Design

State Representation

NOTAIL mode: I initially included snake position, snake direction, apple position and wall locations. Wall locations consist of 4 values (one for each of 4 directions), which are 1 if there is a wall in that direction and 0 if not. This representation was not bad for keeping the snake alive, but the snake was only drawing circles and was not eating many apples. For this reason, I added 2 more values to the state representation: apple direction and apple distance. Also made a few modifications in the reward function (see below). After these changes, results were better. Final state representation for NOTAIL mode:

Variable	**Value**
Snake row	Row (y) coordinate
Snake col	Column (x) coordiante
Apple row	Row (y) coordinate
Apple col	Column (x) coordiante
Apple distance	Manhattan distance from snake head to apple
Snake direction	0 | 1 | 2 | 3
Apple direction	0 | 1 | 2 | 3 | -1
There is wall above	0 | 1
There is wall on left	0 | 1
There is wall below	0 | 1
There is wall on right	0 | 1

CLASSIC mode: I did not make many differences on the state representation because current version was already sufficient, and there is not much difference between NOTAIL and CLASSIC mode. Of course the one important difference is snake body. Therefore I changed the "wall locations" in the previous representation to "obstacle locations", which includes both walls and the snake body. Final state representation for CLASSIC mode:

Variable	**Value**
Snake row	Row (y) coordinate
Snake col	Column (x) coordiante
Apple row	Row (y) coordinate
Apple col	Column (x) coordiante
Apple distance	Manhattan distance from snake head to apple
Snake direction	0 | 1 | 2 | 3
Apple direction	0 | 1 | 2 | 3 | -1
There is an obstacle above	0 | 1
There is an obstacle on left	0 | 1
There is an obstacle below	0 | 1
There is an obstacle on right	0 | 1

TRON mode: I initially tried a state representation without apple information (position, direction, distance). The logic of this reasoning: This is the hardest mode and there are more obstacles than the other modes. If the snake goes for the apple, it might die more quickly (because it always tries to approach the apple) so it should focus on surviving only, and reducing state representation size might help the snake learn safe routes faster. For this state representation, I also configured reward function accordingly (see below). However, this did not go as I expected and results were very bad. I tried changing Snake direction from a single value to 4 boolean values similar to obstacle locations. This did not have a good result either. Then I set it the same as classic mode and result were a little better. Not as good as Classic mode but it could survive for 10 steps on average. Final state representation for TRON mode:

Variable	**Value**
Snake row	Row (y) coordinate
Snake col	Column (x) coordiante
Apple row	Row (y) coordinate
Apple col	Column (x) coordiante
Apple distance	Manhattan distance from snake head to apple
Snake direction	0 | 1 | 2 | 3
Apple direction	0 | 1 | 2 | 3 | -1
There is an obstacle above	0 | 1
There is an obstacle on left	0 | 1
There is an obstacle below	0 | 1
There is an obstacle on right	0 | 1

Rewards

Initially I designed the reward function such that there is a high penalty for dying, a small reward for surviving and an intermediate amount of reward for eating an apple. After observing that the snake does not eat many apples and only tries to survive by moving in circles, I tried adding a penalty for not eating an apple for a long time. For this, I counted the steps since last apple and if it is greater than some threshold, I added some penalty. This had better results but was not sufficient. So I changed the function to reward the snake if it approached the apple and have a penalty if it moved further away. This worked well for NOTAIL and CLASSIC modes so I used this function for them. Final reward function for NOTAIL and CLASSIC modes:

Condition	**Reward / Penalty**
Snake hit an obstacle	High Penalty
Snake ate an apple	Medium Reward
Snake got closer to the apple	Low Reward
Snake got further from the apple	Low Penalty
Snake survived	Low Reward

For the TRON mode, for my initial design of state representation (no apple), I designed the reward function such that there is no extra reward for eating apple, there is only a high penalty for dying and a reward for surviving. Since this method was not good enough and I changed the state representation, I re-added the apple eating reward, but I left out getting closer reward in order to prevent the snake from approaching the apple blindly; and increased surviving reward to make it focus on surviving only. Final reward function for TRON mode:

Condition	**Reward / Penalty**
Snake hit an obstacle	High Penalty
Snake ate an apple	Medium Reward
Snake survived	Medium Reward

Hyper Parameters

- I trained the model with different **number of steps** for example 10000, 50000 and 100000. Generally more steps yielded better results. However, since the learning speed decreases after a while, usually there was not a big difference between 100k steps and 50k steps. Also in tron mode, loss even started increasing after some point. Therefore I mostly trained the model with 50k steps.
- For **memory**: I tried different memory sizes (between 1k and 10k) and different batch sizes (between 32 and 1k). Considering speed and performance trade-off, 5000 as memory size and 500 as batch size worked well for me, yielding sufficient results.
- I tried different values for **learning rate** too. When I increased it (for example 0.1) the training was usually more unstable (loss having ups and downs). If I increase it too much, (for example 0.5) the model cannot even learn and average reward is too low. If I decrease it (for example 0.001) the model learns very slowly and does not yield a good result. Therefore an intermediate value (0.01) was usually ideal for me.
- Additionally, I added a “decay” key to the HYPERPARAMETERS dictionary for decay ratio value, since exponential annealing method had problems. Generally exponential annealing gave better results than linear annealing for the epsilon value.

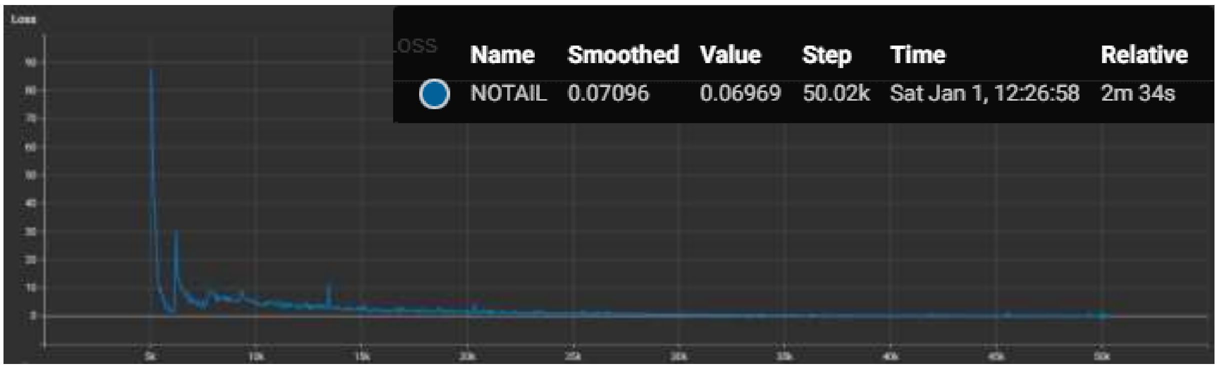
Results

Here are my resulting plots (created by using Tensorboard) of Loss and Reward values for each game mode. We can see that loss is decreasing over time and mean reward is increasing (reward values differ because different coefficients are useed but it is always increasing over time)

Note: plot images are seperately included in the zip file because they are compressed here and it might be difficult to read.

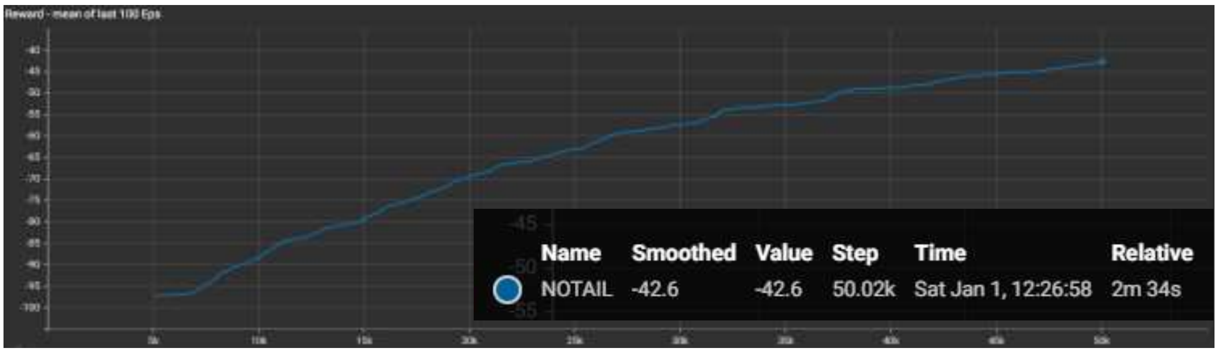
NOTAIL:

Loss:



Name	Smoothed	Value	Step	Time	Relative
● NOTAIL	0.07096	0.06969	50.02k	Sat Jan 1, 12:26:58	2m 34s

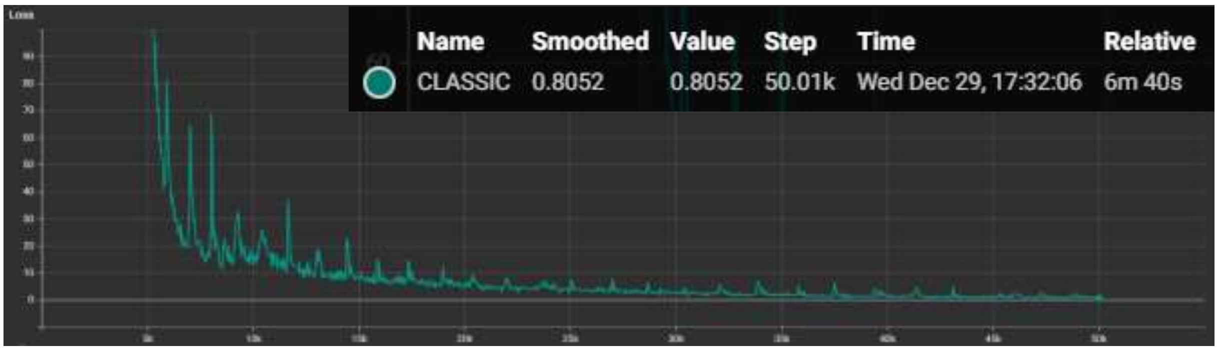
Reward:



Name	Smoothed	Value	Step	Time	Relative
● NOTAIL	-42.6	-42.6	50.02k	Sat Jan 1, 12:26:58	2m 34s

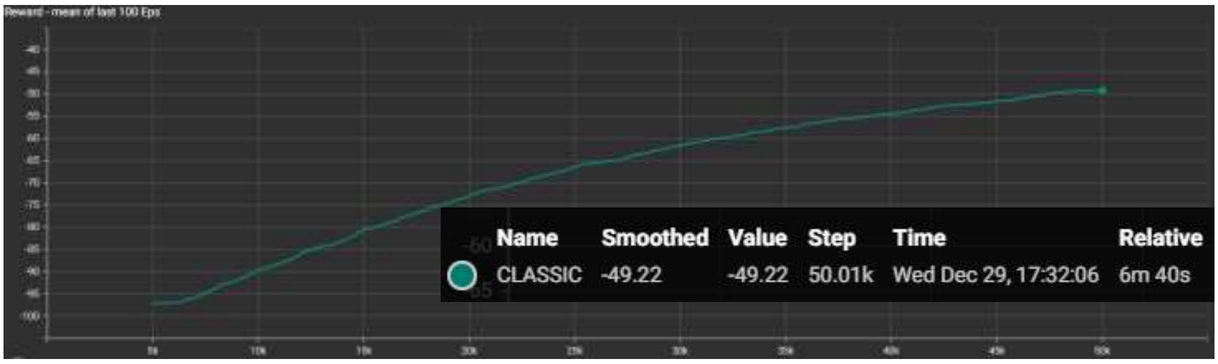
CLASSIC:

Loss:



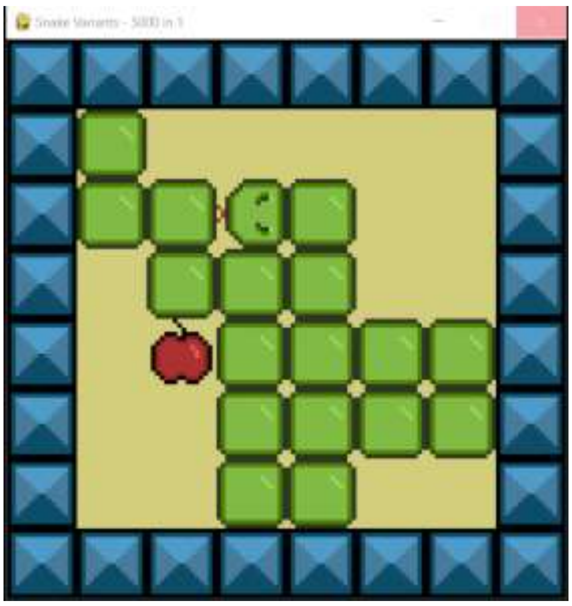
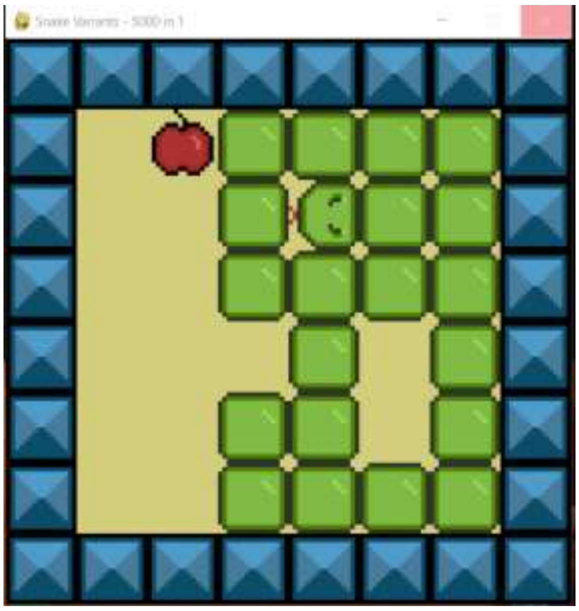
Name	Smoothed	Value	Step	Time	Relative
CLASSIC	0.8052	0.8052	50.01k	Wed Dec 29, 17:32:06	6m 40s

Reward:



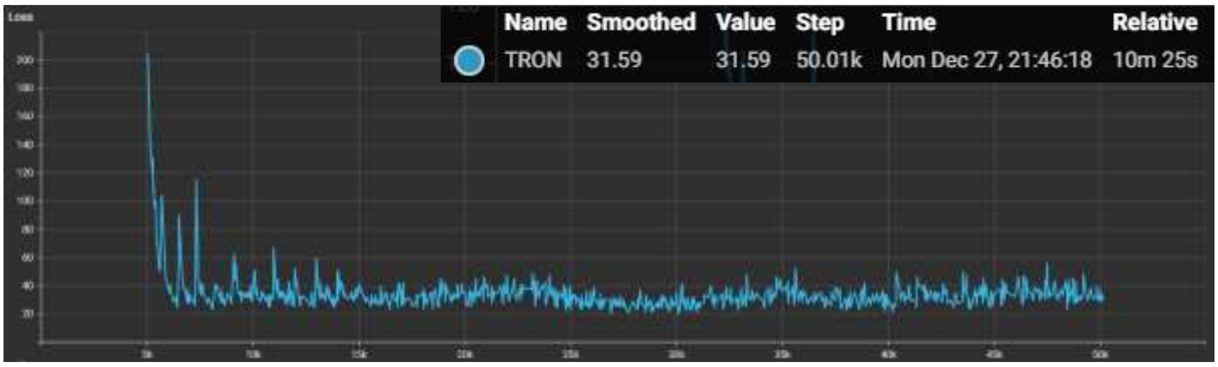
Name	Smoothed	Value	Step	Time	Relative
CLASSIC	-49.22	-49.22	50.01k	Wed Dec 29, 17:32:06	6m 40s

Images from trained model's games:

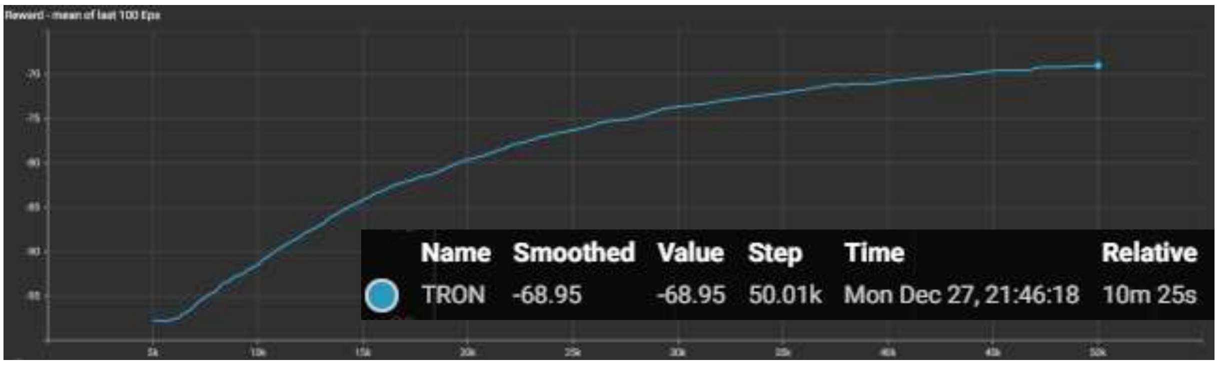
TRON:

Loss:



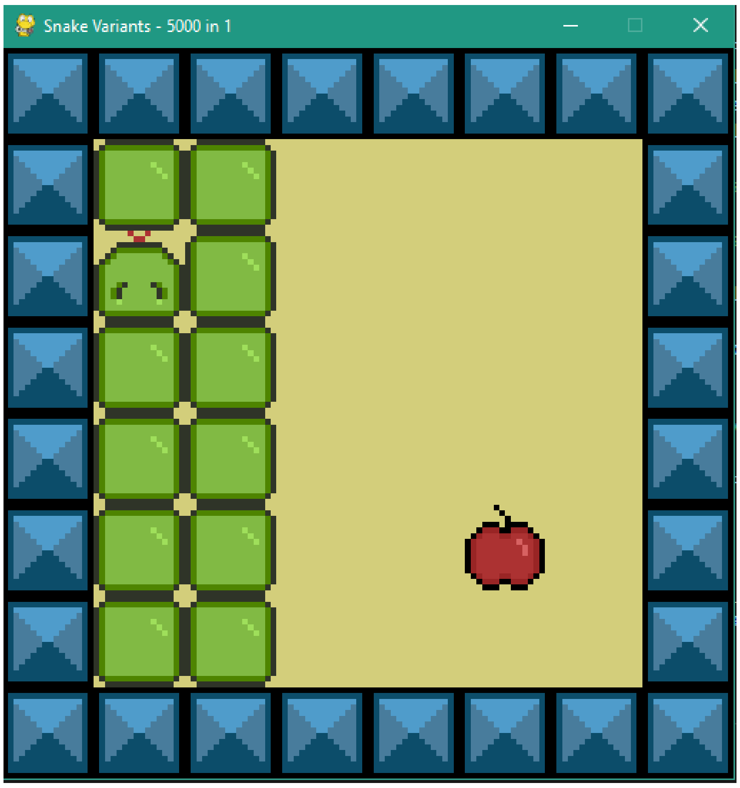
	Name	Smoothed	Value	Step	Time	Relative
●	TRON	31.59	31.59	50.01k	Mon Dec 27, 21:46:18	10m 25s

Reward:



	Name	Smoothed	Value	Step	Time	Relative
●	TRON	-68.95	-68.95	50.01k	Mon Dec 27, 21:46:18	10m 25s

Images from trained model's games:



How to Run the Code

I modified some extra parts of the .py files in addition to the ones mentioned in the homework PDF, in order to obtain some information (it was allowed in the description: “if you need to, you can add helper functions to get the data you want”). I included all necessary files in the submission .zip file.

After getting all files in the working directory and making sure necessary modules (such as pytorch) are installed, running `main.py` is sufficient.