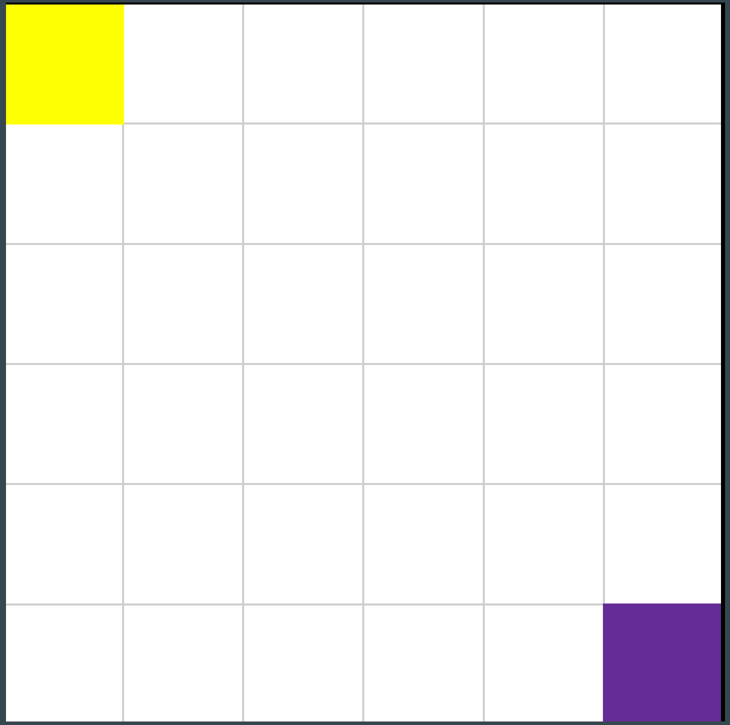
Let's Play a Game!



1
2
3
4

- Knowns:
  - 4 possible keys (1-4)
  - Reward-based
  - Starting position
- Unknowns:
  - Goal of the game
  - What any of the keys does
  - Consequences of objects in the environment

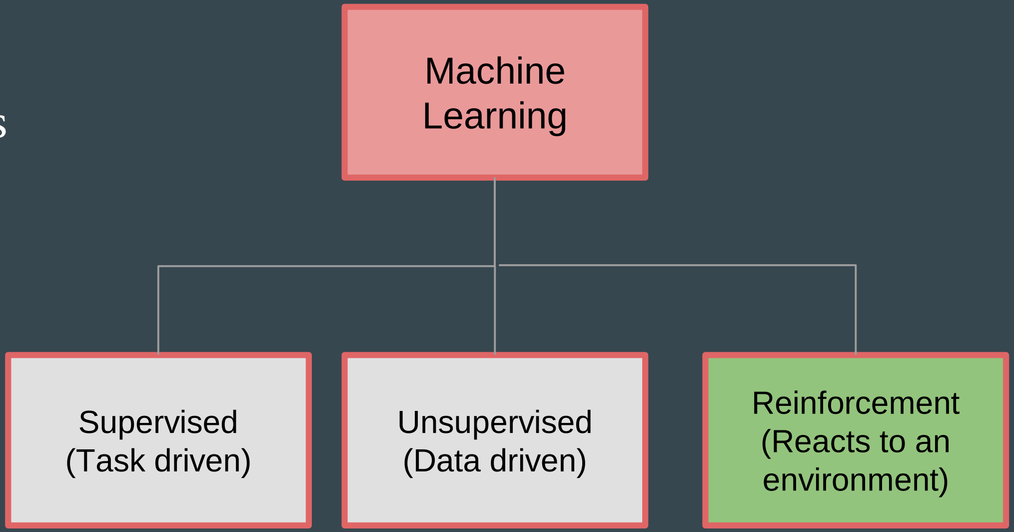
https://classroom.udacity.com/courses/ud600/lessons/4100878601/concepts/6382590580923

Deep Q-Learning

Brittany Haffner, Atharva Jakkanwar, Apar Singhal, Bryan Fuh

Reinforcement Learning

- Neither supervised or unsupervised
- Does not require training data
- Interacts with the environment
- Updates policy based on rewards

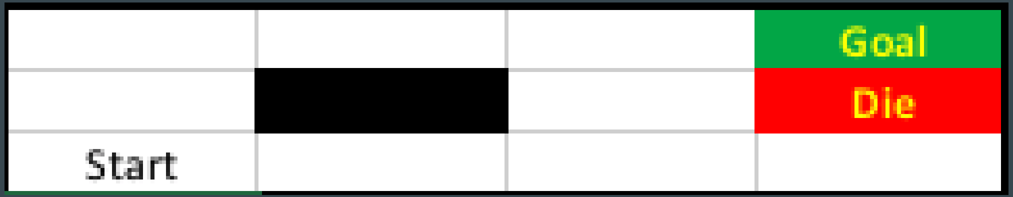


Markov Decision Process

- Framework used to make decision on stochastic environment
- More useful than simple planning
  - MDP: solution will give you optimal action even if something goes wrong
  - Simple planning: Follow the plan after you find best strategy (doesn't allow mistakes/errors)
- Markovian property: the effects of an action taken in a state depend only on that state and not on prior history

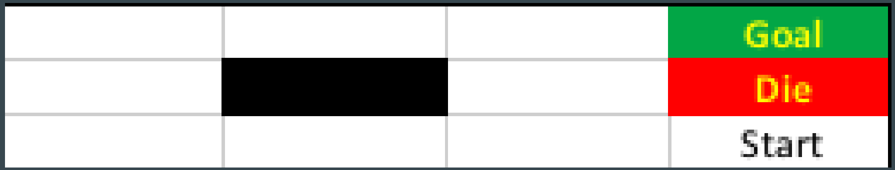
Markov Decision Process

- Discrete time stochastic control process:
  - States: *s*
  - Actions: $A(s)$, a
  - Rewards: $R(s)$, $R(s,a)$
    - E.g. -1 for every move
    - E.g. +100/-100 for end states
  - Transition model (rules/physics of the game): $T(s, a, s') \sim Pr(s' \mid s, a)$
    - For this example, 0.8 chance of going in the desired direction
  - Policy (solution): $\pi(s) \rightarrow a$; $\pi^*(s) \rightarrow a$



Markov Decision Process

- Infinite horizon
- Finite horizon
  - Finite number of time steps
  - Changes policy
- Utility of sequences: $U(s_0, s_1, s_2, \ldots) = \sum_t \gamma^t R(s_t),\ 0 \le \gamma \le 1$
  - γ: discount factor
  - Allows program to go infinite distance in finite amount of time



Bellman Equations

- The key to solving MDP in RL
- $\pi^* = argmax_\pi \mathbb{E}[\sum_t \gamma^t R(s_t) | \pi] \rightarrow \pi^*(s) = argmax_a \sum_{s'} T(s,a,s')U(s')$
- Utility score
  - The expected maximum cumulative reward by following the policy given by state s
  - $U(s)=\alpha\{R(s,a)+\gamma max_a \sum_{s'} T(s, a, s')U(s')\}, 0 < \alpha < 1$ *(learning rate)*
  - $\pi^*(s) = argmax_a U(s,a)$
  - $U(s) = R(s) + \gamma\pi^*(s)$
- Fundamental recursive equation that defines the true value in a particular state
- Determines the best action to take

Bellman Equations

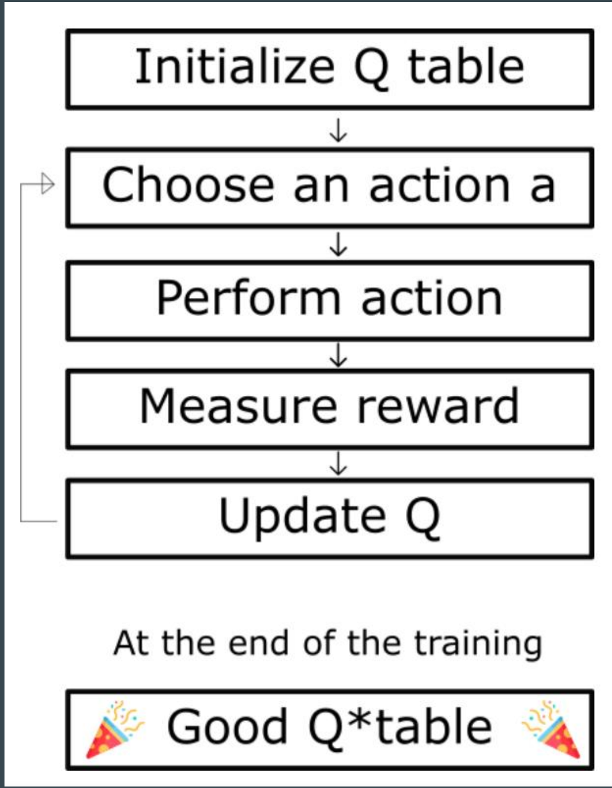
- Quality score (Q-Value)
  - The expected maximum cumulative reward from taking action *a* in state *s* and following the policy
  - $Q(s,a) = \alpha\{R(s,a) + \gamma max_{a'} \sum_{s'} T(s, a, s')Q(s',a')\}$
  - $Q(s',a') \leftarrow Q(s,a) + \alpha\{R(s,a) + \gamma max_{a'} Q(s', a') - Q(s,a)\}$
  - New action value = Old value + learning rate * (new information - old information)
- Finding policies:
  - Initialize: π_0 as initial guess
  - Evaluate: given π_t calculate $Q_t = Q_t^{\pi}$
  - Improve: $\pi_{t+1} = argmax_a \sum T(s, a, s')Q_t(s',a')$
  - Policy equation: $Q_t(s) = \alpha\{R(s) + \gamma \sum_{s'} T(s, \boldsymbol{\pi}_t(t), s')Q_t(s')\}$

Q-Learning

- Q-value-based reinforcement learning
- Take action based on maximum expected future reward for current state: $\pi(s) = argmax_a Q(s,a)$
- Generate a Q-table

Example Q-table

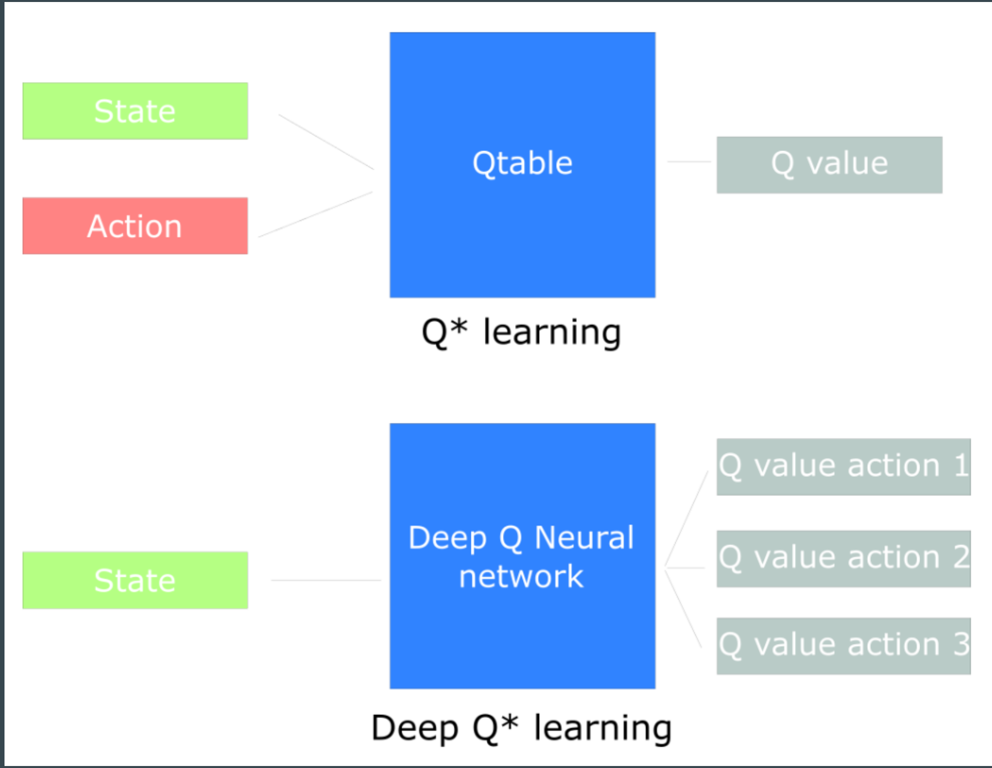
	a_1	a_2	… a_n
s_1	0	0.1	0
s_2	0	0	0
… s_m	0	0	0



https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe

Deep Q-Learning vs. Q-Learning

- Replace Q-table with CNN
- Better for large environments with many different states
- Used for learning control policies from high-dimensional inputs
- Choose the action with the highest Q-value



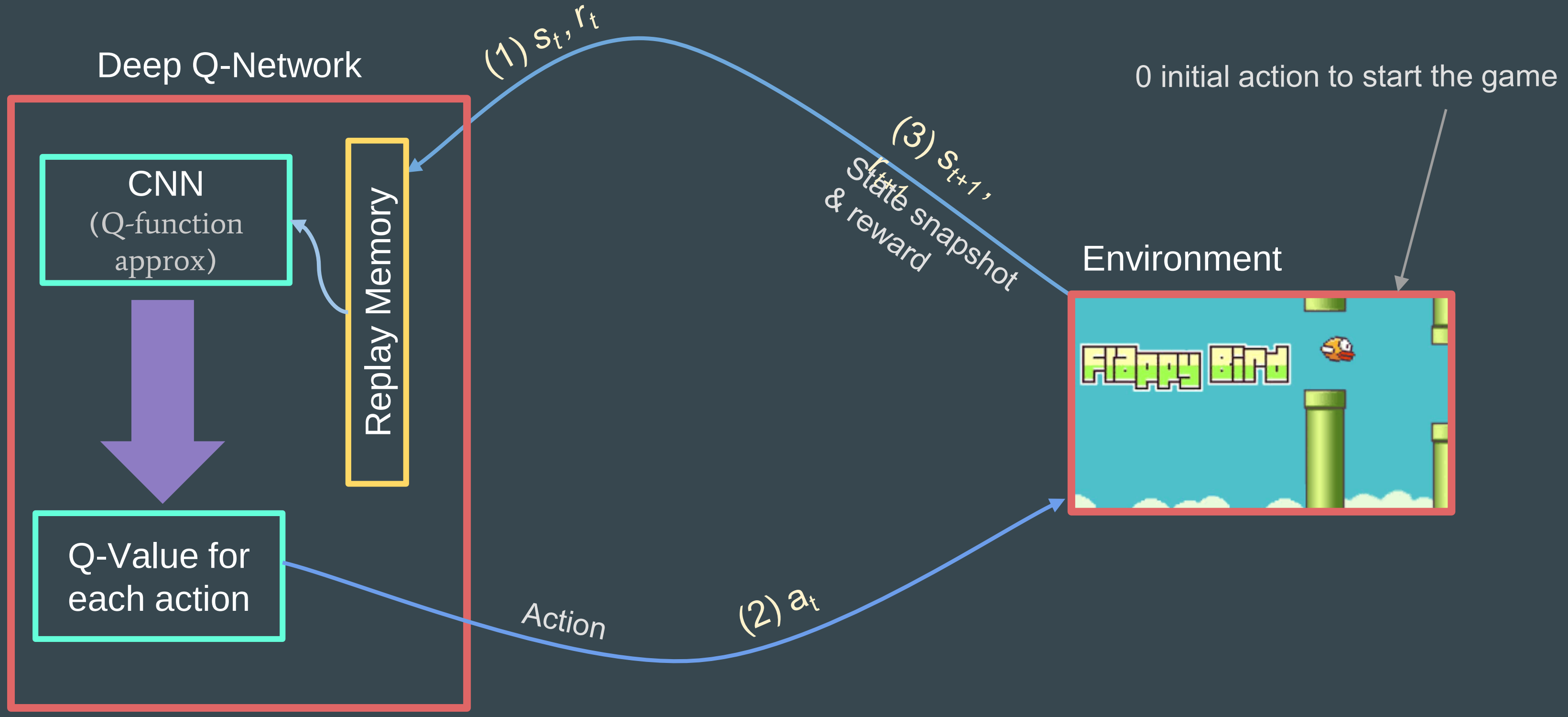
https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419

Problem Statement

- Environment: Flappy Bird (pygame)
- Goal: Continually keeping the bird alive while it's moving forward
- State *(s)*: Position of the bird in the environment (determined by the physics)
- Actions *(a)*: Tap the screen or don't tap the screen
- Rewards *(r)*:
  - +0.1 every timestep it lives
  - +1 every pipe it crosses
  - -1 very time it dies (terminal state)
  - Reward clipping for better generalization

(MDP - perform an action a_t at state s_t to receive reward r_t and next state s_{t+1})

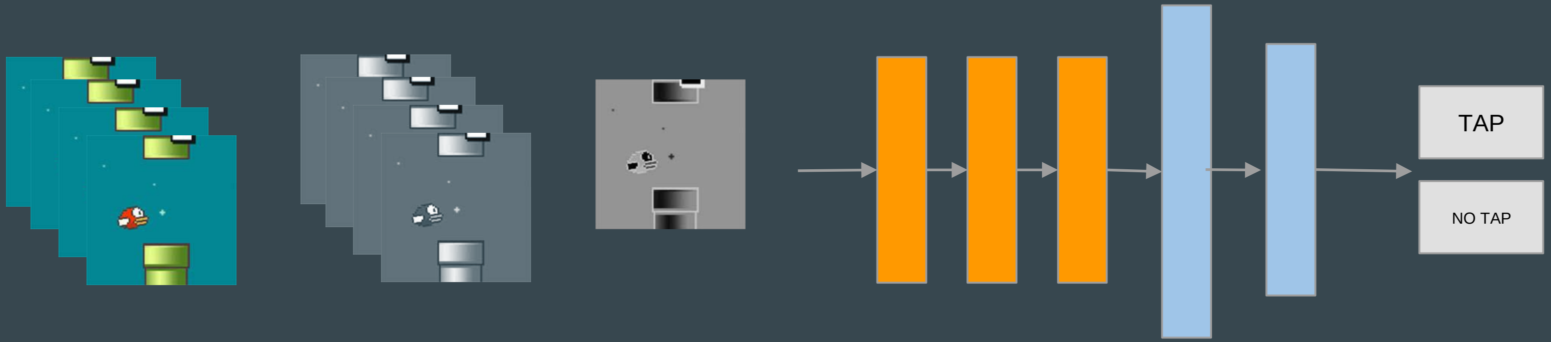
Architecture



Architecture (CNN)

- Preprocessing images for faster learning/training
  - RGB -> Grayscale
  - Resize to a smaller size
  - Stack 4 frames

$$L = \mathbb{E}[(\underbrace{r + \gamma max_{a'} Q(s', a')}_{\text{target}} - Q(s, a))^2]$$



C(8x8x4x**32**)A -> MP(2x2) -> C(4x4x32x**64**)A -> C(3x3x64x**64**)A -> FC(**1600**)A -> FC(**512**) -> FC(**2**)

Epsilon Greedy Policy (Exploration vs Exploitation)

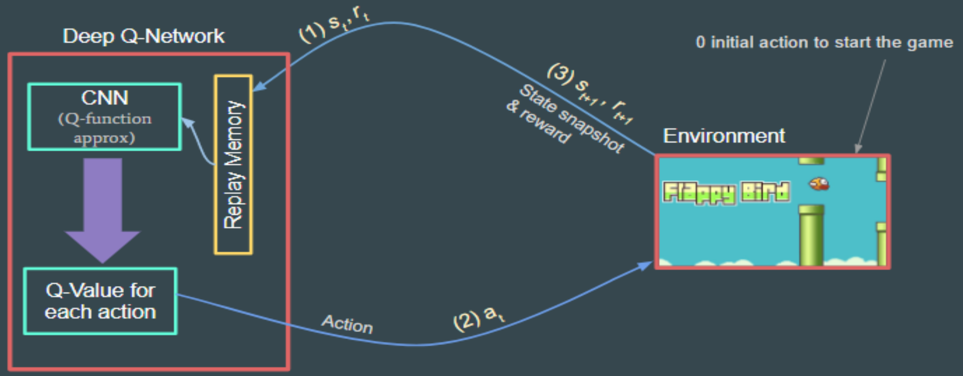
- To learn the optimal policy π*, the agent must see all the possible states
  - So it should be exposed to as many states as possible
  - But the agent has access to only states that are generated by its own action
- Greedy approach (max Q) is optimal within the space already explored
- Epsilon provides new info to the agent; the probability of taking a random action
- Will make decisions based in new data obtained through randomization
- Mostly epsilon in annealed in order to reduce the exploration

Experience Replay

- In reinforcement learning, we receive sequential samples from interactions with the environment, which are highly correlated and can lead to overfitting
- Experience Replay to the rescue!
  - Avoid forgetting previous experiences
  - Reduce correlations between experiences
  - Distribution averaged over many previous states, smoothing out the learning
- Used to train Deep-Q Network (DQN)

https://medium.freecodecamp.org/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8

Deep Q-Learning with Experience Replay



Initialize environment E

Initialize replay memory M with finite capacity N

Initialize the DQN weights w

for *episode* in *max_episode*:

  for *steps* in *max_steps*:

    s = Environment state

    Choose action a from state s using probability ϵ greedily

    Take action a, get reward R and next state s'

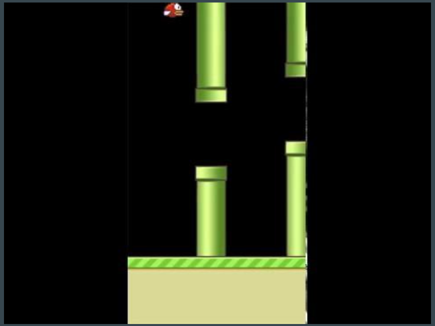
    Store experience $<s_t, a_t, r_t, s_t'>$ in M

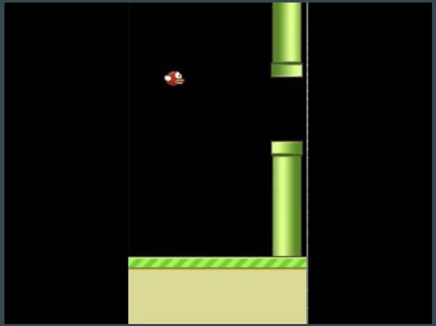
    Get random minibatch of experience $<s_p, a_p, r_p, s_p'>$ from M

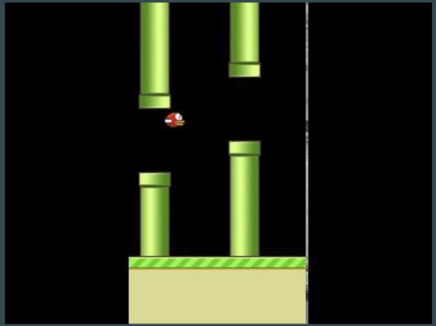
    Set optimal Q-value $Q(s_p)^* = r_p + \gamma \max Q(s'_p)$, where γ is discount value

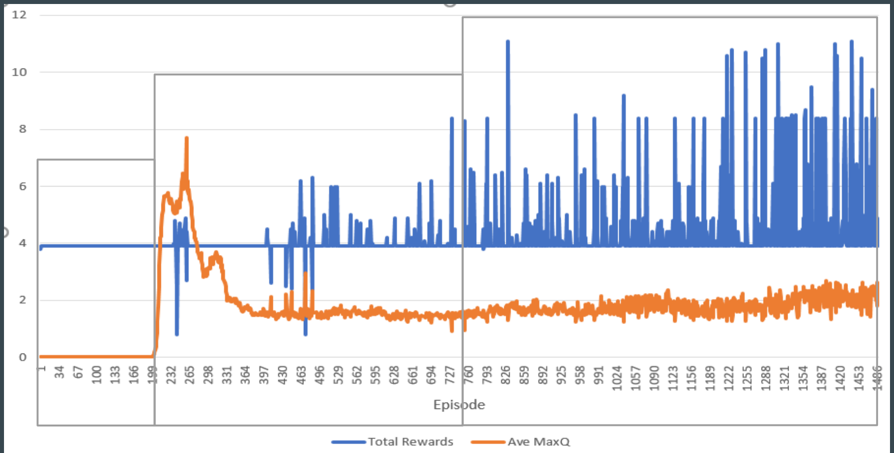
    Update Q by taking gradient descent step on $(Q(s_p)^* - Q(s_p))^2$

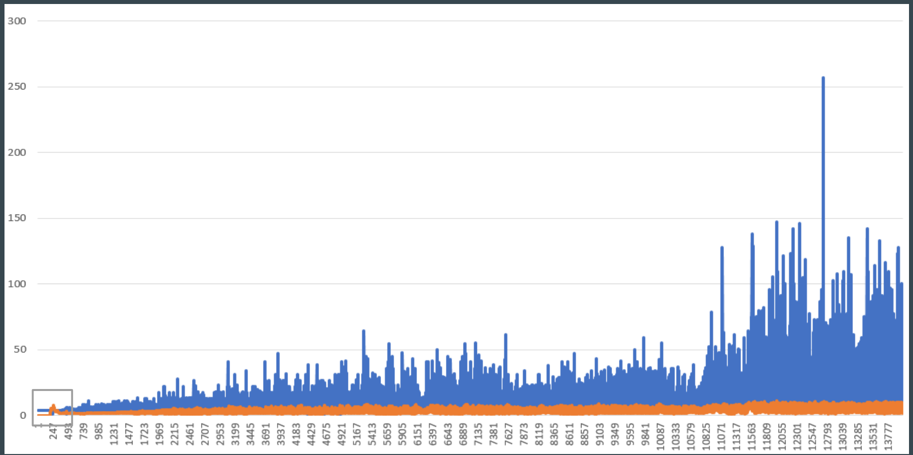
Flappy Bird Demonstration











Thank you!

References:

- Stanford Lecture: Reinforcement Learning
- Human-level control through deep reinforcement learning
- Playing Atari with Deep Reinforcement Learning
- Udacity: Reinforcement Learning GaTech
- Pygame implementation of Flappy Bird