Autonomous Flying Wafty Man

Fall 2014 CS 221 Project Final Report

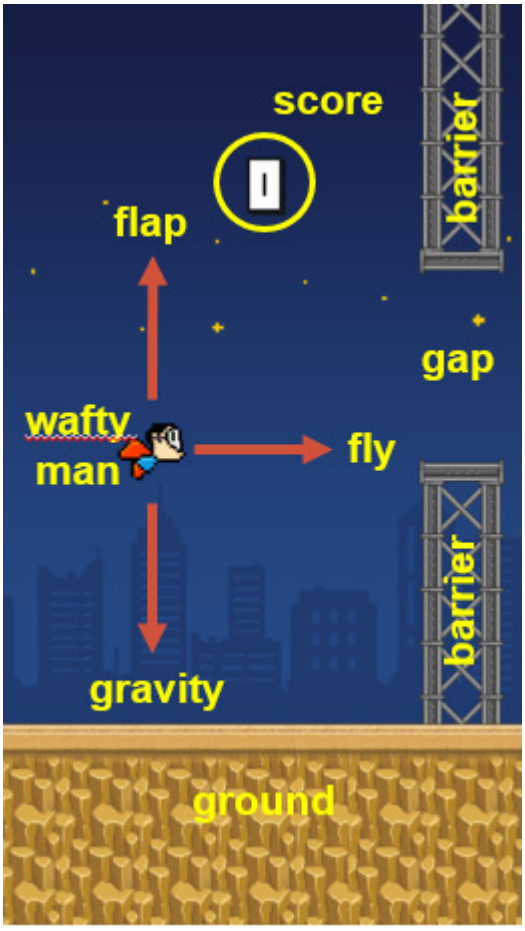
Joe Fan (joefan) and Vinod Kumar (vinodkum)

Introduction

Very few games are simple and yet addictive. One such game is Flappy Bird, a 2013 mobile game, developed by Vietnam-based developer Nguyễn Hà Đông (Dong Nguyen). Flappy Bird was removed from both Apple's App Store and Google Play by its creator on February 10, 2014. Games similar to Flappy Bird became popular on the iTunes App Store in the wake of its removal. And one such clone is Wafty-Man created by Mr.Speaker. He also open-sourced the game on github. We know it is fun (and sometimes annoying) to play but we bet it is even more fun to incorporate AI in it and see it play autonomously.

Task definition

Wafty Man is a popular side-scrolling game in which "Wafty-Man" flies horizontally at a constant velocity through an endless sequence of vertical barriers with gaps. Gravity continually pulls Wafty-Man down toward the ground, so the player must occasionally instruct Wafty-Man to flap (fly momentarily upward). One point is awarded for each barrier that Wafty-Man passes. The game ends when Wafty-Man crashes into the barrier or the ground. To achieve high scores, the player must instruct Wafty-Man to flap during the proper time (e.g. space, click, touch). The game can be played manually here. The task undertaken is to train Wafty-Man to choose either to flap or do nothing so that it stays alive as long as possible. Staying alive is orthogonal to scoring more points as the game always scrolls sideways nonstop.



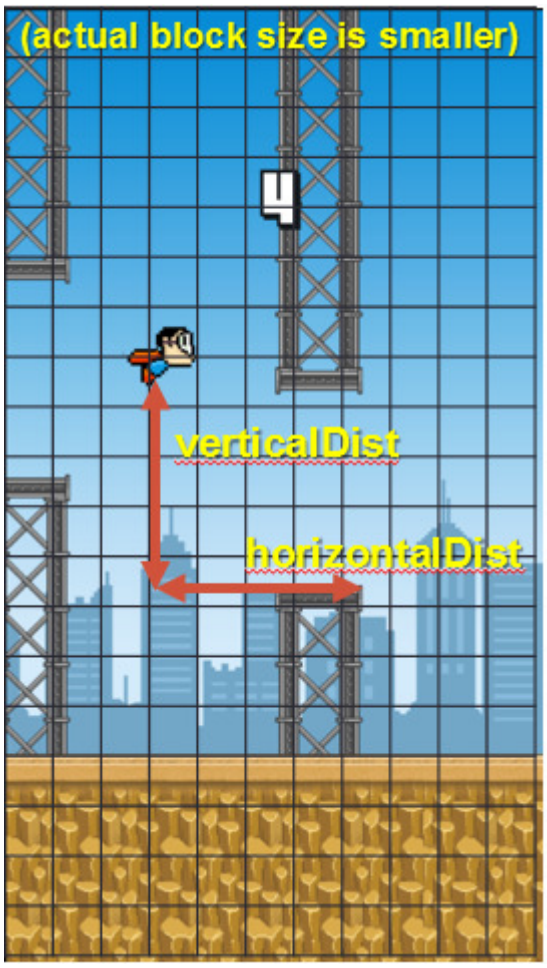
Approach

There are multiple approaches to solve the task. For example, one approach is to formulate it as a search problem by incorporating the knowledge of game engine. But decided to take a different approach by treating the game engine as a black box and enabling the game to play effectively and autonomously by learning in real time. We modeled the Wafty-Man game as a Markov Decision Process (MDP) and use reinforcement learning algorithm to learn to play autonomously. Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are under the control of a game engine. Reinforcement learning is an area of machine learning inspired by behaviorist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.

MDP definition

The Markov Decision Process (MDP) is defined as follows:

- States are a tuple of the vertical distance and the horizontal distance of wafty-man to the lower barrier.
  - s = (verticalDist, horizontalDist)
- The actions are either to flap or to do nothing.
  - Actions(s) = (flap, none)
- The rewards are structured to penalize heavily if Wafty-Man dies.
  - Reward $= \begin{cases} +1 & \text{Running} \\ -1000 & \text{Dying} \end{cases}$



Screen discretization

As the state space is too large due to the huge range of vertical and horizontal pixels, we divided the screen into smaller blocks. This drastically decreased the total number of states thus making learning possible. The new definition of states is:

- States are a tuple of the vertical distance, in blocks, and the horizontal distance, in blocks, of Wafty-Man to the lower barrier.
  s = (verticalDistInBlocks, horizontalDistInBlocks)

Q-Learning

The basic principle is that Wafty-Man performs a certain action in a state. It then finds itself in a new state and gets a reward based on that. The algorithm uses the reward to make better decision next time. We used Q Learning because it is a model free form of reinforcement learning. Hence we didn't have to model the dynamics of Wafty-Man; how it rises and falls, reacts to clicks and other things of that nature.

The policy to choose the action to perform is defined as follows:

$$\pi_{act}(s) = \begin{cases} \arg \max_{a \in Actions(s)} Q(s,a) & \text{probability } 1-\epsilon \\ \text{random from Actions}(s) & \text{probability } \epsilon \end{cases}$$

Once we gets the new state and reward from the game engine, we incorporate the data into the Q-Learning algorithm as follows:

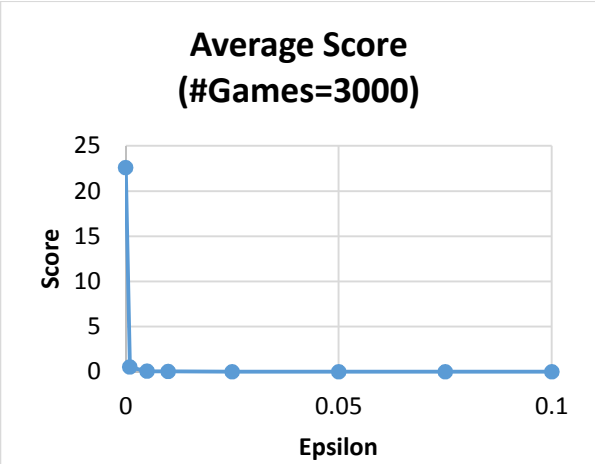
$$Q(s,a) = Q(s,a) - \eta \left[Q(s,a) - \left(Reward(s,a,s') + \gamma \max_{a' \in Actions(s')} Q(s',a') \right) \right]$$

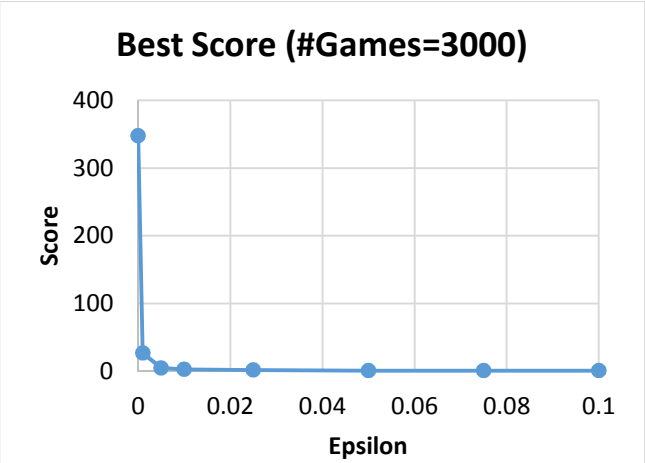
Experiments

Experiment 1: Tuning hyper-parameters

Tuning epsilon ϵ

We recorded the average and the best score attained by Wafty-Man for different values of epsilon ϵ for 3000 games. The values are graphed as below

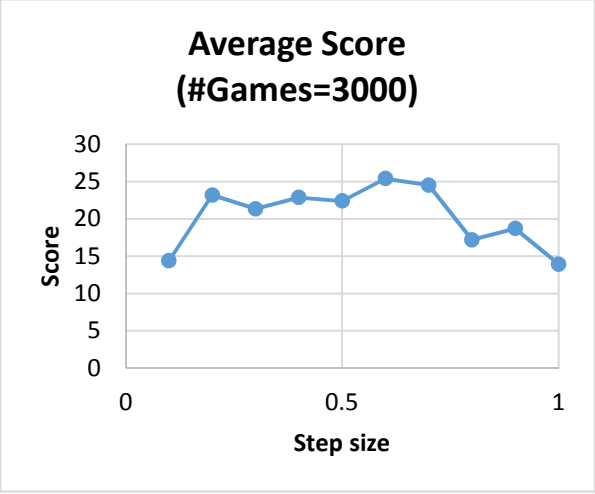


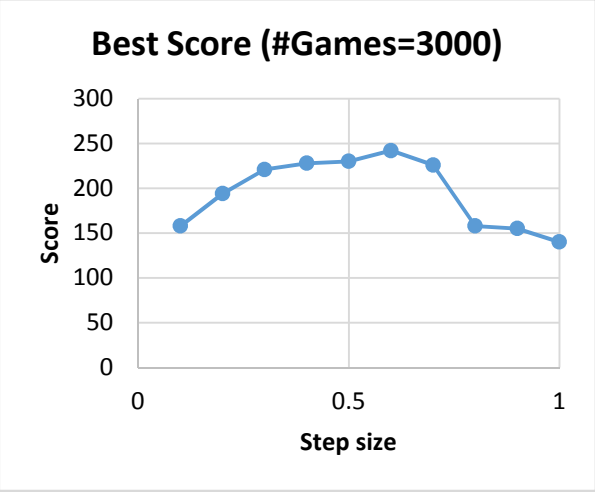


Based on the graphs above we can observe the Wafty-Man does not benefit from exploration. The game physics is exact and the geomentry is identical. Thus the best value of epsilon ϵ is 0.

Tuning step size η

We recorded the average and the best score attained by Wafty-Man for different values of step size η for 3000 games. The values are graphed as below

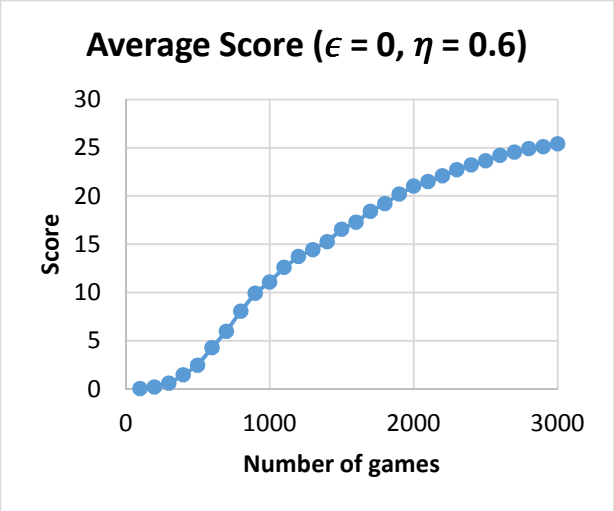


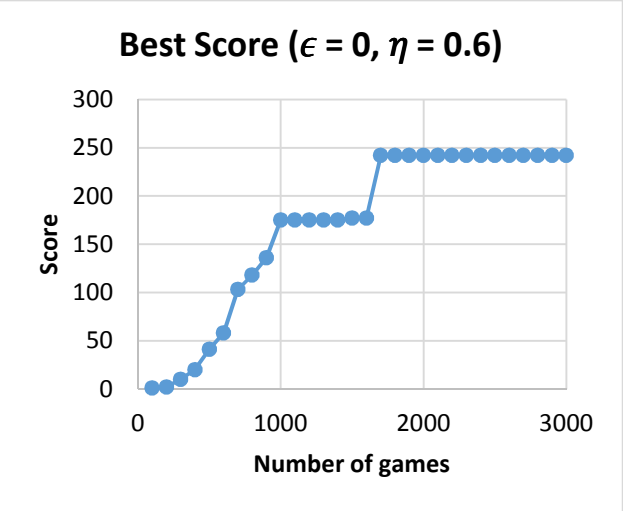


Based on the graphs above we can observe that both too large and too small step size is detrimental. The best value of step size η is 0.6

Experiment 2: Best hyper-parameters

With epsilon $\epsilon = 0$ and step size $\eta = 0.6$, we observed the average score and the best score of Wafty-Man for 3000 games. The values are graphed as below:



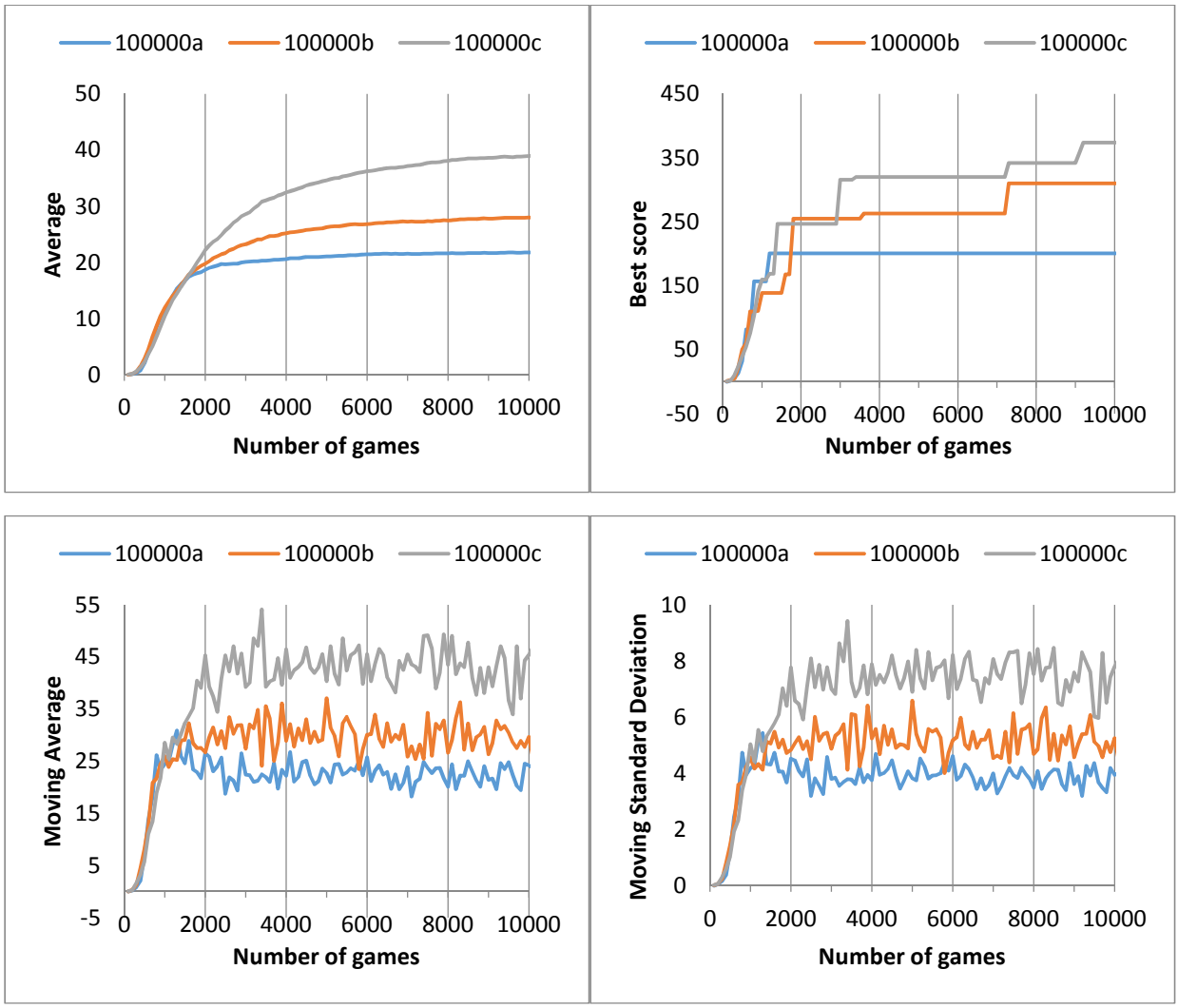


We can see that the average score increases as we train more indicating that Wafty-Man is learning effectively. The best score is a non-decreasing step function. For 3000 games, Wafty-Man was able to achieve a best score of 242 points with an overall average score of 25.4 points.

Error analysis

Now that we have built confidence that the basic Q-Learning implementation actually works, let us try to obtain the best Q values. Firstly, we should determine the requisite number of games to play in order for the performance of a given set of Q values to stabilize. In the above example, is the best score of 242 points a temporary plateau or an asymptotic best performance? As more games are played, the overall average score would tend to increase because the scores in later games would push up the average, but eventually would it reach some point of saturation as well? To answer these questions, we introduce two additional metrics: moving average and moving standard deviation. These metrics capture the average and standard deviation of the last 100 games, and they are provided at every 100 games.

A brute-force approach would be to just run Q-Learning for an absurdly long period of time. We completed 3 runs of 100,000 games (a big increase from 3,000 games earlier). Let us call these runs "100000a", "100000b", and "100000c." For the rest of this report, we introduce a simple nomenclature to append an alphabetical suffix to denote repeated runs of the same scenario. The following 4 charts display the results, albeit the x-axis is truncated at 10,000 games. By all metrics, the relative performance of these runs is: 100000a < 100000b < 100000c. The relative performance does not really matter because the result is expected to be random across multiple runs, but it is useful to note that there appears to be a good correlation between the 4 metrics, so they are all providing non-conflicting signals. What is important is that we run the 100000-game scenario a couple times to get an idea of the range in performance because we discovered earlier that this Wafty Man game MDP does not benefit from ϵ-greedy policy, so we should perform the exploration ourselves. Returning to our first question, we notice that the performance reaches saturation around 3,000 games.



Experiment 3: Beam Search - Finding good initial Q values

When $\epsilon = 0$, we take the risk of not exploring enough so ideally we should run a larger number of simulations. Constrained by resources, we had time to run the 100000-game scenario only 3 times and already experienced a wide range in performance. By this method, better-performing Q values are obtained by dumb luck via cherry picking Monte Carlo runs. Our next objective is to develop a method that could help us obtain good Q values consistently (vs. dumb luck), and we turn to beam search for achieving that goal. In the following we describe how to use beam search with Q learning, present an example, and compare the results with the 100000 game scenarios.

Beam Search methodology

In Q Learning, a fundamental mechanism is that when the reinforcement learning begins, the Q values are learned on top of the initialized Q values which typically begin at zero. If we wish to pause reinforcement learning, take a coffee break, and then resume where we left off, then we could have equivalently saved the Q values before the break and started a new Q learning session by initializing the Q values with the saved Q values. For example, we can play 3000 games and save the Q values, then load these Q values and play 4000 more games, which provides us the Q values for 7000 games. The initialization of Q values is the key building block of beam search.

The initialization of Q values leads us to define the search tree levels in beam search. Each level in the beam search corresponds to playing a certain number of games (e. g. $maxGames = 1000$). The number of games required for Q learning to reach saturation (e. g. $maxSaturation = 10000$) seems to be a good match for the maximum search tree level.

The beam search requires us to define the number of children per node (e. g. $b = 5$). So for example in a particular node in the 3rd beam search level, we would run the 2000-game scenario for 1000 games 5 times: 2000a, 2000b, 2000c, 2000d, 2000e. The K "best" Q values (heatmaps) are stored into the set (e. g. $K = 3$).

There remains one last definition. How do we rank the quality of Q values? The criteria for "best" Q values is quite arbitrary like the other beam search parameters, but consider that the other beam search parameters could be applied to any other games in a model-free manner whereas the "best" Q value metric is especially specific to the Wafty Man game. Therefore, we will defer its definition for the Wafty Man game in the following "Example" section.

Pseudo code

- Initialize C as empty set
- For i = 1 to $maxSaturation$ step $maxGames$
    - Extend
        - Initialize Q for b runs
            - If beam search level = 1, initialize Q with zeros
            - Otherwise, initialize Q with Q values from parent
    - Prune
        - Store the K "best" runs in C

Example

We ran the beam search using the following parameters:

- $maxGames$ = 1000
- $maxSaturation$ = 10000 (10 beam search levels)
- b = 5, K = 3
- The quality of Q values are ranked by the ordinary least squares $slope = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sum(x-\bar{x})^2}$
    - x = number of games per 100 games ($x = 100, 200, \ldots, 1000$)
    - y = moving average (average score in the most recent 100 games)

The following table shows the results of the beam search. There is one beam search level for each column. The three sets of rows correspond to $K = 3$. For each set of rows for each beam search level, we have a group of cells that shows the "extend" phase (illustrated by red and blue boxes). In each group, the title cell shows the beam search path leading to the parent, and the 5 subsequent cells show the OLS slope of 5 Wafty Man runs whose Q values were initialized by that of the parent's search path. After all 15 OLS slope values are recorded for the beam search level, we highlight the top 3 OLS slope values in green to indicate they are candidates for the next beam search level.

The next table shows the beam search paths for all levels.

# Games	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000

Initialized Q(s,a)	Zeros	c	cb	cbb	ecea	ecead	edcad a	edcad ae	edcbd edc	edcbd edcd
a	0.0250	0.0074	-0.0032	-0.0015	-0.0036	-0.0017	-0.0077	-0.0044	-0.0008	-0.0051
b	0.0265	0.0100	0.0068	-0.0027	0.0018	-0.0001	0.0031	-0.0026	-0.0010	0.0005
c	0.0360	-0.0034	-0.0016	-0.0033	-0.0014	-0.0001	-0.0007	0.0039	-0.0003	0.0043
d	0.0337	-0.0036	0.0013	-0.0018	0.0051	-0.0027	0.0018	0.0048	0.0072	-0.0050
e	0.0327	0.0000	-0.0011	0.0007	0.0005	-0.0005	0.0106	-0.0053	0.0039	-0.0040

Initialized Q(s,a)		d	ec	ece	edca	edcad	edcbd c	edcbd ed	edcbd ede	edcbd edce
a		-0.0060	0.0012	0.0031	-0.0033	0.0025	-0.0001	-0.0063	0.0023	0.0019
b		-0.0009	-0.0034	0.0010	0.0027	-0.0038	0.0013	-0.0047	0.0030	0.0006
c		0.0042	0.0011	-0.0004	0.0010	-0.0046	0.0001	0.0054	-0.0014	0.0003
d		0.0017	-0.0004	-0.0035	0.0056	-0.0060	0.0017	-0.0007	0.0087	0.0063
e		0.0061	0.0059	0.0022	-0.0009	-0.0006	0.0012	0.0086	-0.0060	0.0030

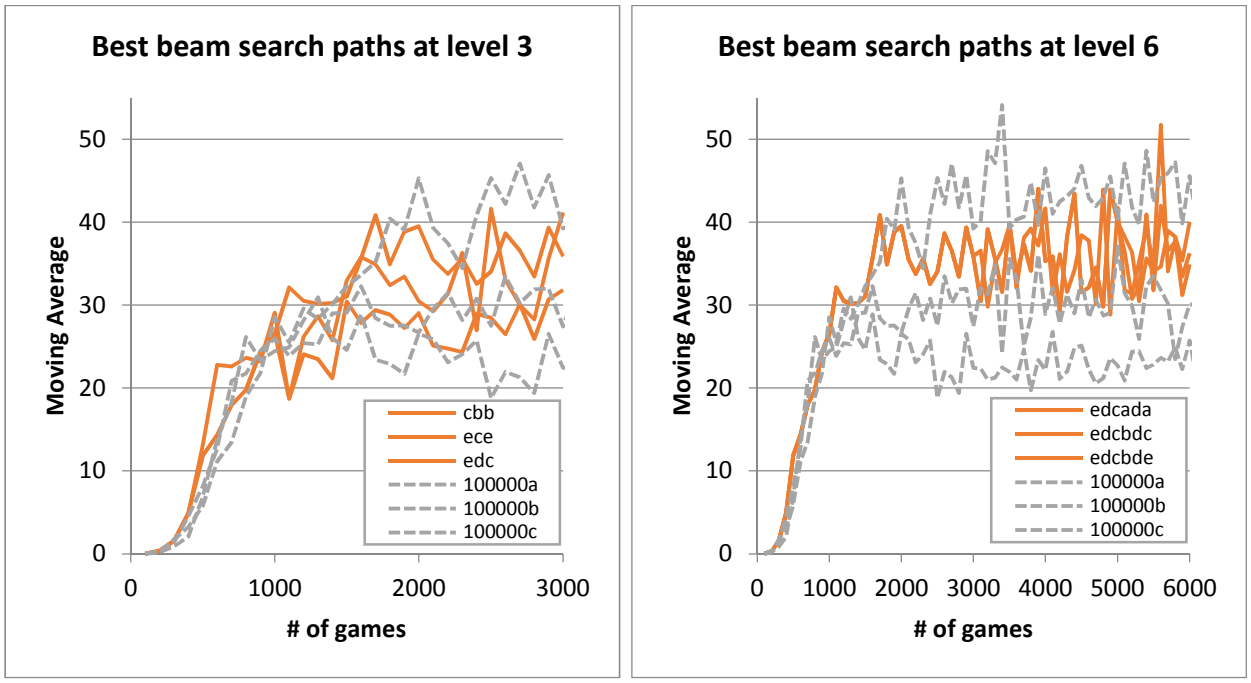
Initialized Q(s,a)		e	ed	edc	edcb	edcbd	edcbd e	edcbd ee	edcbd eed	edcbd eded
a		0.0000	0.0007	0.0070	-0.0021	0.0005	0.0017	-0.0038	0.0012	-0.0031
b		0.0042	-0.0019	0.0049	0.0025	0.0024	0.0019	-0.0024	-0.0056	-0.0008
c		0.0124	0.0030	-0.0002	0.0006	0.0035	0.0012	-0.0013	-0.0035	0.0028
d		0.0112	-0.0021	-0.0023	0.0043	-0.0031	0.0056	0.0054	-0.0010	-0.0065
e		0.0100	-0.0004	-0.0004	-0.0051	0.0025	0.0078	0.0033	-0.0008	0.0044

Levels	Beam search path
1	c
1	d
1	e
2	cb
2	ec
2	ed
3	cbb
3	ece
3	edc
4	ecea
4	edca
4	edcb
5	ecead
5	edcad
5	edcbd

Levels	Beam search path
6	edcada
6	edcbdc
6	edcbde
7	edcadae
7	edcbded
7	edcbdee
8	edcbdedc
8	edcbdede
8	edcbdeed
9	edcbdedcd
9	edcbdedce
9	edcbdeded
10	edcbdedcdc
10	edcbdedced
10	edcbdedede

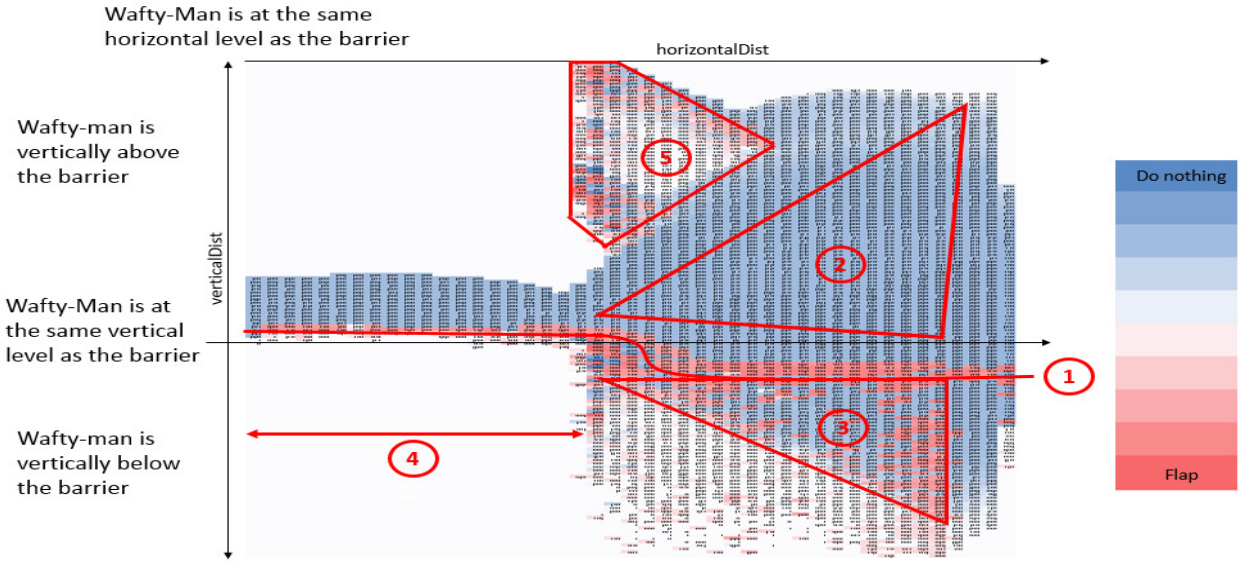
Error analysis for beam search

If there is an economic cost associated with running the game, then beam search can achieve significant cost reductions. We overlay the moving average of the $K = 3$ beam search paths at level 3 and level 6. It is evident that beam search is able to consistently achieve good Q values in fewer runs because the alternative relies on cherry picking off of large amounts of Monte Carlo runs. However, the beam search requires a few more levels for the best paths to converge and reach saturation (6,000 vs. 3,000 games), but that is a small price to pay for achieving consistently good results with fewer runs overall.



Experiment 4: Heat map

To better understand the learned values, we constructed a heat map of the Q values on a grid with rows mapping the relative vertical distance of Wafty-Man to the barrier and columns mapping the relative horizontal distance of Wafty-Man to the barrier. The values in the cells are the difference of the Q Value to flap at that state to the Q Value to do nothing at that state.



Let us now observe the five distinct sections marked on the heat map:

1 – Next barrier at same vertical level

The curved line marked as 1 represents the action Wafty-Man takes when it is relatively at the same vertical level as the barrier irrespective of the horizontal distance. We can observe that Wafty-Man flaps at the right time to maintain its vertical level similar to the incoming barrier which maximizes the probability to stay alive. It can also be interpreted as the plan of action Wafty-Man takes when the next barrier is at the same level as the previous one.

2 – Next barrier at lower vertical level

The triangle marked as 2 represents the action Wafty-Man takes when it is higher than the vertical level of the next barrier. Wafty-Man takes no action when it is higher than the next barrier thus falling and decreasing its altitude to match that of the next barrier. When it reaches the same vertical level as the next barrier it follows the plan of action represented by the curved line marked 1.

3 – Next barrier at higher vertical level

The triangle marked as 3 represents the action Wafty-Man takes when it is lower than the vertical level of the next barrier. Notice the red hue at the right edge of this region, at which its horizontal distance corresponds to the distance between barriers. When Wafty-Man switches its focus to the next barrier and realizes it is lower, it immediately flaps to increasing its altitude to match that of the next barrier. When it reaches the same vertical level as the next barrier it follows the plan of action represented by the curved line marked 1.

4 – Wafty-Man is inside the gap

Recall that the definition of the horizontal distance of the MDP state includes the width of the barrier. In this left-hand region of the heatmap, Wafty-Man is inside the gap and strives to stay alive by maintaining an altitude slightly above the lower barrier. The explored values in this region consist of a narrow corridor because the blank areas that are above and below it actually represents the barriers and therefore can never be explored.

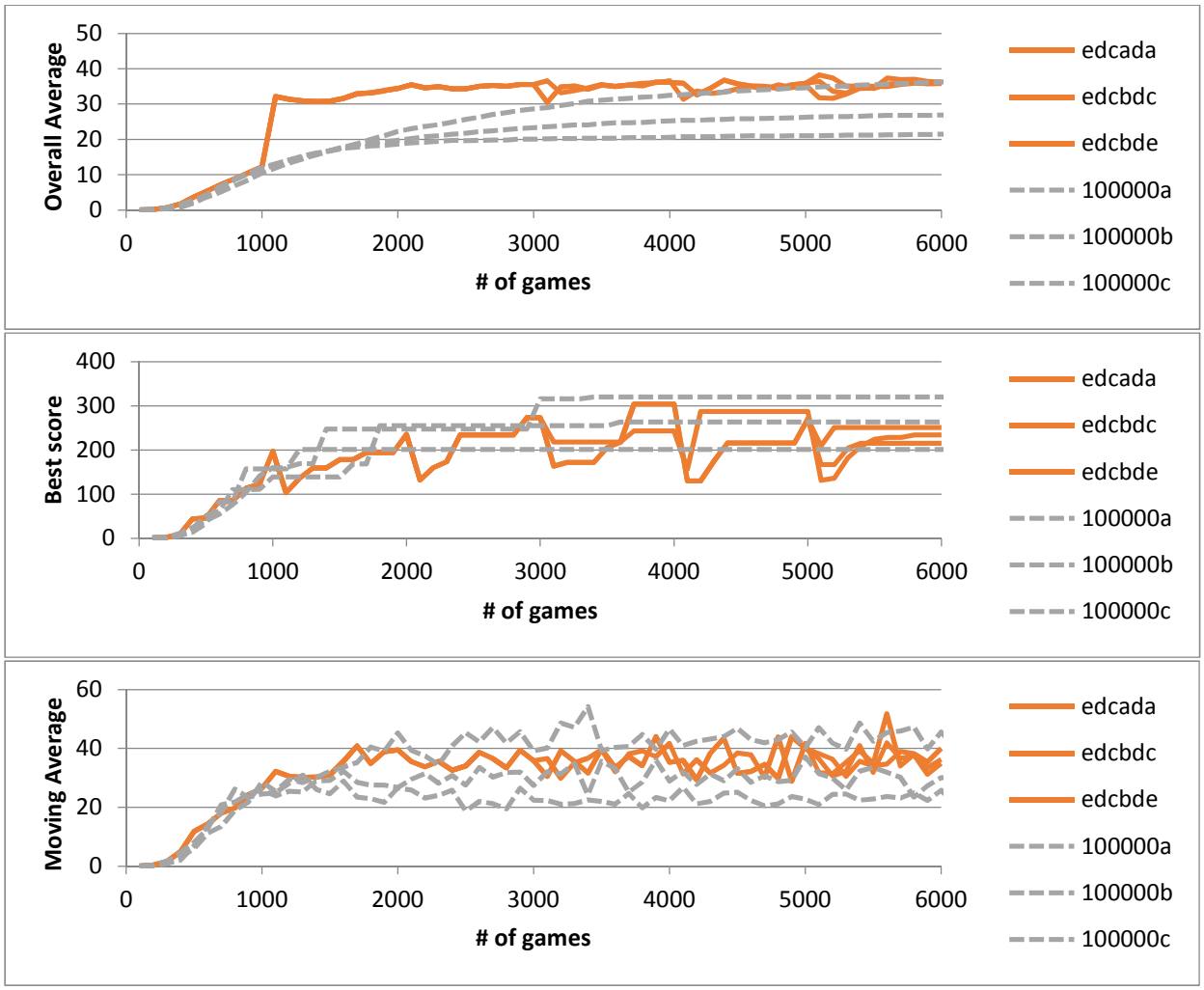
5 – Rage quit

In the top center region, we observe several diagonal stripes of red, which encourages Wafty-Man to flap when he is about to smash into the upper barrier. We have not yet identified this behavior but felt we should at least point it out. For now we call it “rage quit” because Wafty-Man flaps when it is certain to die. Maybe the AI performance can be improved if we prevent any learning in this area.

Result

In a nutshell, with the help of Q-Learning and beam search with $K = 3, b = 5$ and $level = 6$ to bootstrap the initial Q values, we were able to *consistently* achieve better scores in Experiment 3 vs. Experiment 2. But we also know that beam search becomes greedy as K decreases, so we achieve much better results than plain Monte Carlo simulation but cannot guarantee best results for $K = 3$.

# of games = 6000	Expr 2: Trials with no optimization			Expr 3: Beam search K=3, b=5, level 6		
	100000a	100000b	100000c	edcada	edcbdc	edcbde
Overall Average	21.5	26.9	36.3	36.2	36.2	35.8
Average of last 100 games	25.7	30.1	45.6	34.9	40.0	36.3
Best score	201	263	320	234	251	215



Future work

Starting from prior to the beam search discussion, the $\epsilon = 0$ hyper-parameter tuning results the lack of generalization results in rote memorization and slow training time. Also could Wafty Man benefit from taking into account multiple barriers rather than just the nearest one? An approach is to incorporate domain knowledge in feature vectors $\phi(s, a)$ and learn their weights:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \left[Q(s,a) - \left(Reward(s,a,s') + \gamma \max_{a' \in Actions(s')} Q(s',a') \right) \right] \phi(s,a)$$

There are many ways to tune the beam search parameters. And in the heatmap, we should identify the source for the 5th region ("rage quit").

Conclusion

We are excited to successfully implement reinforcement learning. Developing the beam search was an additional bonus because its integration with Q-Learning is also model-free and therefore can be generalized. We would like to convey our heartfelt appreciation to Percy Liang, Aparna Krishnan, and all the TAs for their guidance and support throughout the term.