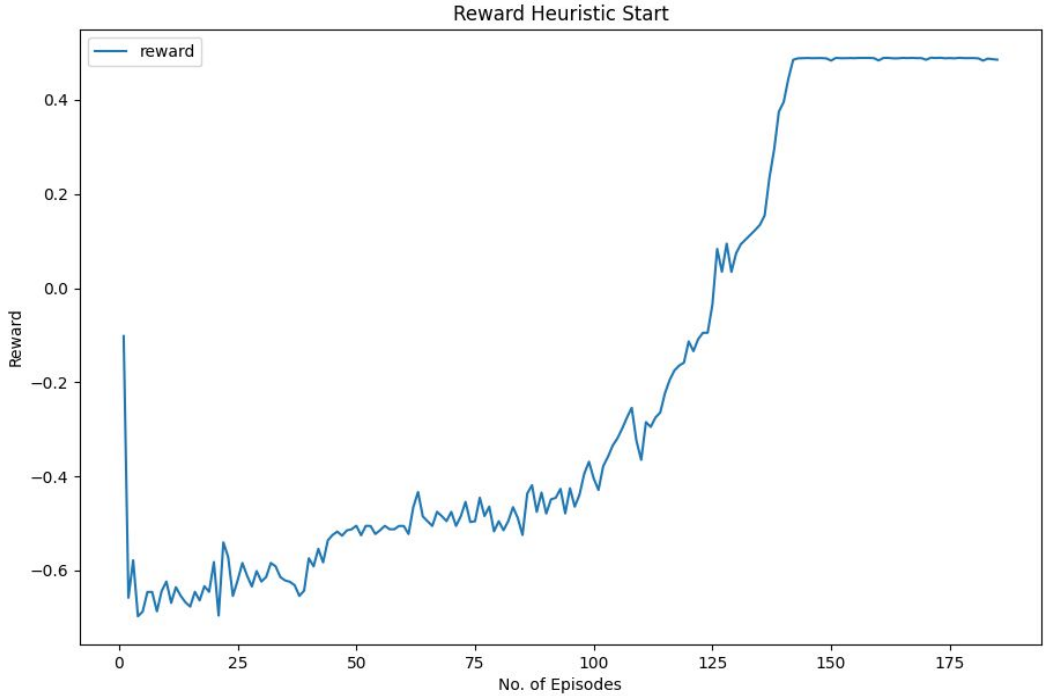
- Trained with 250 Episodes, DQN with motivation still learns slowly, while being slightly better than vanilla DQN.
- Motivation Reward taken is exp(-0.001 * Distance of Closest Berry)
- Computationally Costly Algorithm.
- Hyperparameters: $\gamma = 0.999$, Update Frequency = 10, Batch Size = 64, Epsilon Decay from 0.99 to 0.05 in 500 decay rate



DQN with heuristic starting policy

Observations

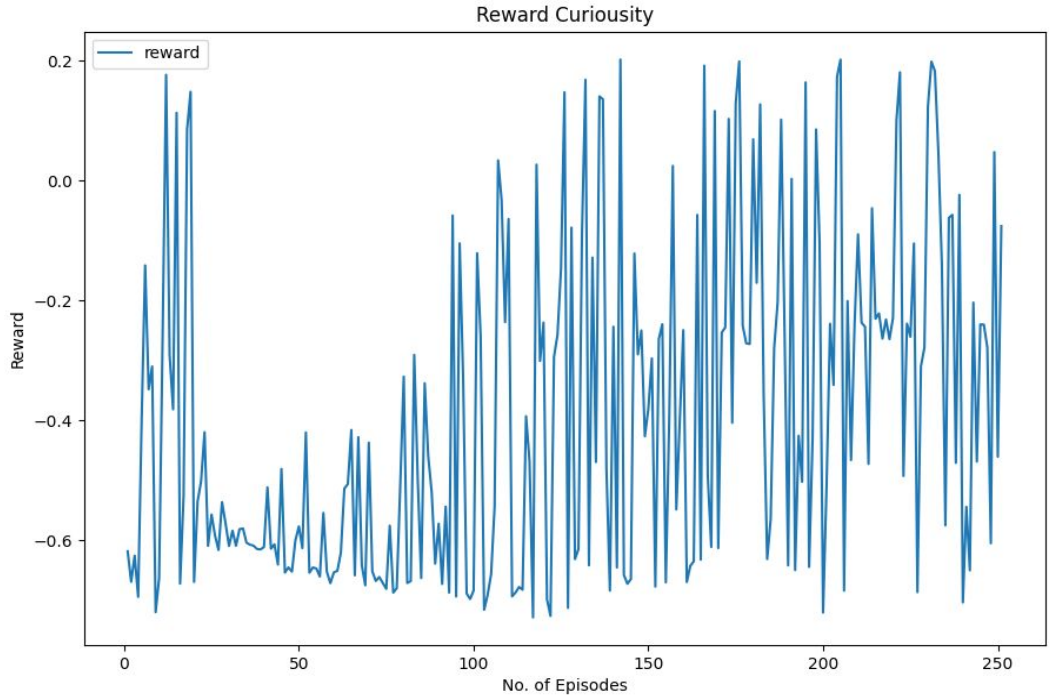
- The starting policy taken was the heuristic policy which gives out an initial cumulative reward of -0.1
- It converges to a cumulative reward of 0.48 in around 150 episodes.
- While rendering the environment, it was seen that the agent did not explore, and instead traversed only in the initial patch and then decided to stop.
- Hence, this is not an optimal solution.



DQN with Curiosity Approach

Observations

- In order to tackle the problem of the agent not exploring the environment much we introduce the curiosity reward in order to motivate the agent to explore more grids in the environment
- However this algorithm is computationally very complex and hence the agent does not learn much in the 200 episodes that it was trained



Challenges

Two main challenges that we faced while solving the problem are:

- The inherent nature of the problem
  - Sparse rewards in the environment
  - Long term credit assignment - Not able to tell which actions lead to the negative (or zero) reward
  - Cancelling out of the random exploration
- Computational problems

Challenge 1 - Sparse Reward

The chances of the agent taking a series of steps (correctly) and ending up getting a reward while exploring (randomly) are pretty low. These chances gets even low when the agent has to do this a number of times so that it can learn how to collect rewards.

When the agent does not get a reward (or gets a negative reward) it is very difficult for the agent to pinpoint the step (or series of steps) that went wrong.

This problem can be compared to the problem of solving the 1980s video game Montezuma's Revenge using Reinforcement Learning in which the agent had to learn the path to the key. DeepMind in their research paper mentioned that they solved this problem by giving the agent reward for exploring the environment (taking inspiration from children).

This problem can also be compared (to some extent) the problem of developing an AI for playing chess using Reinforcement Learning, as it also shares the same challenges of sparse rewards and difficulty in pointing out the wrong move for a negative (or a zero) reward.

One more problem was that we cannot use the vanilla random random actions for exploring the environment as the agent would not go anywhere in long term

Challenge 2 - Computational Problem

The other main problem that we faced while solving this problem was the lack of proper computational resources. An average model takes about 1 day to complete 200 episodes which is the bare minimum we require to test whether the algorithm is working with the chosen hyperparameters. This becomes more complex because we also have to test the model with different hyperparameters to chose the ones with best performance

Research for the Sparse Problem

- 1980s video game Montezuma's Revenge
  - In this game the goal of the agent is to find the key to go to the next level
  - The reward in this game was too sparse as the agent had to take several steps in the right order to reach the key (and get a reward)
  - To overcome this problem, DeepMind used "carrot and stick" approach
  - The idea is to give the agent reward to explore new things in the environment
  - The agent was encouraged to find new sub-tasks for itself like climbing a ladder or jumping over an enemy etc.
  - Rewarding the agent not only after getting the key but also at several occasions which were less obvious but are good in long term so that the agent does not have to take several steps just to realize that all those actions were not optimal is the takeaway we took from this
  - Research paper - https://arxiv.org/pdf/1810.12894.pdf

Contributions

Dibyojyoti Sinha	Environment Rendering and Testing, Curiosity Search Implementation, ,Environment Optimization, Implementation of Heuristic Agent.
Milind Nakul	Base DQN Agent Implementation, DQN Agent with Heuristic start policy, Curiosity Search and related research work
Prakhar Pandey	Environment Implementation, DQN Class Approach and Motivational Reward
Samarth Varma	Base DQN Agent Implementation, Motivational Reward, Rendering, Environment Testing and Result Analysis

References

- Playing Atari with Deep Reinforcement Learning, Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, 2013: https://arxiv.org/abs/1312.5602
- "Deep reinforcement learning with double q-learning.", Van Hasselt, Hado, Arthur Guez, and David Silver, 2016: https://ojs.aaai.org/index.php/AAAI/article/view/10295
- "Dueling network architectures for deep reinforcement learning. ", Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. , ICML 2016: https://proceedings.mlr.press/v48/wangf16.pdf
- http://edu-upenn-wharton-bhlab-foraging-pixi-web.s3-website-us-east-1.amazonaws.com/
- Deep Curiosity Search: Intra-Life Exploration Can Improve Performance on Challenging Deep Reinforcement Learning Problems https://arxiv.org/pdf/1806.00553.pdf
- Deep Recurrent Q-Learning for Partially Observable MDPs : https://arxiv.org/abs/1507.06527