AI Project: Slither

Roee Cates
308298827

Carl Veksler
724637420

Barak Rabenou
203284682

***Abstract*—Q-Learning is a widely used algorithm in Reenforcement Learning. In this project we adopted Q-Learning into different Neural Networks in order to try and create an agent which learns how to play Slither.**

I. Introduction

Slither is a multiplayer variant of snake. Players each try to eat the randomly placed food, while also trying to navigate through other snakes on the game field, while accumulating points. In this project we tried to create successful players, i.e players that can achieve high scores.

II. The Game

A. Goal

The goal of the game is to accumulate as many points as possible, trying to achieve a better score than the other players.

B. Playing Field

The playing field is an nXn grid with a mirroring quality meaning that if a snake crosses a border it will continue from the other side of the playing field.

C. Gameplay

- Like in snake, each player at any given time can choose to go in one of three ways, forward, right or left which are translated to either up, down, left or right depending on the orientation of the snake.
- Score is accumulated by three different actions:

  - Eating an apple

  -"Killing" a snake

  - Getting "killed"
- Eating an apple adds points to the point total and simultaneously, increases the length of the snake that ate it.
- "Killing" a snake grants points. This is done by causing the head of another snake to collide with any part of your snake.
- Getting "killed" deducts points. This happens when the head of your snake collides into another part of another snake.
- In case of a head to head collision, both snakes are killed both snakes gain points for "killing" and then lose points for getting "killed"
- Once a snake is killed, it resets to its initial length and spawns in a random location on the board

III. The Problem

We wanted to create a player which would be able to play the game successfully, meaning a player which would be able to achieve high scores in the game We wanted this player to be able to achieve all of this in any given settings. Also, we implemented a cool GUI :).

IV. The Players

In order to achieve our goal of creating a successful slither player, we implemented a variety of agents that interact with our game

A. Human Player

A human decides what moves to perform

B. Random Player

The program chooses a move at random

C. Greedy Player

The greedy player acts according to a heuristic which attempts to eat the closest apple while avoiding collisions with snake parts so not to be killed

D. Deep Q-Learning Network (DQN) Player

In this methodology we use a deep neural network (NN) model to approximate the Q-Function. In our project we implemented two types of this model:

- CNN Player - Convolutional Neural Network Player - this model receives a series of binary images, which represent the game state, as its input
- NN Player - Neural Network Player - this model receives a number of features which are derived from the game state as the input for its learning

V. The Approach

This problem falls naturally into the reenforcement learning framework since there is a reward for "good" actions and a penalty for "bad" actions. This led us to adopt the Q-Learning algorithm, however, we realised that the most basic form of Q-Learning would not quite cut it, due to the vast amount of states. For example a 10X10 board with only one player has 100 tiles which can be either an apple, a part of the snake, or empty thus resulting in 3^{100} possible states. Therefore we can conclude that the number of states is exponential in relation to the size of the playing field. Due to this, we decided to implement a Deep Q-Learning (DQN) algorithm which is a variation of the basic Q-Learning algorithm in which we use a deep neural network (DNN) to approximate the actual Q-Function. This is a good solution

because NN models in general and specifically the CNN models have shown great results approximating complicated functions given sufficiently large training data which we can extract with ease using a simulation of the game. We implemented two types of models for this. The CNN model and the NN model.

Furthermore, we used the following framework for both models. The model to predicts the score for each state-action pair as opposed to a framework where the model predicts a vector of scores representing the score for each action given a specific state. We implemented this by changing the orientation of the state according to the considered action (we rotated the state so that the north or up direction matched the direction the snake would move after taking the action).

VI. Implementation

Our models were implemented using an online-policy approach, which means that our models learn and improve as the play the game as opposed to learning from static data.

Also, we adopted an epsilon-greedy mechanism with $\epsilon = 0.001$ which showed similar results to $\epsilon = 0.1$ due to the fact that the game is quite stochastic (other players may move differently in similar states and the food spawns randomly on the game board). This mechanism was crucial for our model to avoid getting stuck at local minimums.

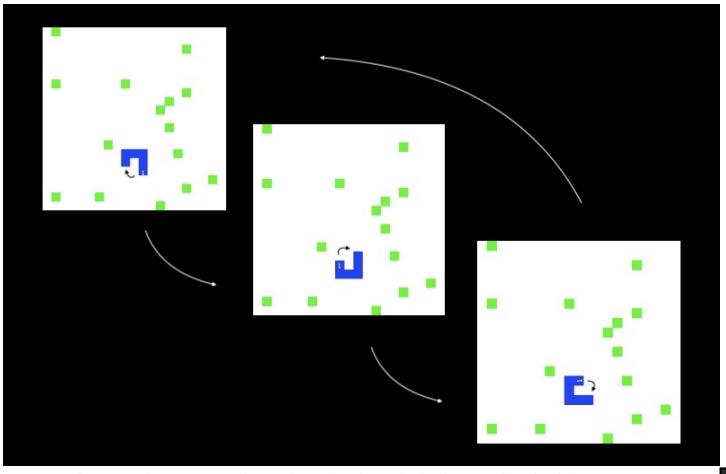


Figure 1 - here we see the snake going in a circle due to a local minimum

For both models we used the Mean Squared Error (MSE) loss function where we calculate the difference between the predicted Q-value and Bellman Equation target Q-value.

$$Q^*(s,a) = E[r_{t+1} + \gamma \max_{a' \in A_s} \{Q^*(s',a')\}]$$

Figure 2 - Here we see the Bellman Equation

A. CNN - Convolutional Neural Network

The CNN model demonstrates good results on image related tasks mainly due to the fact that it is highly expressive with a relatively low model complexity which exploits the spatiality of the images received as inputs (the same convolution weights are applied across the processed image given as input). Due to all of this we chose the CNN model to approximate the Q-Function.

1. Inputs

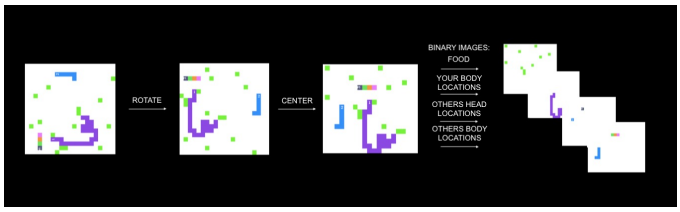


Figure 3 - the preprocessing of the input image into 4 different binary images

- Our first obstacle was how to encode the mirroring property of the game board into the input. This was solved by centering the game state according to the location of the head of the snake (specifically we chose an odd dimension for width and height in order to make it so different actions would have the same input shape after rotating the state, this could also be solved by using part of the playing field and not all of it).
- The next obstacle was how to encode the orientation of the head into the input state. We solved this by "normalizing" the state according to the direction the head was facing (i.e. we rotated the board so that the head was always facing up).
- Third, built the features by modelling the game board into 4 binary images. Each image gives the following information: food, our players body, other players heads and other players bodies.

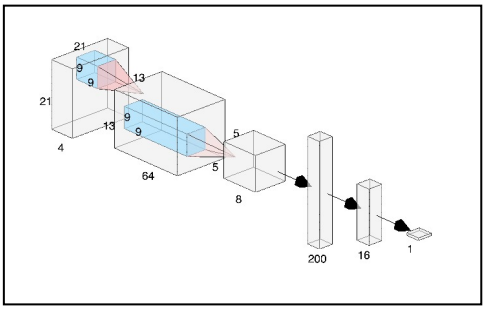


Figure 4 - Here we see a visual representation of the CNN architecture

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 13, 13, 64)	20800
activation_1 (Activation)	(None, 13, 13, 64)	0
conv2d_2 (Conv2D)	(None, 5, 5, 8)	41480
activation_2 (Activation)	(None, 5, 5, 8)	0
flatten_1 (Flatten)	(None, 200)	0
dense_1 (Dense)	(None, 16)	3216
activation_3 (Activation)	(None, 16)	0
dense_2 (Dense)	(None, 1)	17

Total params: 65,513
Trainable params: 65,513
Non-trainable params: 0

Figure 5 - Here we see a text representation of the CNN architecture

2. Architecture - Similarly to the state-of-the-art CNNs our CNN architecture consists of several blocks of convolutional layers followed by a ReLu activation layer. After that, we have two blocks of a fully connected layer followed by a ReLu activation. However, our architecture differentiates itself by having a decreasing number of kernels in each convolutional layer throughout the network, as opposed to most cases where the number of kernels increases throughout. The rationale behind this is due to the fact that we preprocessed the input into four binary heat maps which reduce the overhead of processing the features in the first layers of the network.

B. *NN - Neural Network*

We wanted to explore the ability of this model to learn and explore given specific features which we deemed relevant. We used a relatively shallow model consisting of two fully connected layers which is known to represent simple functions such as “and”, “or" and other similar functions which are suitable for processing input features

1. Inputs

We chose the following features:

• binary feature representing if an action will result in food being eaten

• binary feature representing if an action will result in the snake colliding with another snake

• an int which indicates how much food is ahead of the snake in a 7×7 window

• an int which indicates how many obstacles are ahead of the snake in a 7×7 window

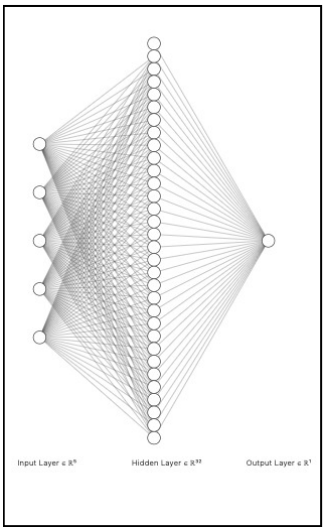


Figure 6 - Here we see a visual representation of the NN architecture

2. Architecture - This architecture of two fully connected layers followed by a ReLu activation is able to express simple functions such as “and”, “or”, “xor”, etc. We concluded that this is suitable to process handful of features which we chose, which were relatively intuitive.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 32)	160
activation_1 (Activation)	(None, 32)	0
dense_2 (Dense)	(None, 1)	33

Total params: 193
Trainable params: 193
Non-trainable params: 0

Figure 7 - Here we see a text representation of the NN architecture

VII. Settings

A. *Game Settings - These settings are configurable to the user’s satisfaction*

- Board Size - $n \times n$
- Starting Snake Length
- Points Rewarded by Eating an Apple
- Points Rewarded for “Killing” Another Snake
- Points Deducted for Getting “Killed”
- How Much Eating an Apple Increases the length of a snake
- How apples are present on the board at any given moment

B. *Hyper-Parameters*

1. RL Hyper-Parameters
   - Decay Factor - γ
   - Rewards - same as scores, the goal is to achieve the highest possible score
   - Epsilon-Greedy - $\epsilon = 0.0001$
2. Deep Learning Parameters
   - “Adam” optimizer using a 0.0001 learning rate.
   - Batch size - 64.

VIII. Metrics

Since this game is everlasting, we needed to find a way to evaluate how well our agents performed. We chose several metrics. The main and most intuitive metric, is the score per iteration. Additionally we also measure how much food is eaten per iteration, how many snakes it killed per iteration and how many times it was killed per iteration.

For the DQN models we also added a loss metric which measures how well the model converges to the Bellman Equation.

IX. Results

Since we have a lot of parameters (game related and model related) we had to focus on specific cases and settings.

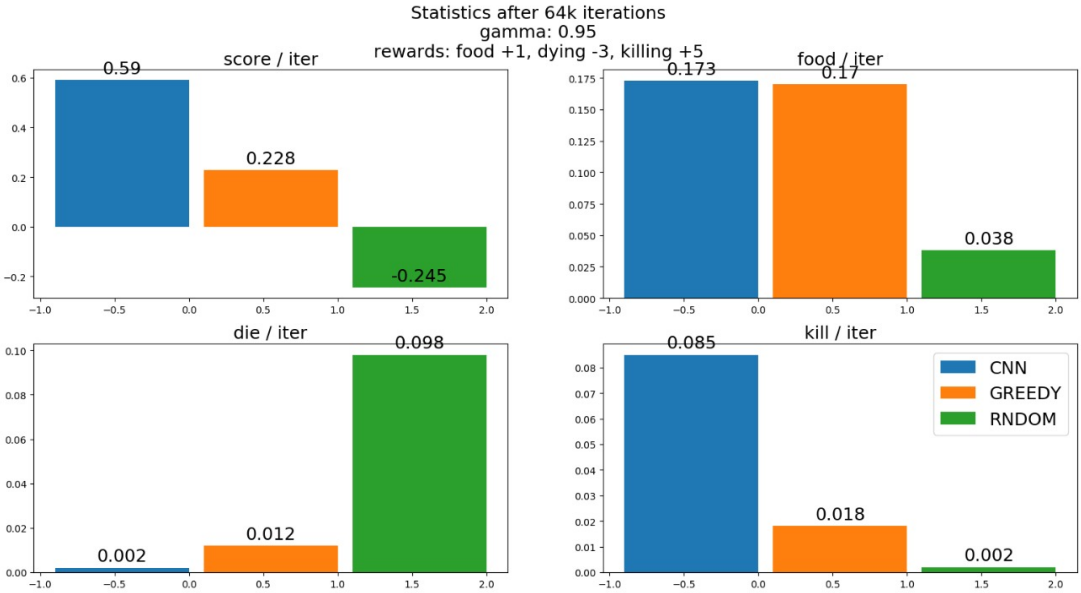
We fixed the game board to 21x21 squares which allow the players to demonstrate their learned strategies while keeping computational complexity quiet low to enable fast model training.

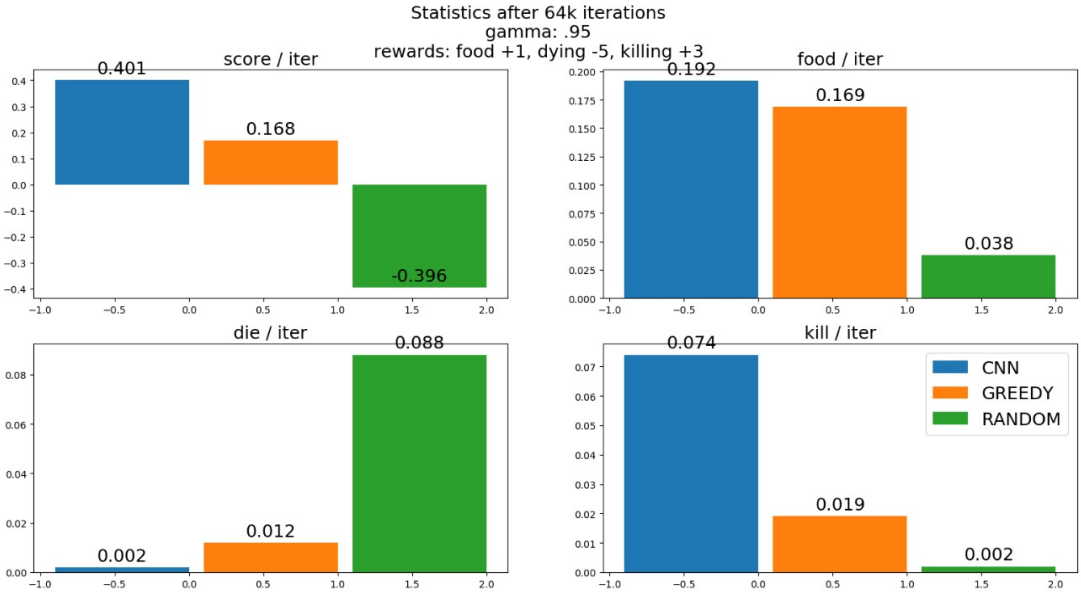
We fixed the number of apples in the game to 15 which doesn’t overfill the game board while allowing each player to reach food that is close by, even if playing against a perfect player - thus enabling and accelerating the training process.

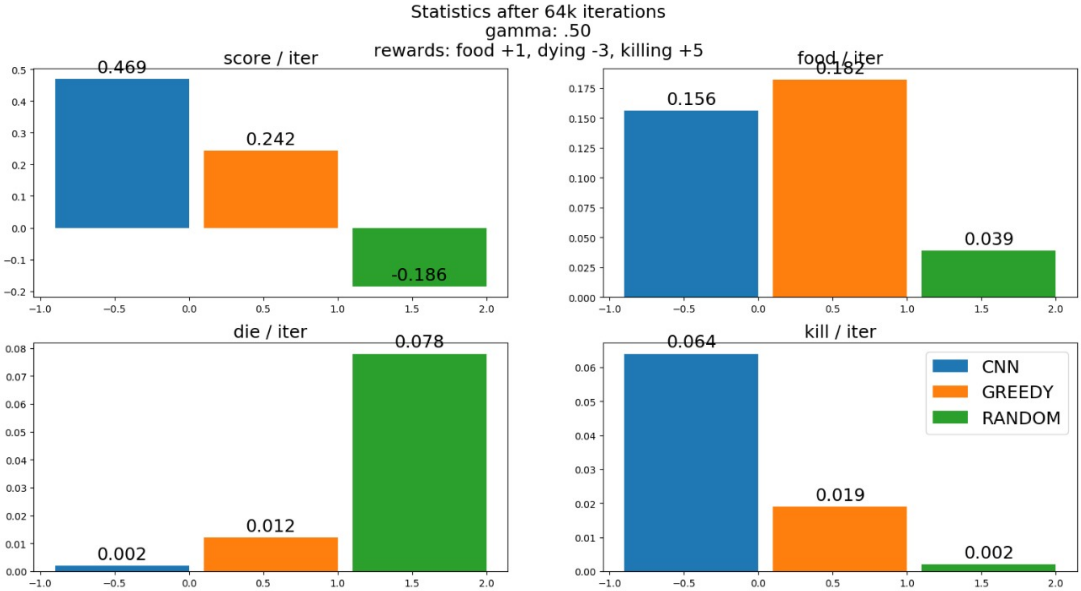
We also chose to have 2 more players on the game board - the Greedy player and the Random player.

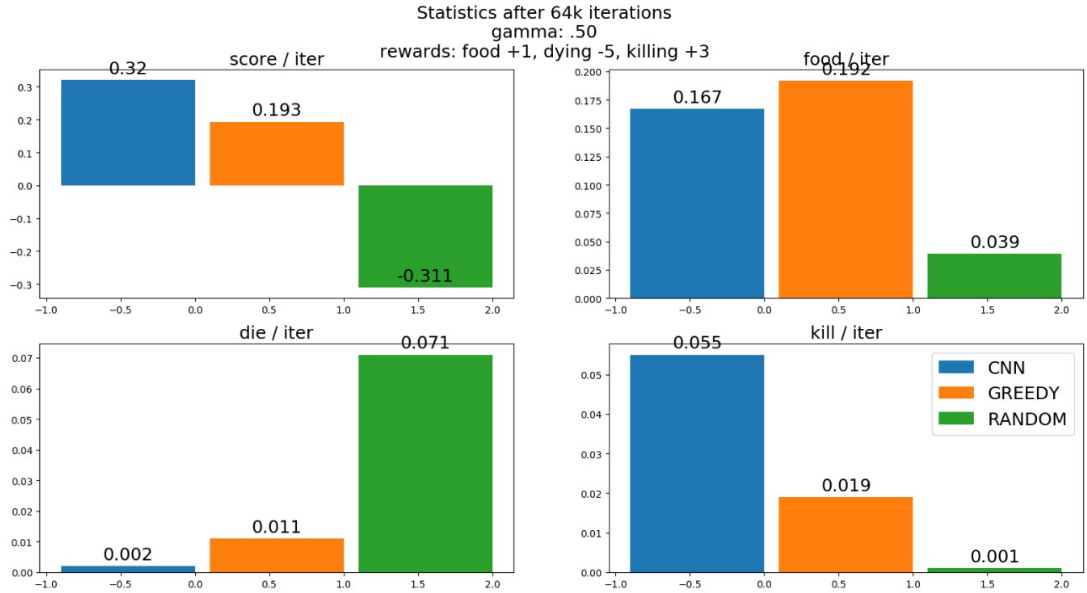
We now present the results of different RL parameters: decay factor (γ) and rewards (scores of the 3 events - eating, killing and being killed).

A. Results for CNN Player:

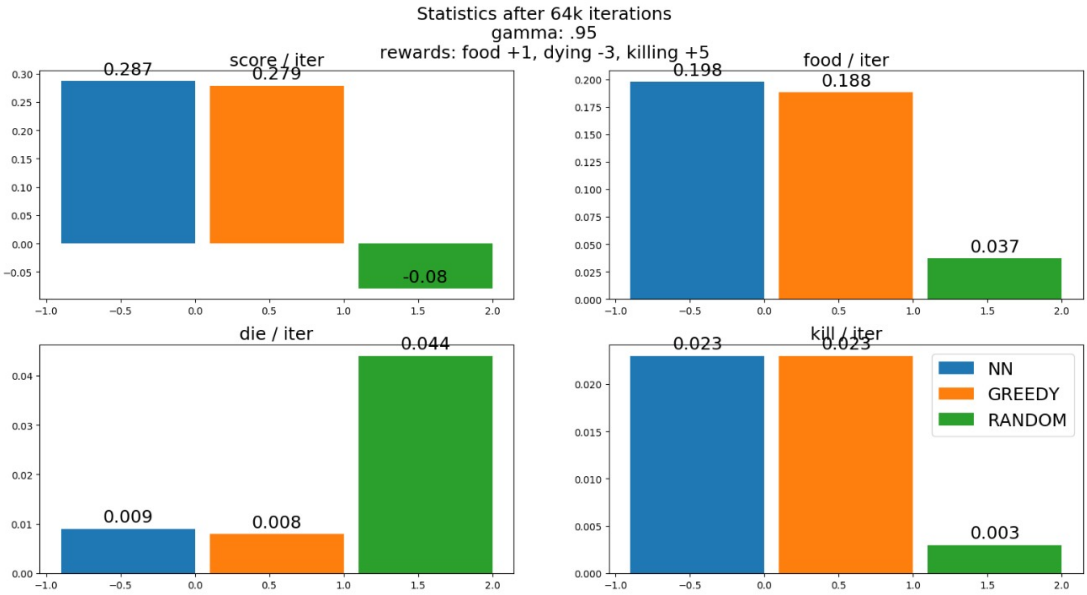


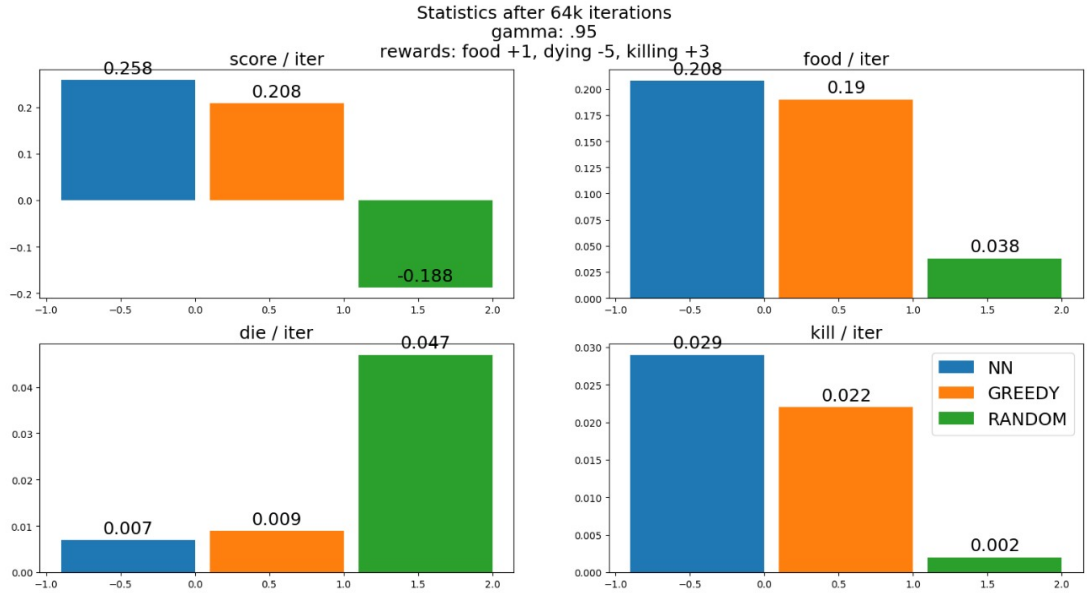


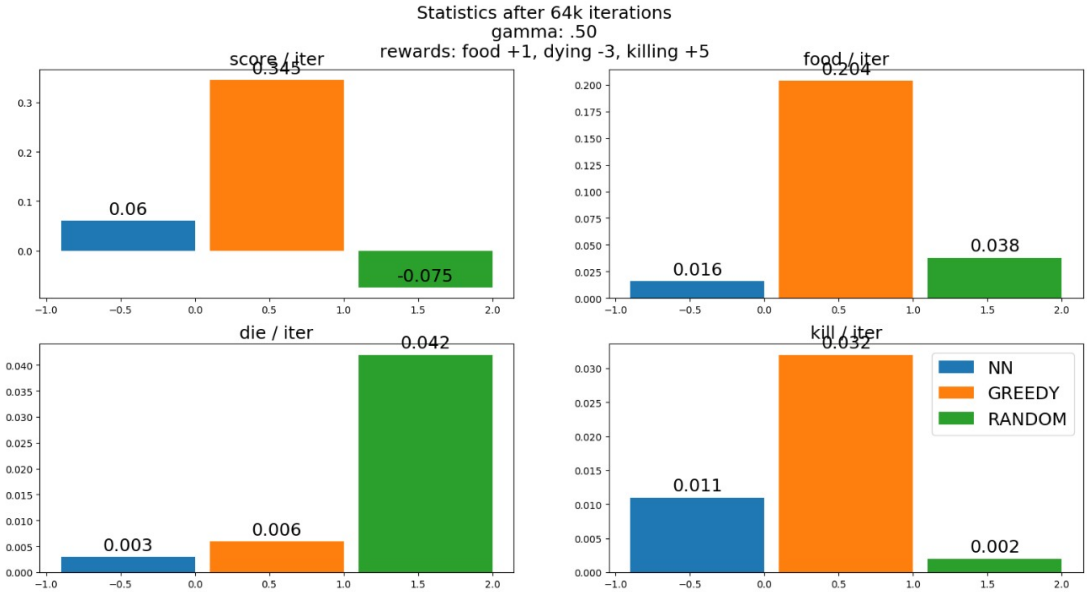


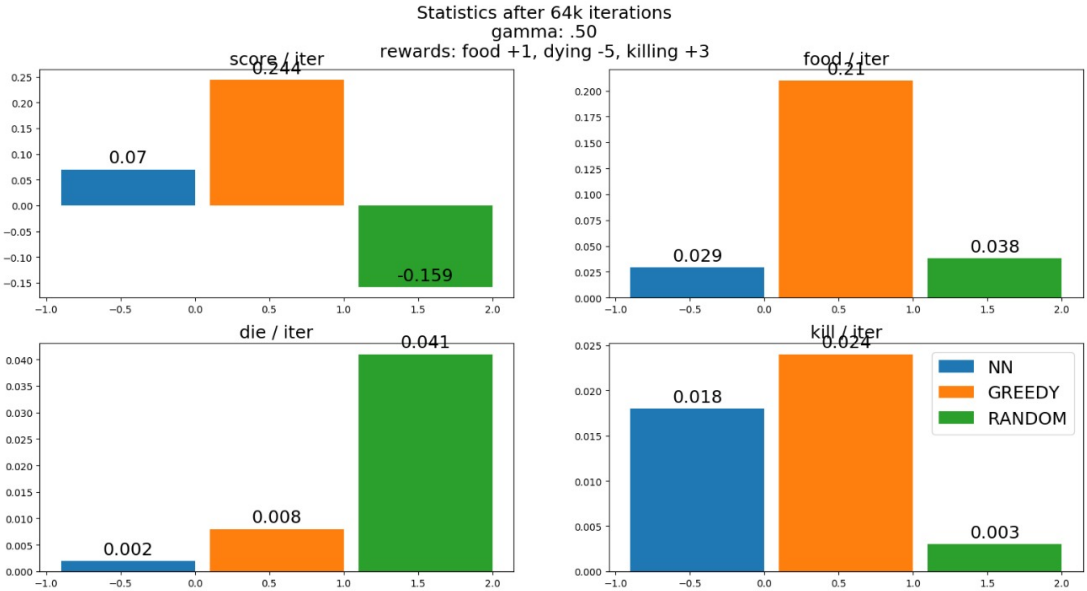


B. Results for NN Player:

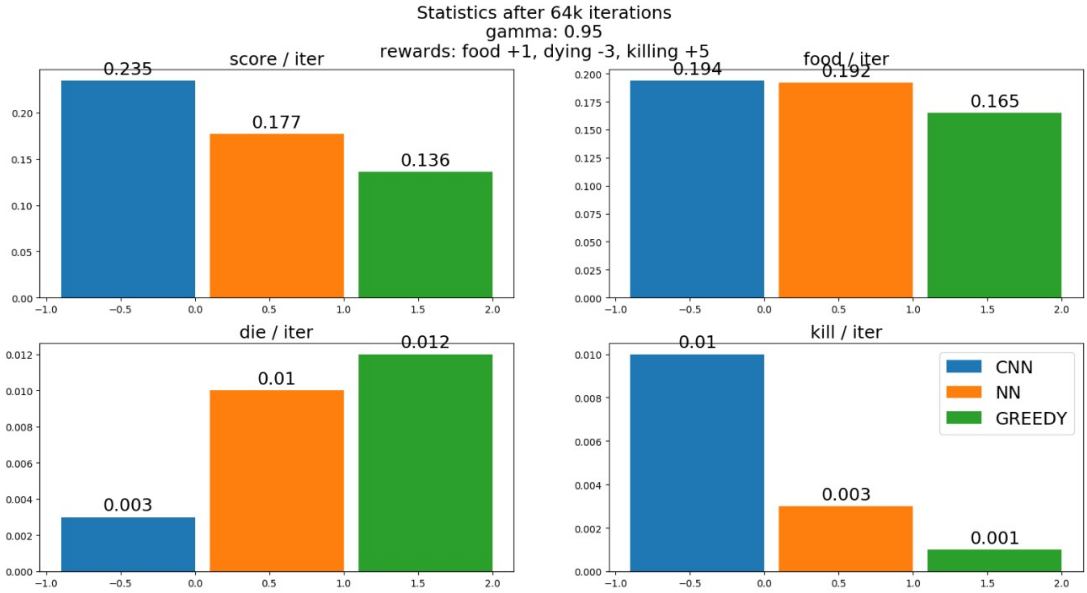


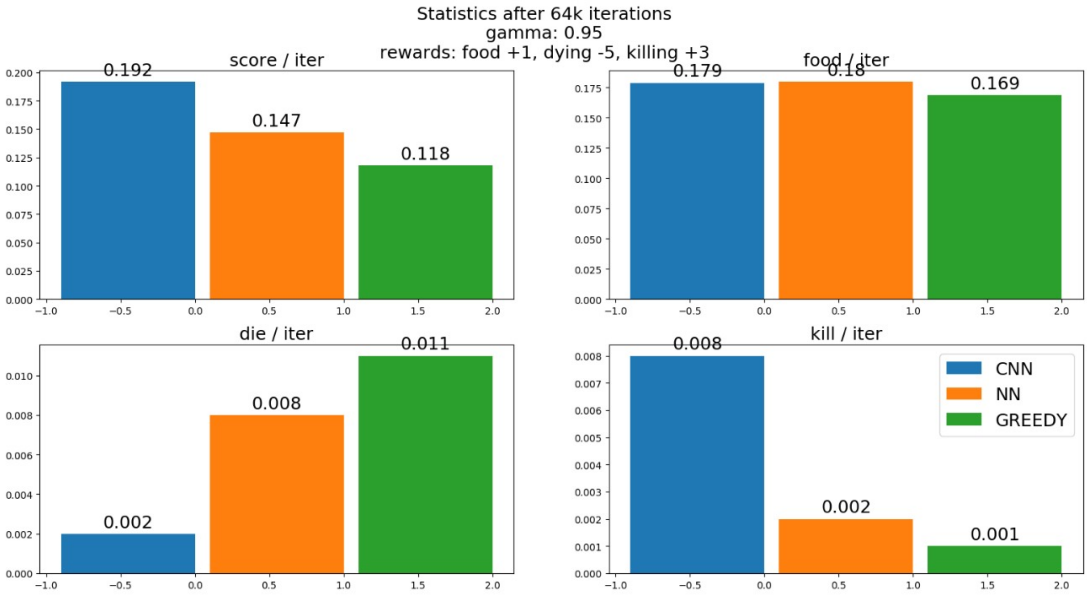

Statistics after 64k iterations
gamma: .95
rewards: food +1, dying -5, killing +3
score / iter
0.258
0.208
-0.188
food / iter
0.208
0.19
0.038
die / iter
0.007
0.009
0.047
kill / iter
0.029
0.022
0.002
NN
GREEDY
RANDOM


Statistics after 64k iterations
gamma: .50
rewards: food +1, dying -3, killing +5
score / iter
0.06
0.345
-0.075
food / iter
0.016
0.204
0.038
die / iter
0.003
0.006
0.042
kill / iter
0.011
0.032
0.002
NN
GREEDY
RANDOM


Statistics after 64k iterations
gamma: .50
rewards: food +1, dying -5, killing +3
score / iter
0.07
0.244
-0.159
food / iter
0.029
0.21
0.038
die / iter
0.002
0.008
0.041
kill / iter
0.018
0.024
0.003
NN
GREEDY
RANDOM

C. Results for CNN Player vs NN Player:





X. Analysis

A. Analysis of the effect of the γ parameter:

• We can see in all cases that a higher value of γ yields better results on both models.

• It is also apparent that the effect of the γ parameter is much more subtle using the CNN model as opposed to the NN model

• The higher death rate on a higher γ value in the NN model can be attributed to the snake being longer due to it eating more, making it harder to navigate the board without dying.

B. Analysis of the effect of the rewards ($\gamma = 0.95$):

1. CNN result analysis

• We can see that given different rewards for each event, the CNN player converges to different strategies. In the settings where killing rewards 5, eating rewards 1 and dying deducts 3, the CNN player eats less frequently, however it kills snakes more efficiently as opposed to the settings in which killing rewards 3, eating rewards 1 and dying deducts 5 where the CNN player eats more and kills less.

2. NN result analysis:

• We can see that as the point deduction for dying grows, the NN model dies at a decreasing rate

• On the other hand, receiving a higher reward for killing actually resulted in the NN player killing less. Although this feels wrong, it happens because our NN model does not receive any input which can indicate to the model on how to kill. Therefore, the difference is equivalent to adding noise to the results.

C. CNN vs NN result analysis:

• We can see that both models learn how to eat food equally as well, however the CNN model learns how to kill other snakes much more efficiently in all of the cases.

• Also, the NN model dies a lot more than the CNN model

• Note that both models achieve better scores than our greedy player

XI. Conclusions

1. As we know, the decay factor (γ) determines the value of future rewards, the higher the value γ receives, the more value future rewards hold with the extreme case being when $\gamma = 1$ meaning future rewards hold as much value as immediate ones. We can see this using the following calculation: $2^{-5} = 0.03125$. This is negligible when compared to 0.95^5 meaning that rewards that are 5 steps away are not taken into consideration as much when using a smaller γ value. In our analysis, focused on two demonstrative gamma values: 0.95 and 0.5, one can see that the 0.95 value showed better results for both models compared to the 0.5 value. Therefore, we conclude that for this game a long-term planning strategy yields better results than short-term ones.

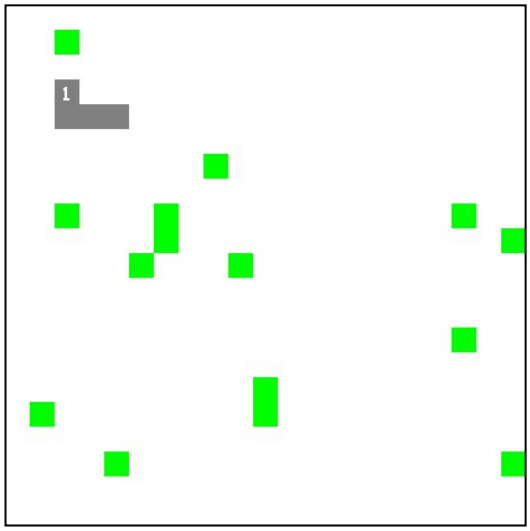


Figure 8 - Here we see the snake using a short-term strategy, trying to eat the close food instead of going for the cluster of apples which are further away

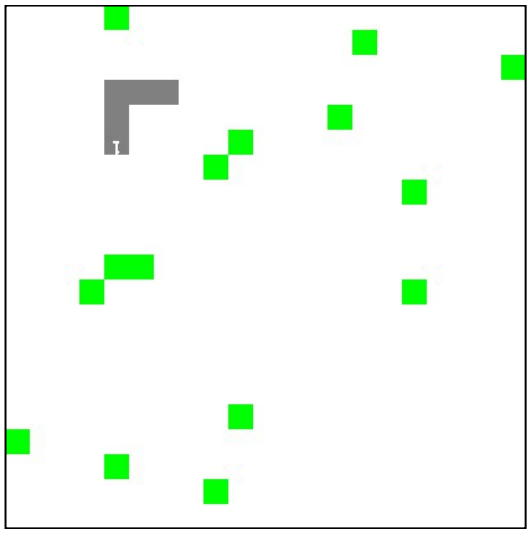


Figure 9 - Here we see the snake using a long-term strategy, trying to go for the cluster of food instead of going for the closest available food

Also, when examining the difference in affect the γ drop-off has on the two models, we can see that it affects the CNN much less. Thus, we can conclude that the CNN is better suited to devise short-term strategies than the NN.

2. It is easy to see that the CNN model converges to different strategies that fit the given set of rewards successfully as opposed to the NN model. This can be attributed to the CNN being a more expressive model than the NN model. The NN's expressive potential is limited because it does not have enough features and therefore it cannot express more complex strategies. On the other hand, the CNN model converged to a superior strategy than the NN model. We can attribute this to the fact that the CNN specialises in exploiting the spatial property of the game. We can conclude that this model successfully learns how to play using the raw given data despite the fact that the game has a vast state space (similar to the calculations before and using a board of size 21×21 we get a state space of size $\approx 3^{441}$.

XII. References

Slither- http://slither.io (The game our game is based on)