、

、

、

、

、

   Jihui Wu(Joe 19034794)

## Reinforcement learning in ‘ flappy bird’

# table of contents

[table of contents 1](#_Toc900449449)

[1.ABSTRACT 2](#_Toc536628934)

[2.KEYWORDS 2](#_Toc1838659985)

[3.INTRODUCTION 3](#_Toc68687565)

[3.1 Problem definition 3](#_Toc1233186516)

[3.2 Related work 3](#_Toc801097215)

[3.3 Reinforcement learning(RL) 4](#_Toc1465909462)

[3.4 Q-LEARNING(QL) 5](#_Toc1607929450)

[4. IMPLEMENTAION 6](#_Toc536052302)

[4.1 ENVIROMENT 6](#_Toc737140549)

[4.2 APPLY ALGORITHM 6](#_Toc288047500)

[4.3 ACTION SELECTION POLICY 8](#_Toc786192162)

[4.4 Q-VALUE UPDATES 8](#_Toc64786743)

[5. DISSCUSSION 11](#_Toc96580572)

[4.1 STRENTHS & LIMITATIONS 11](#_Toc1879520119)

[4.2 parameters 12](#_Toc1757676310)

[6. Core cODE 12](#_Toc480694038)

[6.1 reward calculation function 12](#_Toc191216652)

[7. Conclusion 14](#_Toc1142734252)

[Reference 15](#_Toc988318243)

1.ABSTRACT

Machine learning has already been applied to numerous fields. Specifically, researchers in AI has invested more in training AI to to play popular games. In order to solve this kind of problems, researchers have to understand more specific features , environment. Then they can apply their knowledge into this field. The game in this report is Flappy Bird, which involves controlling a bird navigate through pipes (obstacles) to score. My goal is to implement reinforcement learning, in this case,specifically, Q learning that I have learned from lecture to train the bird to take safe action in each state. Firstly, I will explain the definition and concepts of reinforcement learning and Q-learning, and then describe how to use this algorithm in this specific case by using pseudo-code and an example. Finally, I will discuss the results and recommendation for improving the performance.

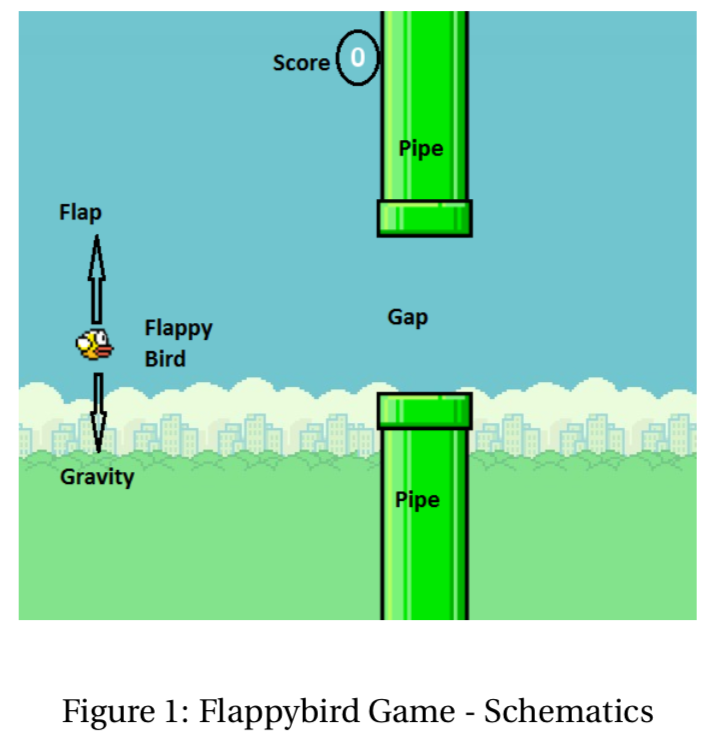
# 2.KEYWORDS

Machine learning, Q learning, Flappy bird, Reinforcement learning, Mobile games

# 3.INTRODUCTION

## 3.1 Problem definition

Flappy bird ,which is shown in FIgure1,is a mobile game where the user controls the bird, which is the "agent" of the game to pass through the gaps between two pipes. At each second, there are two actions that the player can choose to take: to ‘click’ , which makes the bird jump upward or do nothing, which makes it drop at a constant rate(like gravity in real life).



The goal here is use algorithms I have learned this semester to develop a machine learning model to learn specific game features just from the environment (raw pixels world) and train the agent to take correct actions in each frame. Inspired by [1] and [2], I decided to use the reinforcement algorithm set-up to learn and play this game.

Considering the encapsulation of smartphone, as a front-end developer, I decided to use web browser to implement and test this game.

## 3.2 Related work

There are lots of work has been done in implementing AI algorithms in video games. For instance, Google Deepmind team has put efforts to receive extremely high performance in some games, such as the AlphaGo[4], which has defeated the experts in the game and make the public believe the power of AI. Besides, in the recent StarCraft II competition, the AI AlphaStar from the DeepMind has achieved 10-1 against those professionals in this game[1,2];

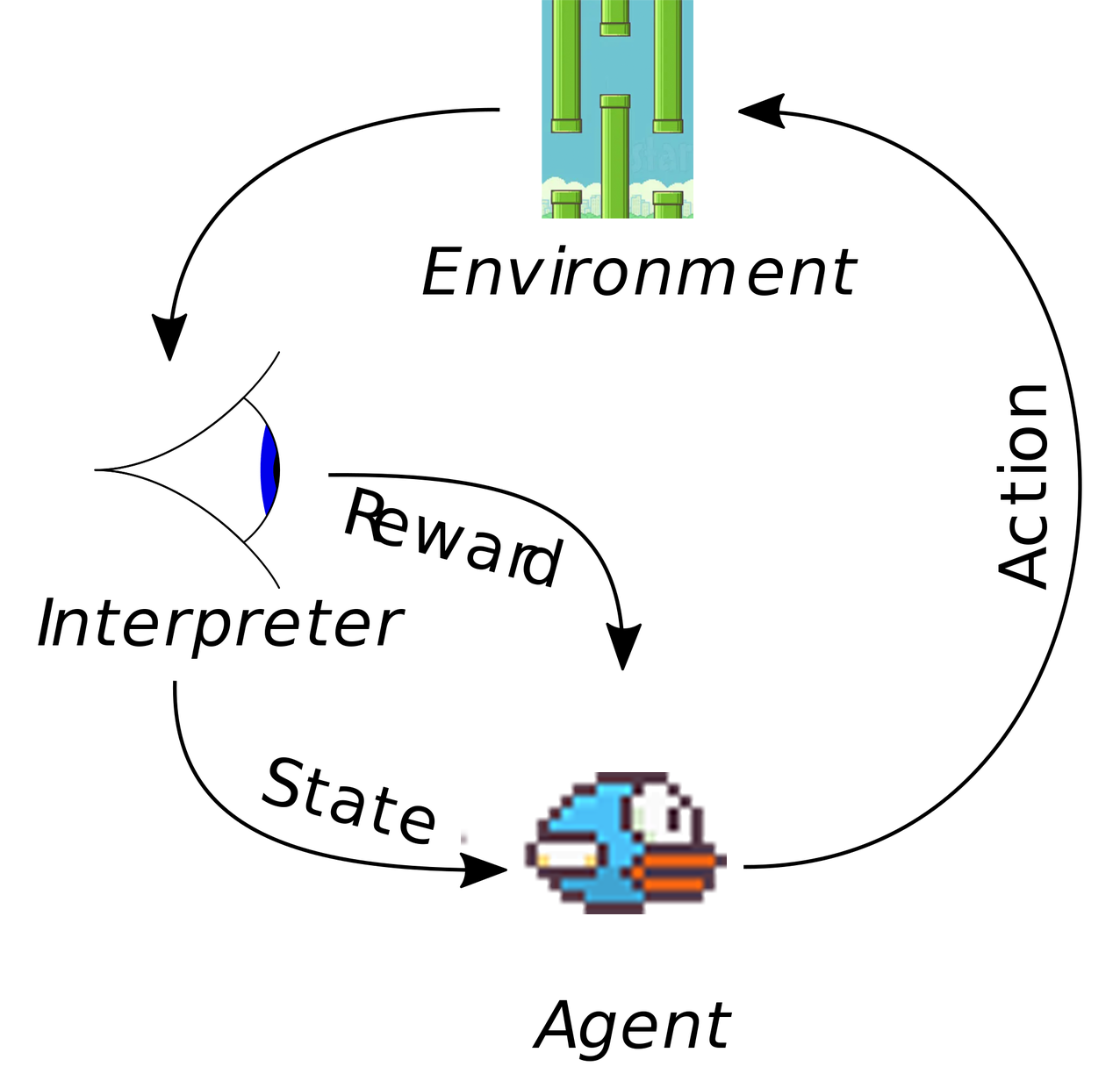
And in this popular mobile game, many python programmers have used a lot of AI algorithms, such as Neuro-evolution,Deep-Q-learning algorithm[6] and so on to reach score as high as possible in a short period of time. I just chose the Q learning method and used a relatively simple way to define the state.

## 3.3 Reinforcement learning(RL)

**3.3.1 definition**

It is just one of machine learning methods, parallel with supervised and supervised learning. What makes it different from the other two is that rather than being fed with data, it explores to get data.

According to my understanding, Reinforcement learning is like training to get conditioned reflex. We navigate the agent in the environment again and again to take actions and records every rewards or punishments it takes. And then after enough times of trials and errors, the agent will probably choose the best action according to the records(memory).So based on the learning experience, the strategy is very simple, in certain state, just choose the action that is expected to receive the highest reward.



In a typically RL scenario, an agent explores in the certain environments, when it takes some action, it will receive some rewards or punishments, and then it can navigate in the environment to get higher rewards by exploiting the previous experience.

## 3.4 Q-LEARNING(QL)

As the most popular algorithm in reinforcement learning, this algorithm learns the approximate function of the optimal action-utility function:

Q[S,A] ← (1-α)\*Q[S,A] + α\*(R(S,a) + γ\* max Q[S',a])

In this equation,

* Q: like the mathematical expectation, that means the reward we will probably get from taking a certain action in a certain state.
* R: value/score of the rewards or punishment,
* S: a certain state,
* A: the action taken.
* max Q[S',a]: the highest Q value can be obtained in the next state according to previous experience
* α : Learning rate, it defines how important new learned information is to the agent. The higher this value, the fast the learning process, but if we set this value too high, like 1, we can see in this equation, this is (1-α)\*Q[S,A], that means will lose the records from previous learning, that will lead to instability in learning.
* γ - Discount factor, defines whether new information is more important than the previously learned information. Range [0, 1], If immediate rewards are more important than long-term rewards, set discount factor = 0. If future (delayed) rewards are more important, set discount factor close to 1.

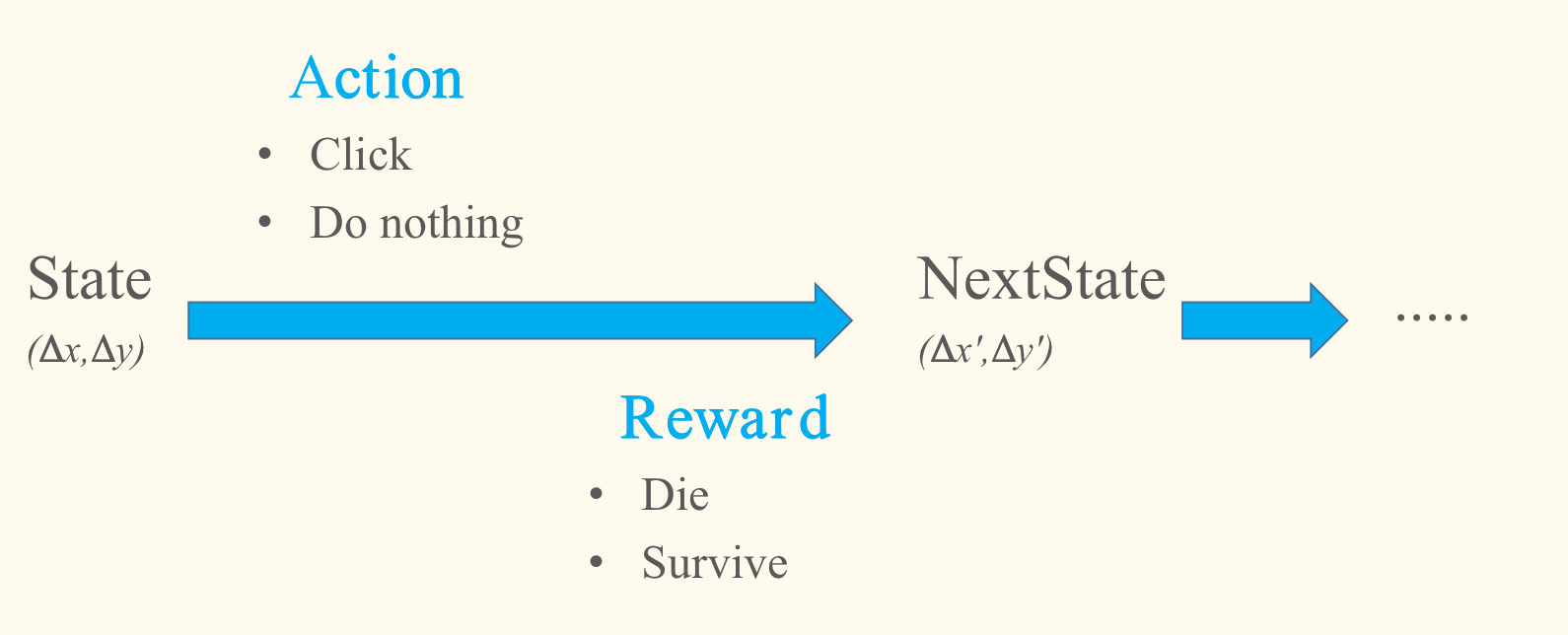
# IMPLEMENTAION

## 4.1 ENVIROMENT

Considering the encapsulation of the application on mobile phone, I decide to use the game that has already been built on web client (written in HTML and Javascript), and add the code of reinforcement algorithm to make this learning work.

## 4.2 APPLY ALGORITHM

Firstly, the state, action and rewards in this case should be considered firstly.



**4.2.1 Action**

It is fairly clear that there are only two actions available in each state: click(jump) or do nothing(stay).

**4.2.2 Rewards/Punishments**

We can consider that keeping alive for one frame scores 1 point.

And if died, 100 points will be deducted.

|  |  |
| --- | --- |
| **State** | **Score/reward** |
| die | -100 |
| survive | +1/frame |

**4.2.3 States**

As showed in the figure 2

I defined the state by two distances to the next pair of pipes:

Δx: Vertical distance from lower pipe

Δy: Horizontal distance from next pair of pipes

So that we can use (Δy,Δx) to represent each state.

Figure 2 states

|  |  |  |
| --- | --- | --- |
| **State(m\*n)** | **Click(jump)** | **Do nothing(stay)** |
| **(Δx-1,Δy-1)** | 1 | 20 |
| **(Δx-1,Δy-2)** | 20 | -100 |
| ... | ... | ... |
| **(Δx-m,Δ-n-1)** | -100 | 2 |
| **(Δx-m,Δ-n)** | 50 | -100 |

## 4.3 ACTION SELECTION POLICY

There are two kinds of action selection policy in this algorithm.

**4.3.1 greedy**

This policy is directly base on Q-value, it always select the action with the highest Q value .

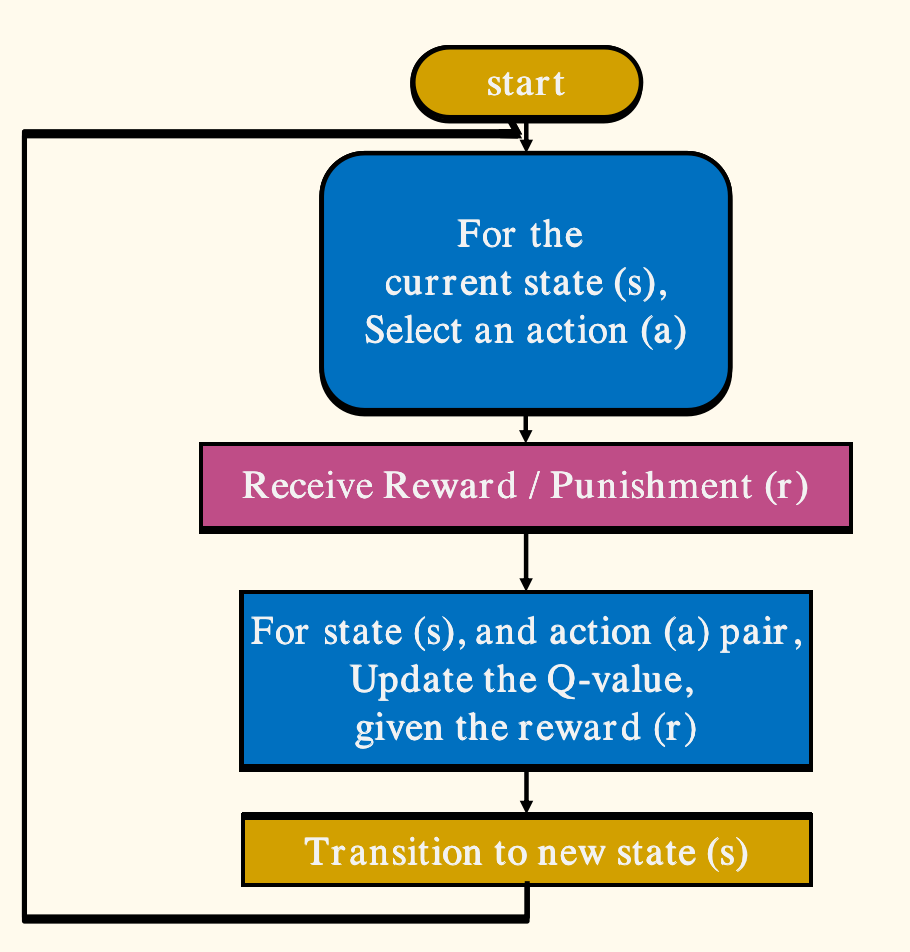
But considered that if the first action has been tried in a certain state and get a positive score, then this policy will continue to choose this action and will never that other actions that may lead to higher score. Thus, using this policy may lead to the localized optimal result.

**4.3.2 P-greedy(Probabilistic greedy)**

This difference of this policy is that it will try to explore the environment and experiment all the possible actions by using probability, which is calculated base on the Q value. The higher the Q value, the more likely this action will be chosen in this state.

## 4.4 Q-VALUE UPDATES

**4.4.1 Process diagram**



**4.4.2 Pseudo code:**

**Initialize Q table as {} or [m\*n][2];**

**Repeat(Until Q is sufficiently good):**

**Initialize position S, start a game:**

**while (S != died)：**

**use strategy π，receive action a=π(S)**

**adopt action a，receive new position S' and reward R(S,a)**

**Q[S,A] ← Q[S,A] + α\*(R(S,a) + γ\* max Q[S',a]- Q[S,A]) //update Q**

**S ← S' //update S**

**4.4.3 An example**

|  |  |  |
| --- | --- | --- |
| **State(Δx,Δy)** | **Click(jump)** | **Do nothing(stay)** |
| **(30,50)** | 0 | 20 |
| **...** | ... | ... |
| **(30,55)** | 50 | -100 |
| ... | ... | ... |

**In this case, we set α=0.6, because we values information from new learning, and set γ=0.8, because in this case ,the punishment is often delayed.**

1. **The bird is in state (30,50) ;**
2. **According to the action selection policy π, it choose to ‘click’ which has not been done before in this state;**
3. **Get the reward from the environment is 10;**
4. **The agent will enter into the next state(30,55):**

**Qmax(30,55) = 50;**

1. **According to the Q function, do the calculation**

**Q[S,A] ← Q[S,A] + α\*(R(S,a) + γ\* max Q[S',a]- Q[S,A])**

**Q((30,50),’jump’)← 0+0.6(10+ 0.8\*50-0)= 30**

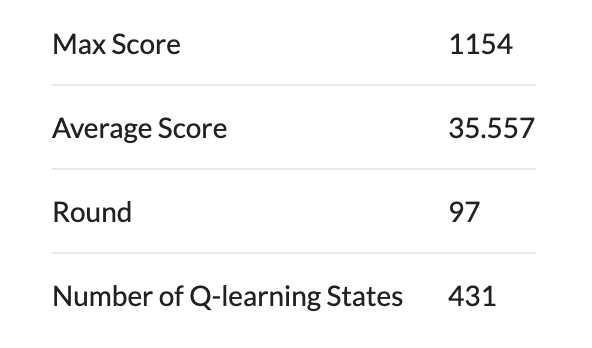
**(5)update the Q((30,50),’jump’) value to be 30 in the Q table.**

# DISSCUSSION

## 4.1 STRENTHS & LIMITATIONS

4.1.1 Strengths

* In this specific scenario, it is quietly straight forward to use reinforcement to train the bird to achieve higher score instead of supervised or unsupervised learning, because we don have to use data to feed. Instead, we let the bird to try as many times as possible to get the data itself to improve the score.
* And by using the P-greedy strategy, we allow the agent to explore the environment and try all kinds of actions available. That means we can get the overall rather than the localized optimal.
* By using Q-learning, we can get a large range of data, and if we train it long enough, we may even get all the value for the Q-table.
* Q-learning algorithm is relatively simple and easy to implement, that means we can use the shortest time to write the code to train the bird.
* The result is fairly good, after less than 100 epochs, it can score more than 1000 points, that is almost impossible for normal human players.



4.1.2 Limitations

* In this specific case, because of the size of the map, the number of states in this table is fairly large(320\*220), that means it will take a long time to converge and may not be able to.
* Because of the rewards are not given immediately, so the rewards we calculated may not be accurate enough, which sometimes causes the bird to crash again and again in the beginning.
* Because of the large number of states, the Q-table is fairly big, it needs a large array to store all the value and sometimes it needs relatively long time to access value in the table, that may cause the learning to stuck.

## 4.2 parameters

* Game size: 512\*288px
* maxΔ: 320\* (172+48) px;
* α : (learning rate): 0.6 (values new information)
* γ : (discount factor): 0.8 (delayed reward)

# Core cODE

## 6.1 reward calculation function

*function reward(Q, S, S\_, A, R) {*

*if (S && S\_ && A in [0, 1] && S in Q && S\_ in Q)*

*Q[S][A] = (1 - qlAlpha) \* Q[S][A] + qlAlpha \* (R + qlGamma \* max(Q[S\_]));*

*return Q;*

*}*

6.2 update QL function

*function updateQL(gameState) {*

*if (!updateQL.enabled) return gameState;*

*if (updateQL.skip) {*

*updateQL.A = null;*

*updateQL.S = null;*

*}*

*if (!updateQL.Q) {*

*updateQL.Q = {};*

*updateQL.S = null;*

*}*

*var Q = updateQL.Q;*

*// prev state*

*var S = updateQL.S;*

*// prev action*

*var A = updateQL.A;*

*// current state*

*var S\_ = getQLState(gameState);*

*if (S\_ && !(S\_ in Q)) Q[S\_] = [0, 0];*

*if (gameState.mode == "playing") {*

*updateQL.Q = reward(Q, S, S\_, A, qlAliveReward);*

*updateQL.S = S\_;*

*// current action, 0 for stay, 1 for jump*

*var A\_ = 0;*

*if (Math.random() < qlEpsilon) { // explore*

*A\_ = Math.random() < qlExploreJumpRate ? 1 : 0;*

*} else if (S\_ in Q) { // exploit*

*A\_ = Q[S\_][0] >= Q[S\_][1] ? 0 : 1;*

*}*

*if (A\_ === 1) gameState = jump(gameState);*

*updateQL.A = A\_;*

*} else if (gameState.mode == "dead") {*

*updateQL.Q = reward(Q, S, S\_, A, qlDeadReward);*

*updateQL.S = null;*

*updateQL.A = null;*

*// restart the game*

*updateQL.skip = false;*

*gameState = jump(gameState);*

*}*

*return gameState;*

*}*

# Conclusion

By learning the concepts of of reinforcement learning through lectures this semester, I was able to implement it into this popular game to make the AI to reach extremely high scores in just a few minutes, which may take a human hundreds of hours to achieve. Though the results did not show scores that are not reachable for human and the performance is still room for improvements, it was definitely a meaningful step in the right direction for learning AI. Overall, the result shows the capacity of reinforcement learning and how to put learned machine learning knowledge into practice. It also proves that there are lot of applications fields that we can use machine learning algorithms to improve the performance in very short amount of time. There are still a lot to be learned and done in this field

## Reference

[0] Lecture slides “Lec2019 - 4 - 159740- Reinforcement Learning - Part 1 - QLearning”

[1] C.ClarkandA.Storkey.Teachingdeepconvolutionalneural networks to play go. arXiv preprint arXiv:1412.3409, 2014. 1, 2

[2] V.Mnih,K.Kavukcuoglu,D.Silver,A.Graves,I.Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep rein- forcement learning. arXiv preprint arXiv:1312.5602, 2013. 1, 2

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep re- inforcement learning. Nature, 518(7540):529–533, 2015. 3, 5

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Pan- neershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. 529(7587):484–489, 2016. 1

[5]Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, et al. 2019. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/. (2019)

[6]Naveen Appiah,Sagar Vare,Playing FlappyBird with Deep Reinforcement Learning,

http://cs231n.stanford.edu/reports/2016/pdfs/111\_Report.pdf