3D Flappy Bird with Reinforcement Learning

Hang Qi
hangqi@cs.ucla.edu

Jian Gong
jiangong@cs.ucla.edu

Lunbo Xu
lunbo_xu@cs.ucla.edu

ABSTRACT

In this work, we present a reinforcement learning method to allow a artificial bird to survive in a randomly generated environment without collision with any obstacles by deciding when to flap the wings. The setting is based on a very popular mobile game called *FlappyBird*. We implemented it in 3D with WebGL, making it easy for public access and experiments with browsers. The bird is given two different perception models: (1) instant fovy, (2) fovy with short-term memory. In both models, the perception state is represented by a 16×1 vector, consisting of coordinates of the two pillars in front of the bird. A reinforcement learning algorithm Q-learning is used to learn a decision policy, powered by which the bird will be able to survive by repeatedly deciding which action to perform given its current perception state. The experiments show our Q-learning algorithm allows the bird to survive for more than 200 pillars.

Keywords

Artificial Life, Reinforcement Learning, Flappy Bird

1. INTRODUCTION

Recently, as the mobile game app Flappy Bird hitting the market and becoming popular, people complaint about how hard it is to accomplish the task preventing the bird from crashing into pillars. As human are trying hard to adapt the physics in the game, we found it is promissing to use reinforcement learning [8] algorithms to let the bird learns a decision rule by itself through interactions with the environment.

In this work, we modeled the physics in a graphics environment, simulate the perception of the bird by feeding in the relative coordinates of selected key points of the pillars, and quantize the relative position between the bird and pillars into a state space with reasonable size. A decision policy between to jump or not to jump at each state is learned using Q-learning algorithm[10].

In this report, we will briefly talked about the motivation and related work in section 2. Section 3 will discuss our formulation of the world and the learning problem. Section 4 will present the learning algorithm we used to learn the decision policy. Tools and implementation details will be presented win section 5. Finally, section 6 and section 7 will include our experiments and discussions on future directions.

2. RELATED WORK

Recently a mobile game app called Flappy Bird, hit the market and became very popular on Android and iOS platform. This game was originally created by Nguyen Ha Dong, a Vietnamese developer. In this game, a player can control a bird to make a jump by tapping the touch screen. The goal is to let the bird fly as far as possible between columns of green pipes without coming into contact with them. A plethora of Flappy Bird copies and clones emerged, such as Squishy Bird [4], Clumsy Bird [5], and even MMO version [1]. The popularity of the game not only originates from its simplicity, but also from the challenges it raises to human's adaptiveness to its dynamics.

Although the rule of the game is simple, it is agreed to be a challenging to make the bird fly over more than ten columns of pipes. It requires the player to keep sending commands to the bird precisely at the correct moments with high concentration and fast action.

Some works have been done to let the bird become intelligent enough to fly by itself. FlappyBirdRL [9] is one of the works which used a 2D model and reinforcement learning to teach the virtual bird fly. It assumes that the bird fully knows the configuration of the environment, including obstacles out of reach from the bird's sight, which is obviously not true for a real bird.

Thus, in our project, we try to overcome this limitation by explictly modeling its field of view. In our 2.5D world, we defined the field of view of the bird in vertical direction, which means the bird can only see things that is within a certain degrees of its sight. Particularly, anything column that is too high or too low will fall outside of the bird's field of view. Detailed modeling method is discussed in the following section.

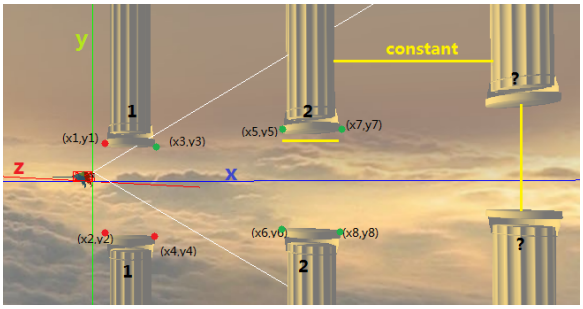

Figure 1: Occluded points or those not in fovy (red points) are unknown to the bird. The bird know only what it saw (green points) with instant fovy perception. The field of view of the bird is represented by white rays. RGB lines are the axises of the world coordinate system.

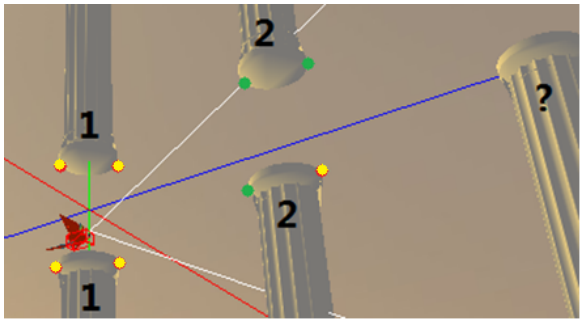

Figure 2: With short-term memory, the bird not only knows what it sees at the current position (green points), but also remembers what it saw(yellow points).

3. FORMULATION

In this section, we introduce our modeling and formulation of the world.

3.1 World

Although the bird is flying in a 3D world, the space is essentially represented in 2D. In our 2D model, the bird is represented as its bounding box, a $W_b \times H_b$ rectangle. The bird is given a constant horizontal velocity v_x to fly, whereas the vertical velocity v_y is controlled by gravity by default. Whenever the bird chose to perform a jump action, a vertical velocity increment Δv_y is added.

Cylinder pillars are represented by rectangles as well. The y positions of the gap between two pillars in the same column are generated randomly. The bird is dead and therfore game is over when the bounding box of the bird overlaps with pillars. To pervent the game to be too difficult to accomplish, however, the gap between pillars in the same column, the distance between two adjacent columns, and the width of the pillars are fixed to constant values (yellow lines in Figure 1).

3.2 Perception

To learn a decision policy, the input to the learning algorithm is the the perception of the bird of course. However, to simulate the real vision as a full-size image directly would have introduced too much complexities in segmentation, triangulation, 3D reconstruction, and vision feature extraction. Given that we want to focus our work in reinforcement learning, we consider the perception in two scenarios based on different assumptions.

3.2.1 Instant Fovy

In the first scenario, we assume that the bird can only see two nearest columns of pillars and cannot see occluded corners. Each column is abstracted as four key points (i.e. corners) as shown in Figure 1. Hence, eight coordinates relative to bird's eys position $\{(x_i, y_i)\}_{i=1}^{8}$ are used to represent the vision. For those occluded points or points not in the fovy (i.e. field of view y), they are marked *unknown* to the bird by setting $x = -1$ explicitly. $(1, -1)$ is used to represent the occluded corner above the bird, whereas $(-1, -1)$ is used for points below the bird.

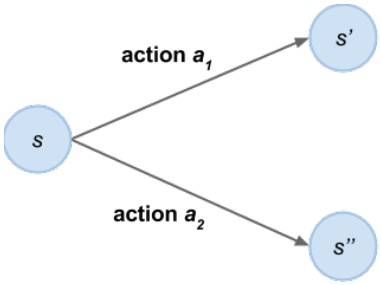

Figure 3: From a given state s, the bird can perform two actions (1) jump and (2) not jump, each of which leads the bird to a new state.

3.2.2 Fovy with Short-term Memory

In the second scenario, in addition to perception, we assume the bird has a short-term memory so that it will remember the existance of the corner once he saw it. In this case, we can simply give all the four corners to the bird without marking any occlusion explicitly since the bird will see all the points anyway as it jumps up and down. 2.

3.3 States and actions

Given the above discussion on perception, we can represent the *state* of the bird by a 16×1 vector

$$s = (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$$

consisting of coordinates (relative to the bird's eyes) of the eight key points of the nearest two pillars. Real numbers are rounded to their nearest tenth to limit the size of the state space. It is clear that in the case of instant fovy, the state space are smaller due to the occlusion marks.

The *action* leads to the transition between one state to another. In our configuration, the bird can only perform two actions: (1) jump, (2) not jump as shown in Figure 3. Give the current state s_t, the next state s_{t+1} can be easily obtained by computing the movement of the bird according to the action a_t performed.

4. LEARNING

To let the bird survive in the game, we want to learn a decision policy $a_t = \pi(s_t)$ which decides an action a_t for each state s_t that maximize the total reward accumulated over time t:

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}),$$

where γ is a discount factor, $R_{a_t}(s_t, s_{t+1})$ is the reward gained from action a_t which causes the state transition from s_t to s_{t+1}.

If we know the exact transition probability $P(s, s')$ and the reward of the states $V(s)$, we can solve the optimal policy that maximize the expected reward from the following equation:

$$\pi(s) = \arg\max_a \left\{ \sum_{s'} P(s, s')(R_a(s, s') + \gamma V(s')) \right\}.$$

However, in our game environment, we assume the bird has no prior knowledge about the world, so that the transition probability and rewards are unknown to us. This leads to the use of reinforcement learning[8].

In the framework of Q-Learning[10], a specific reinforcement learning algorithm, we want to learn a policy in the form of a quality table

$$Q : S \times A \to \mathbb{R},$$

which maps a state $s \in S$ and an action $a \in A$ to a real number $r \in \mathbb{R}$ that measures the reward of performing action a at state s. Once this quality table is learned, the decision can be easily made at any given state s as follows:

$$a^* = \arg\max_{a \in A} Q(s, a).$$

Starting from a arbitary Q, the algorithm of Q-learning update the table iteratively using equation

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \mathcal{F} - Q_t(s_t, a_t) \right],$$

where $\mathcal{F} = \max_a Q_t(s_{t+1}, a)$ is the maximum expected future reward at state s_{t+1}, α is the learning rate controlling the extend to which we want to forget the history, γ is the discount factor for the expected future reward. This equation ensentially says an action a_t will get higher reward if it lands to a state s_{t+1} that has high reward and we are not expected to die fast at that state. It has been proved that Q-learning algorithm is guaranteed to converge to the optimal decision policy.

5. GRAPHICS

This project is implemented with WebGL technique, which enables to render complex graphics on HTML5 canvas with JavaScript. Our project can run directly in the browser without the help of any other plugins efficiently. This makes it easy for public to access our project.

In this section, we'll introduce the major development tools, and some graphics techniques used in this project. All the graphics related functions and APIs are wrapped in the `graphics.js` file.

5.1 Tools

Three.js [2] is a lightweight cross-browser JavaScript library/API used to create and display animated 3D computer graphics on a Web browser . Three.js scripts may be used in conjunction with the HTML5 canvas element, SVG or WebGL. It greatly speeds up our development process.

Blender [3] is a free and open source 3D creation software, which is used in this project for building and converting 3D models, including the greek columns[1] and the flapping bird[2].

Chrome Developer Tools are used to assist the interface design and most importantly, to debug JavaScript functions and APIs.

Figure 4: Background scene is built with skybox. It consists of six properly aligned images.

5.2 Scene Rendering

The background scene in the game is created with SkyBox technique [6]. Cube mapping is performed to project images on to different faces of the cube. As shown in Figure 4, images were properly aligned on the box. As the cube moves with the camera, it creates an illusion of distant three-dimensional surroundings. In our implementation, a separate camera is used to build the skybox[3].

The animation in the scene is mainly controled by the enforcement learning module to decide the exact moment the bird need to make a jump. Whenever the bird jumps, the model of bird will play an predefined animation to flap.

The point light is placed close to the setting sun in the background. We also adjusted the colors of both point light and ambient light to render the scene more realistically.

6. RESULTS

Under the first senario where we assumpe the bird only aware points in the fovy, the bird's intelligence is evolving faster due to the limited size of the state space. Figure 5 plots the score against generation. Although the training speed is fast, after generation 16, it starts to converge to a non-promissing results. We believe this "degeneration" is due to the limitation of the features for reinforcement learning and the possible contradiction in the underlying randomly generated world.

In the second senario where the bird is given both fovy and short-term memory, the score is ploted against generation in Figure 6. In this case, our bird is able to fly further at its best performance. However, the training process takes much more time due to the large state space.

For the sake of easy debugging, we also implemented a quick training mdoe in which we were able to train without visualize the flying bird. Quick training take much less time

[1]The greek column model is downloaded from `http://archive3d.net/?category=555`

[2]The bird model is created by mirada from `ro.me`. [7]

[3]Images in skybox are created by Jochum Skoglund.

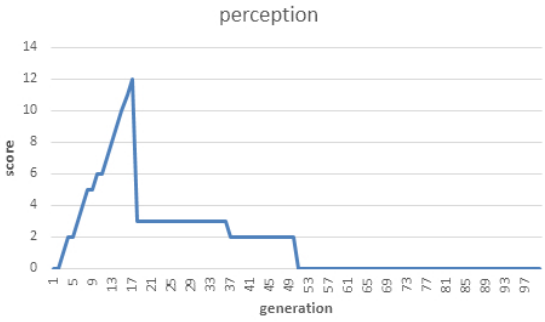

Figure 5: Reinforcement learning result if the bird aware only unocluded points in the *fovy*.

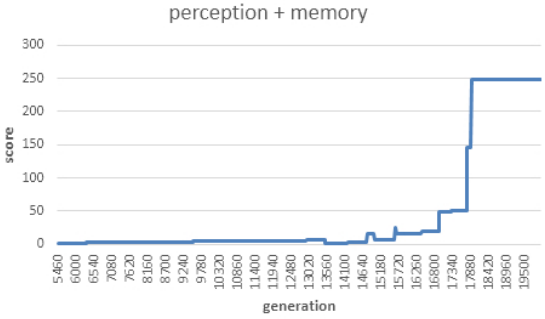

Figure 6: Reinforcement learning result if the bird aware all eight points by its sight and short-term memory.

since it cut off the time used for real-time graphics rendering. It only takes five minutes to quick training the bird to generation 18,000. However, it is interesting to witness the bird's growth from scratch (generation 0). Related video clips can be found on our project webpage.

7. FUTURE WORK

Our future work includes implementing real 3D environment and modification of the training method. First, instead of flying up and down only, we want to enable the bird turning left and right. Features extracted for 3D game environment are more complicated than the one we use for 2D game model.

Besides, we also want to improve the feature extraction for reinforcement learning. First, we need to add the verticle velocity v_y of the bird into the state space, since it is nature that the bird aware its own velocity and we strongly believe that some failure results of training were due to unawareness of this parameter. In addition, we would like to set a constant time interval, e.g. one second, between two consecutive decision to prevent the bird to flying up straightly.

8. ACKNOWLEDGEMENT

We want to give special thanks to Professor Terzopoulos who gives us a great course about artificial life and overview of related techniques. We had enough freedom when doing this interesting project and have learnt a lot during the process.

9. REFERENCES

[1] *Flappy Bird Massively Multiplayer Online.* `http://flapmmo.com/`, Feb. 2014.

[2] R. Cabello. *Three.js.* `http://www.threejs.org/`, Apr. 2010.

[3] B. Foundation. *Blender: Open source 3D graphics and animation software.* `http://www.blender.org/`, 1995.

[4] R. Games. *Splashy Fish.* `https://play.google.com/store/apps/details?id=it.junglestudios.splashyfish`, Jan. 2014.

[5] E. Leão. *A MelonJS port of the famous Flappy Bird Game.* `https://github.com/ellisonleao/clumsy-bird`, Jan. 2014.

[6] E. Meiri. *Tutorial 25 - SkyBox.* `http://ogldev.atspace.co.uk/www/tutorial25/tutorial25.html`, Oct. 2010.

[7] Mirada. *Dynamic Procedural Terrain Using 3D Simple Noise.* `http://alteredqualia.com/three/examples/webgl_terrain_dynamic.html`.

[8] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning.* MIT Press, 1998.

[9] S. Vaish. *Flappy Bird hack using Reinforcement Learning.* `https://github.com/SarvagyaVaish/FlappyBirdRL`, Feb. 2014.

[10] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.