Course: Autonomous Agents
Flappy Bird Agent using Q-Learning
Georgios Klioumis 2017030116

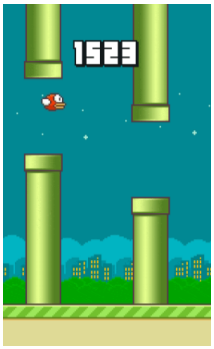
February 25, 2022

## Introduction

This project implements an autonomous agent that plays the well known Flappy Bird game using Q-Learning. The game core is forked from the FlapPyBird game on github.

## The Game

Flappy Bird is a game where the user controls a bird that is stationary in x-axis but is constantly free falling on y-axis. The player can press the space bar to flap the birds wings and lift it up, stopping the free fall. While the user tries to stop the bird from falling to the ground, a series of upward and downward pipes is constantly approaching the bird. The goal of the game is to avoid the pipes (pass between them) and the ground for as long as you can. Each time the bird flies through a set of pipes a point is acquired. As mentioned above the actual game used in this project is forked from the github repository, which provides the implementation of the game in python. During each frame the agent needs to decide on whether to flap the bird's wings or not.



## The Agent

In order to use Q-Learning to train the agent, we need to set the states and actions that are possible in this game. The Q-table used to store the above aspects is saved in csv format. This way the states and the corresponding weights for each action, are in readable form. The trained agent with the respectful Q-table is provided with the code. The alpha parameter can be either fixed for the whole learning process or it can decay as the bird reaches higher and higher scores until it converges to a low value.

The states in this game can be defined in a lot of ways. This implementation uses the below info (in form of number of pixels) to define each unique state.

1. The x-distance between the bird and the upcoming low pipe
2. The y-distance between the bird and the upcoming low pipe
3. The y-velocity that the bird currently has
4. The y-distance between the bird and the second upcoming low pipe

The actions that need to be weighted are the flap and no flap action. The rewards of the agent are 10 when the agent wins a point using the decided action and -1000 when the player dies. The -1000 "reward" penalizes the last 2 actions the player. The +10 reward approach is chosen in favor of 0 approach in order to stop the bird from experimenting with new actions when deciding on an already experienced state. The above might cause some over-fitting issues that can be examined in future project work. Moreover, a third parameter holding the total times a state is experienced is added to the Q Table.

## Q Learning

The Q-value update function is

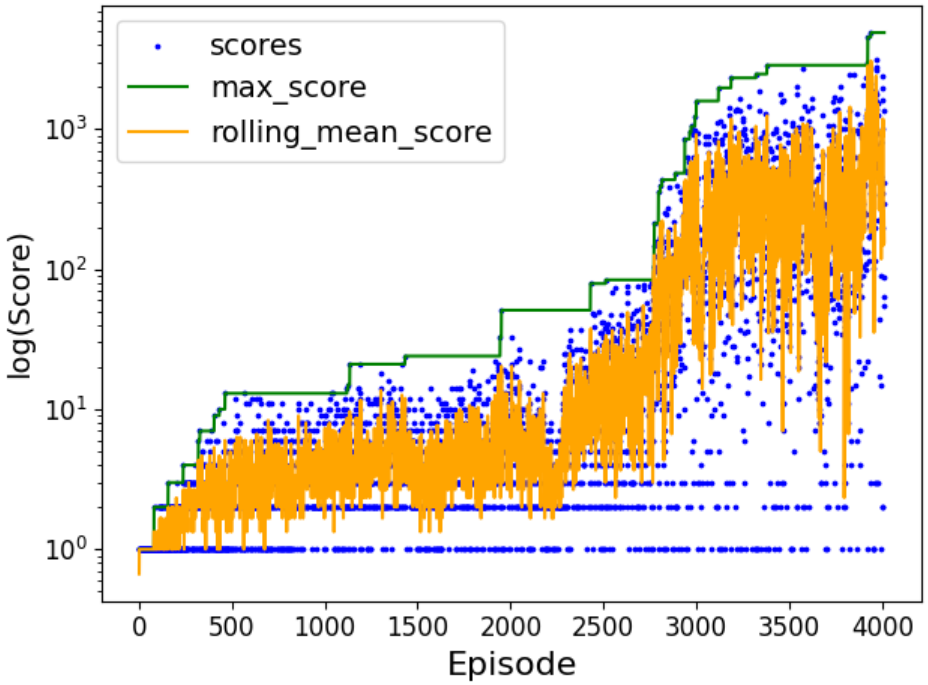
$$Q^*(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \; max_a Q(s',a) - Q(s,a)]$$

where s is current state, a is action, s' is next state, R is the reward function, $\alpha$ is the learning rate and $\gamma$ is the discount factor. The Q values of the Q table are updated on the death of the bird from the last action to the first, penalizing the last 2 actions and rewarding the actions that made the bird live. In order to make the learning algorithm run quicker, the game is run at 10000 frames per second.

# Results

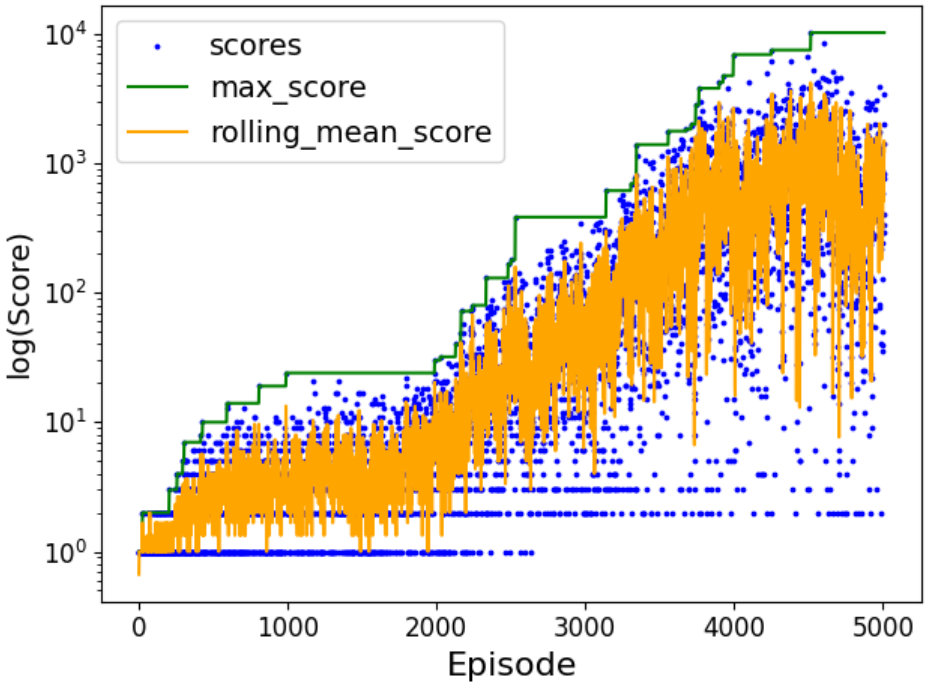
## Without Alpha Decay

The agent was first trained with $\alpha = 0.9$ and $\gamma = 0.95$. The agent's learning progression was logged at the end of the learning process which was bound initially to 4000 episodes (re-spawns). Below is the mentioned diagram:

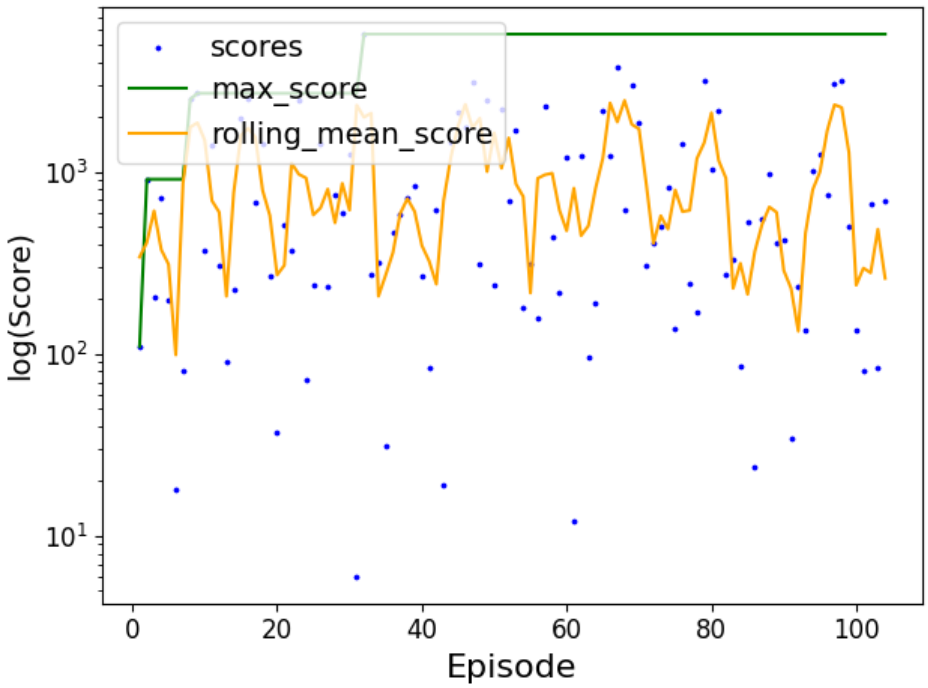



It can be seen that the agent learns quite quickly to survive until the 1500th episode and then the max score is rarely increased until we heat 3000 episodes. In the 3000th episode starts a great learning curve that can be seen from the rolling mean score of the agent. Before the 3000th episode the agent has a rolling mean of roughly 50, whereas afterwards the rolling mean spikes to 500 with a max score of 1500. The scores acquired by the agent are pretty stable after this spike, but show an increasing action. It is clear that with more training, the agent can reach highest scores. The training time for this setting was about an 1 hour and 30 minutes.

A second run of a fresh agent with the same setting was run until 5000 episodes in order to test the above hypothesis. The results were:




In this run we can clearly see the linear (in logarithmic scale) increase of the rolling mean score that the agent acquired. Furthermore, the above hypothesis is correct as the agent is able to reach a max score of around 10000 with a training time of roughly 2 hours and 30 minutes. The rolling mean score at he end of the training session is 1000. In order to test the stability of the trained agent, he played the game for 100 episodes without any learning algorithm. The results were pretty stable and listed below:



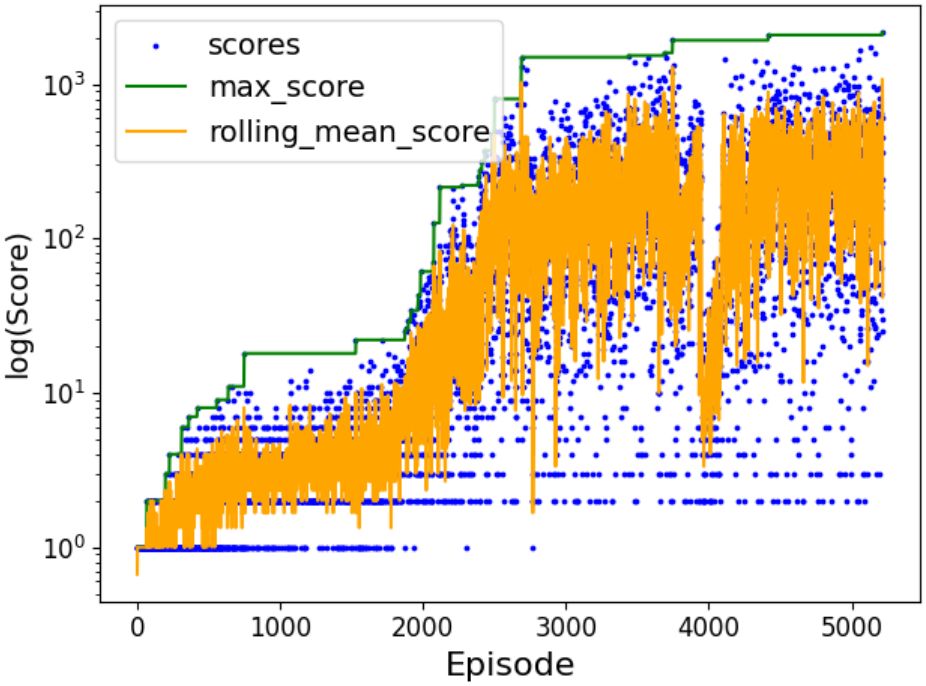

## With Alpha Decay

### Decay $\alpha$ to 0.75

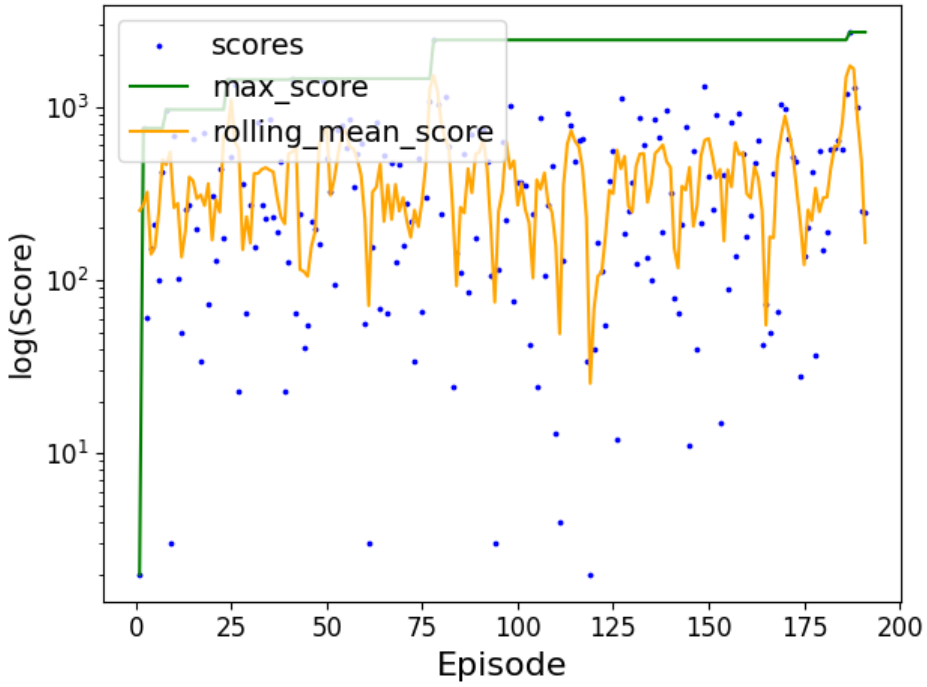
The second approach is to decay the alpha value to a smaller number as the episodes progress in order to reach $\alpha = 0.75$ at 5000 episodes. The decay function is listed below:

$$\alpha_{new} = max(0.75, \alpha - \alpha_{decay})$$

The agent performed a bit worse that the agent that did not decay alpha value. The agent's performance is listed below:



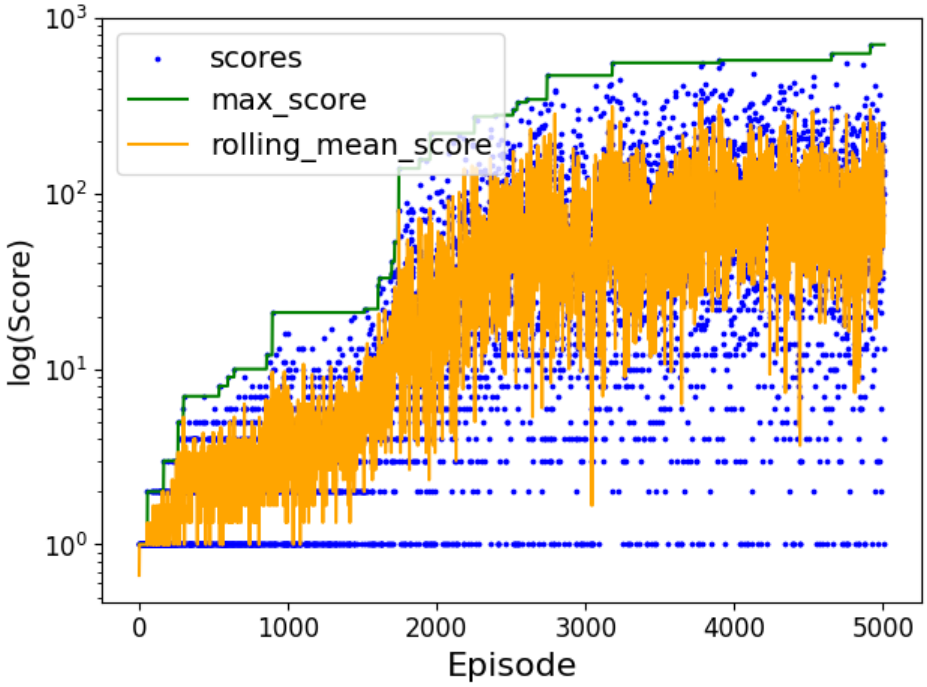

It can be seen that the learning curve is steeper in the beginning of the learning phase and becomes more and more stable as we further decrease the alpha value. The stable learning region begins at episode 3000 where $\alpha$ has decayed to aprrox. $\alpha = 0.8$. It is also crucial to mention that the learning algorithm continues to learn after the 3000th episode, but with a smaller rate. The above can be seen from the increasing value of max score. In order to test the stability of the learned agent a test was performed, that loaded the Q-table and the agent played the game for 100 episodes as before. The results can be seen below:



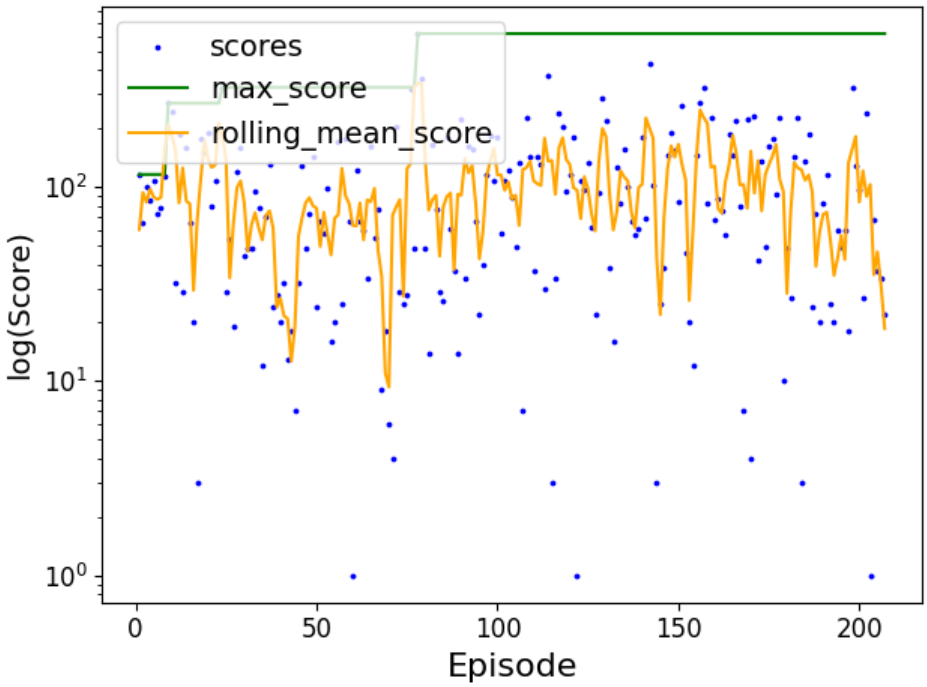

As we see the score is approximately 110 which is the same as the rolling mean score from the previous diagram in the 5000th episode.

### Decay $\alpha$ to 0.1

In order to further understand and test the alpha decay mechanism, a further run was performed that decays the alpha value to $\alpha = 0.1$ during a 5000 episode period. The results can be seen below:




We can see that the further decay of alpha value resulted to worse overall max score (900) and rolling mean score (100) but stabilized the above values at around 2000 episodes. It can be further seen that the there is little progress to the max score after the 3000th episode. In order to test the stability of the Q-table, the same test as the the previous mechanisms was performed and the results were pretty stable as well.

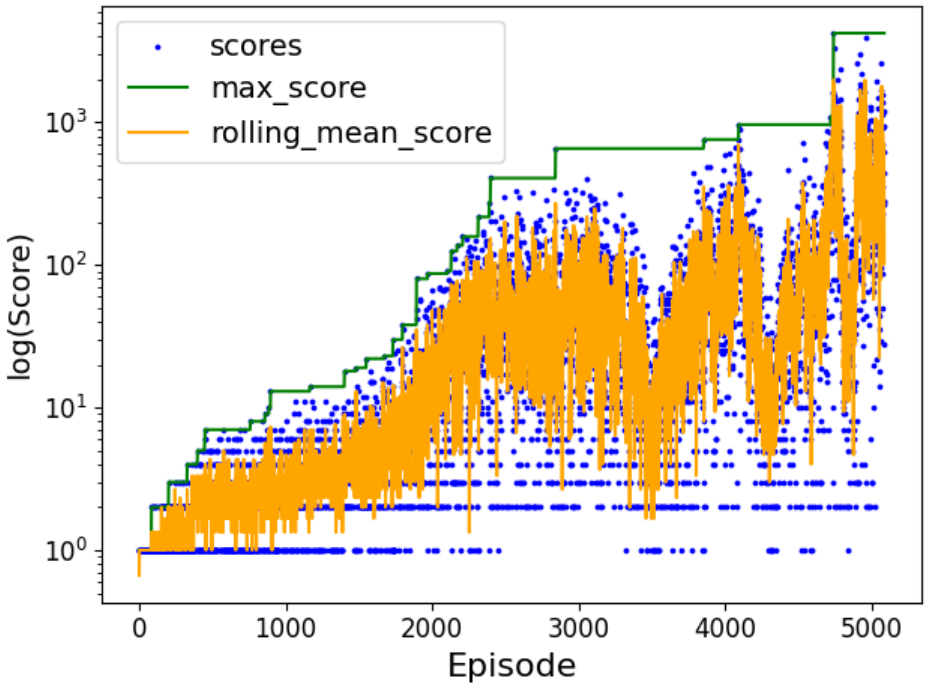



Due to smaller overall scores per episode the training time of this agent was far less compared to the previous agents. The above can be beneficial if we want an agent that can reach a certain score and stabilize the learning rate at that point quickly.

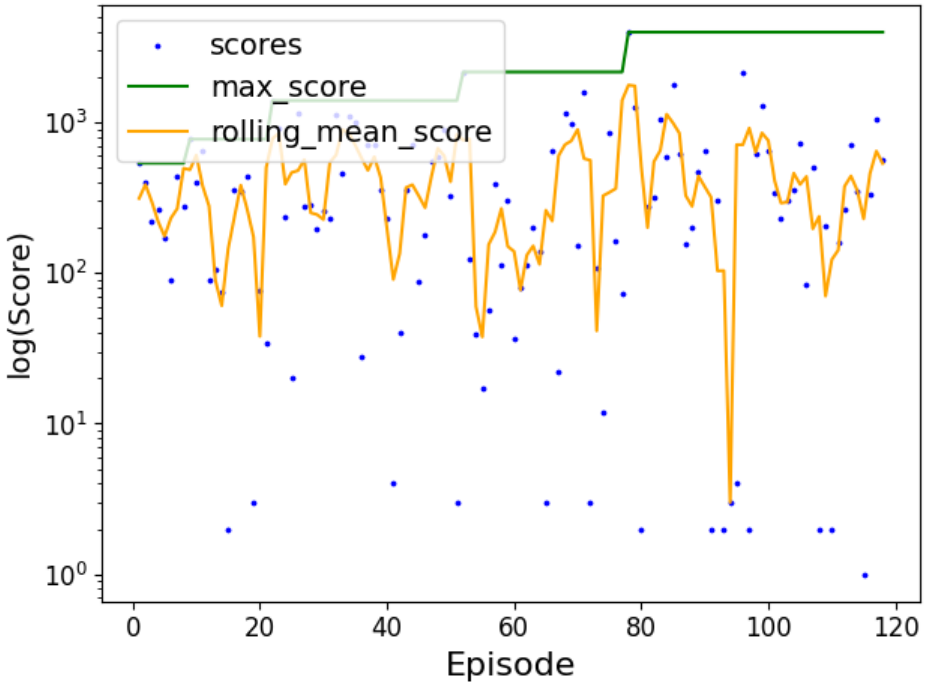
It can clearly be seen though that the alpha decay mechanism performed worse in 5000 episodes, which is there to show that the agent can learn a lot more and is not ready to converge the Q-table values in 5000 episodes. The alpha decay mechanism though will be handy if the agent is further trained and stability becomes a key needed part in late training region.

## ε-Greedy

A further idea to make the agent train faster is use the ε-Greedy policy. With this mechanism a chance ε makes the agent enter an exploration state, in which the agent decides randomly among the 2 action of the game, flap or no flap. Initially, this ε chance is set to 0.1 and is decaying until it reaches 0 in 3000 episodes. The hypothesis is that this mechanism will make the agent reach highest scores quicker through exploration, but will also be stable near the end (5000th episode) and not die due to random exploration (ε-decay concept). The results of the agent are shown below:




It can be seen that this mechanism enables the agent to learn quicker than any agent tested so far. Furthermore, it is important to note that this agent has comparable performance with the fixed alpha agent as they both reach max scores near 10000. The learning process stabilizes with an increasing trend at around 2500 episodes, where $\varepsilon$ value has decresed considerably. In order to test for stability the known test was executed:




We can see that the agent performed around 500 score, which shows the comparable performance with fixed alpha agent.

### Overall

Overall we can see that the agent with no alpha decay and an alpha value of 0.9 and $\gamma$ value of 0.95, performed better than any other setting tested. The above is direct result of the fact that game has no end. An optimal agent can continue playing to infinity, which makes convergence of the Q-table values seem useless. It is clear though that when a certain predefined score goal is set the alpha decay mechanism is benefitial.

## Future Work

A lot of ideas were brainstormed during the development of this agent, but the most remarkable is the one listed below. In order to increase the slow literal speed of learning a lot of birds can be spawned simultaneously on the same game and compete with each other for the biggest score. Every bird will have their own Q-table. The states that have not been experienced can begin with weights randomly distributed among the bird's of the same generation. Upon the death of the last bird, the Q-table of the n (probably n=4,5) fittest birds, can be weighted with the ranking of each bird (the last bird to die is the fittest)

and define the Q-table of the next generation. Every bird can be run by its own thread to reduce CPU latency (and increase the framerate of the game further).