Goldsmiths
UNIVERSITY OF LONDON

Department of Computing - MSc. Computer Games and Entertainment

IS71027B – AI for Games
Dr. Jeremy Gow

Term 2 - Assignment 1 (2013-2014)

Reinforcement Learning - Flappy Bird

Brian Gatt (ma301bg)

28 March 2014

Table of Contents

Introduction

The following is the documentation for an application of reinforcement learning on the popular game Dong Nguyen's 'Flappy Bird'. An overview of the attached deliverable is provided explaining the aims, background and implementation details. The appendices section contains proof of the resulting program.

Aims and Objectives

The main aim of this project is to apply and implement the AI technique of reinforcement learning. Dong Nguyen's popular 'Flappy Bird' game was chosen as a test bed to implement reinforcement learning in a simple game context. The following quote is the original project proposal which summarises the intent of the project:

> *The project will consist of a 'Flappy Bird' implementation using the Unity game engine and C#. The bird will be automatically controlled using reinforcement learning, in particular Q-Learning. The deliverable will provide a visualization of the learning process (on-line learning) with accompanying statistics in a text file format. Certain attributes will be made available for modification via the Unity editor interface.*

In the end, all of the intended objectives were achieved while also allowing for on-line learning from player input.

Background

Reinforcement Learning

Reinforcement learning is a machine learning technique which is prevalent in today's modern games. Agents learn from their past experience which in turn allows them to better judge future actions. This is achieved by providing the agent with a reward for their actions in relation to the current world state.

Reinforcement learning is based on three important factors:

1. The reward function – A function which rewards or punishes (perceived as a negative reward) the agent for the action he just performed.
2. The learning rule – A rule which reinforces the agent's memory based on the current experience and reward.
3. The exploration strategy – The strategy which the agent employs in order to select actions.

The reward function, learning rule and exploration strategy generally depend on the current world state; a set of variables which define what the learning rule will take into consideration to adapt.

One popular implementation of reinforcement learning is Q-learning. It is a learning rule which evaluates Q-values, values which represent the quality score of a state-action tuple. The learning rule returns a Q-value by blending the prior state-action Q-value and the best Q-value of the current state. This is represented as:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max(Q(s',a')))$$

The α variable is the learning rate which linearly blends the Q-values, r is the perceived reward and γ is the discount factor which defines "how much an action's Q-value depends on the Q-value at the state (or states) it leads to." (Millington & Funge, 2009). Both α and γ are bound between 0 and 1 inclusive.

Reinforcement Learning requires time in order to train to a state where agents can react in reasonable manners. This is all dependant on the parameters chosen for the learning rule, the rewards and how the world is encoded (discretizing the world state in a reasonable manner). The data structures used to store this information can also hinder the experience.

Design

The design for this implementation closely followed GitHub user's SarvagyaVaish implementation (Vaish, 2014).

Vaish uses Q-Learning in order to train the 'Flappy Bird' character using a reward function which rewards the bird with a score of 1 on each frame the bird is alive. Once the bird dies, the reward function heavily punishes with a score of -1000. The learning state is defined as the horizontal and vertical proximity between the bird and the upcoming pipe hazard. The bird is allowed to do any of the available actions (Do Nothing, Jump) at any time during the simulation.

We took some liberties in relation to Vaish's implementation in order to make the implementation easier and adapt it to the Unity engine. One clear example being the data structures used to store the character's experience. Vaish uses a multi-dimensional fixed size array based on the maximum distances between the bird and the pipes on the horizontal and vertical axis. He then creates a basic hash mechanism and stores Q-values in this data structure, overwriting values which exceed the minimum and maximum in the lowest or greatest element respectively. In our case, we use a simple map or dictionary and store the experiences accordingly.

Implementation

Following are some details on how to configure the AI parameters and how the implementation is defined in terms of the major Unity game objects, components, and behaviours.

Scene

The scene mainly consists of the main camera, the 'Flappy Bird' character and hazard spawn points. The 'GameController' game object is an empty game object which is used to control the overall game. The 'Destroyer' object is used to destroy previously instantiated hazards in order to keep the game object instance count to a minimum. The 'Cover Up' game object is essentially a cover layer which hides what is occurring behind the scenes so that users are not able to see hazard spawn points and destruction points. Finally, the 'ceiling' game object was necessary to avoid cheating. During implementation, there were cases where the bird was exploring areas beyond its intended space, allowing him to stay alive yet not achieving a better score since he was, literally, jumping over the pipes.

Configuration

AI parameter configuration is achieved by modifying the variables within the 'Player Character' script component attached to the 'Flappy Bird' game object. Below is a screenshot of the mentioned component.

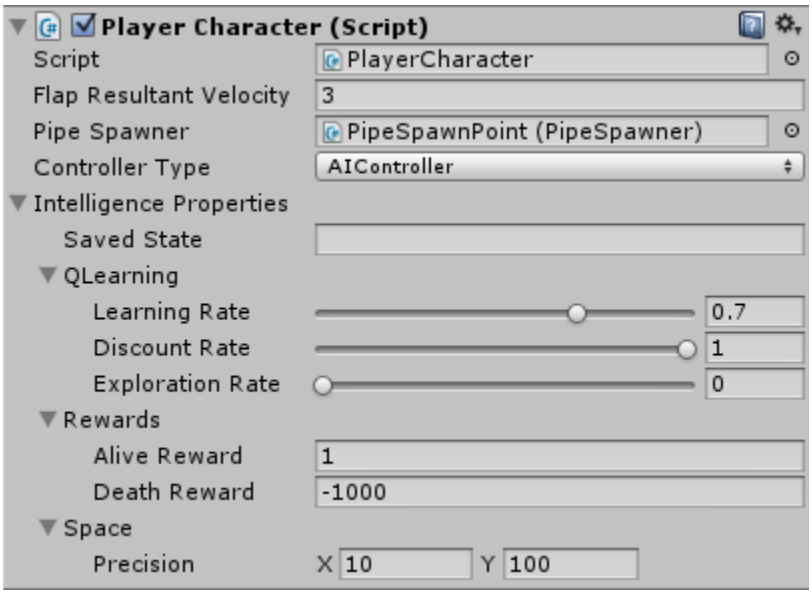


Figure 1: 'Flappy Bird' Configuration

The 'Controller Type' variable lets the user choose between 'InputController', 'AIController' and 'HybridController'. The 'InputController' is an implementation artifact which was used to test the game mechanics. It does not learn and only reacts to user input. The 'AIController' is the reinforcement learning controller strategy which learns and acts on its own. The 'HybridController' is an extension of the 'AIController' which learns but also allows user to provide his own input to allow the learning algorithm to learn from a different source.

The Intelligence properties expose the learning parameters previously mentioned in the 'Background' and 'Design' sections. Note that the 'Saved State' field is an implementation artifact. It was intended to allow learning to be resumable from different sessions, alas, encoding the state of the learning algorithm within the log file became unwieldy due to excessive file sizes so it was abandoned. The 'Precision' field specifies the scale for how many decimal places for the bird-pipe proximity are taken into consideration.

The 'GameController' game object exists solely to host the 'Game Controller' script controller which hosts minor configuration parameters for the overall game. Below is a screenshot of the mentioned component:

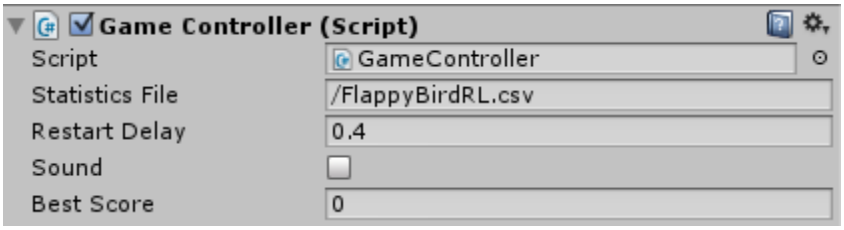


Figure 2: 'GameController' Overall Configuration

Please note that the 'Best Score' field does not affect the game per se but is only there as a visual cue to keep up with the best score recorded.

To speed up the simulation, simply modify the Unity time engine parameter from 'Edit' – 'Project Settings' – 'Time' and modify the 'Time Scale' parameter.

Classes and Interfaces

Following is a brief overview of the major classes and interfaces which compose the overall implementation.

- `GameController`

Manages the overall game flow by managing the state of the game and restarting accordingly. It is based on the singleton pattern and is persisted across scene loads. It stores the state of the AI algorithms on death of the player and restores them once the level is initiated.

- `PlayerCharacter`

Contains the behaviour of the player character. The strategy pattern is used to represent the controller implementations and is switched on scene load accordingly. The update event delegates to the underlying controller implementation.

- `AIController`

A `Controller` implementation which uses the Q-Learning algorithm to store and base the actions of the player character.

- `CoalescedQValueStore`

An `IQValueStore` implementation. This follows Millington's and Funge's (Millington & Funge, 2009) recommendations by coupling the state and the action as one entity. This implementation follows the `QValueStore` implementation which was used in earlier stages of development. We were initially afraid that the original version had multiple hash collisions so we implemented the coalesced version which provided better, more stable, results.

- `QLearning`

The implementation of the Q-Learning algorithm according to Millington and Funge.

User Manual

Prerequisites

Please ensure that Unity is installed on your system. This project was developed and tested using Unity version 4.3.

Launching the Project

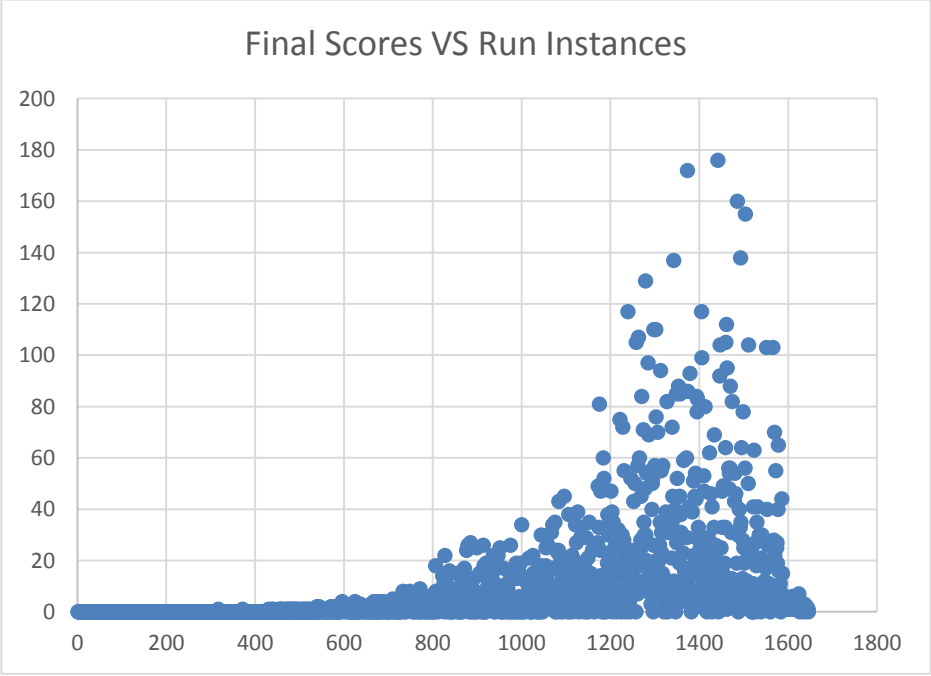
In order to launch the project, double-click on the 'MainScene' scene file located in 'Assets/Scenes/'. Alternatively, open Unity and via the 'Open Project…' menu item, navigate to the top-level directory of the project and launch it from there.

Initiating the Simulation

In order to start the simulation, simply click the 'play' button in the Unity editor. Ensure that prior to starting the simulation, the parameters are set up correctly. Certain parameters can also be modified mid-run. It is recommended that for mid-run parameter modification, the simulation is paused via the 'pause' button in the Unity editor so that it is easier to modify AI parameters accordingly.

Evaluation

Based on our implementation and the generated log files (refer to 'Appendices'), it takes time for the character to learn the problem. Important elements which defines the learning algorithm are the space quantization parameters and the underlying data structures used to store the experiences. Imprecise space quantization precision can lead to faster results but the character will start to generalize quickly. On the other hand precise values lead to longer training but more fine-tuned results. Following is a chart which shows the best run we managed to achieve (refer to the attached 'FlappyBirdRL.best.1x.csv' for a detailed overview):



Whether or not reinforcement learning is useful for this type of application is debatable. According to the one of the comments on SarvagyaVaish repository, 'Flappy Bird' uses a deterministic physics model where both the height jump and width can easily be computed.

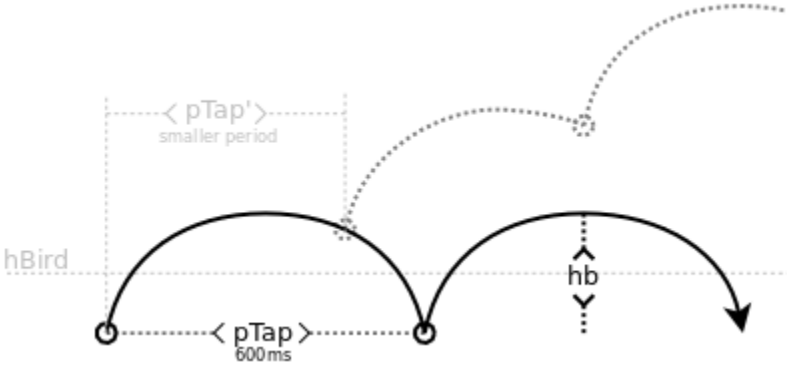


Figure 3: Flappy Bird's deterministic physics model (Vaish, 2014)

Based on these two values, a simpler scheme can be used. Another project (Jou, 2014) shows another implementation of this concept but focusing on computer vision. We believe that this project exploits the deterministic physics model in order to achieve its results.

Conclusion

Reinforcement learning is an AI technique which allows agents to learn and adapt via a reward-punishment system. This technique is implemented and applied on Dong Nguyen's 'Flappy Bird' and its implications are evaluated. During development, testing was continuously performed in order to ensure that the requirements are met and the deliverable is of a high quality. The appendices section shows a running demonstration of the artefact.

References

Jou, E. (2014, February 24). *Chinese Robot Will Decimate Your Flappy Bird Score*. Retrieved from Kotaku: http://kotaku.com/chinese-robot-will-decimate-your-flappy-bird-score-1529530681

Millington, I., & Funge, J. (2009). *Artificial Intelligence for Games.* Morgan Kaufmann.

Vaish, S. (2014, February 15). *Flappy Bird RL*. Retrieved February 21, 2014, from Github Pages: http://sarvagyavaish.github.io/FlappyBirdRL/

Appendices

List of Included Log files

- FlappyBirdRL.3.x1 – Sample log file (using non-coalesced QValueStore).
- FlappyBirdRL.4.x1 – Sample log file using different parameters (using non-coalesced QValueStore).
- FlappyBirdRL.coalesced.x4 – The log file generated from a 4x speed up when using the `CoalescedQValue` store.
- FlappyBirdRL.video.x2 – The log file generated by the run which is shown in the video linked below.
- FlappyBirdRL.best.x1 – The log file generated by what we consider, the best (and longest) run, in which 176 pipes are recorded as the best score when using the `CoalescedQValue` store.

Source Repository

https://bitbucket.org/briangatt/flappy-bird-rl-unity

Video Demonstration

http://youtu.be/H9y9sfFHen0

The video shows a demonstration of the attached deliverable. The game was sped up by a factor of 2 in order to keep footage short while showing the concept of reinforcement learning applied to 'Flappy Bird'.

Screenshots

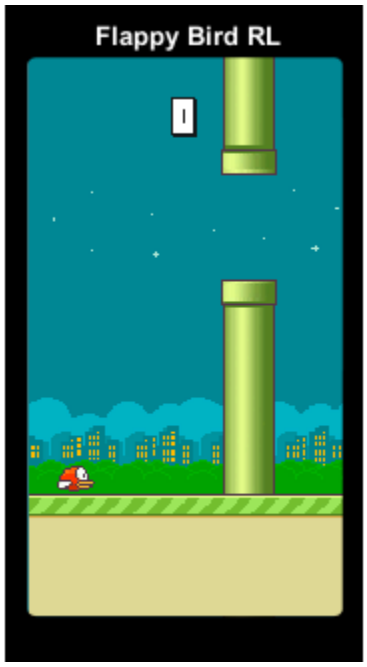


Figure 4: Screenshot

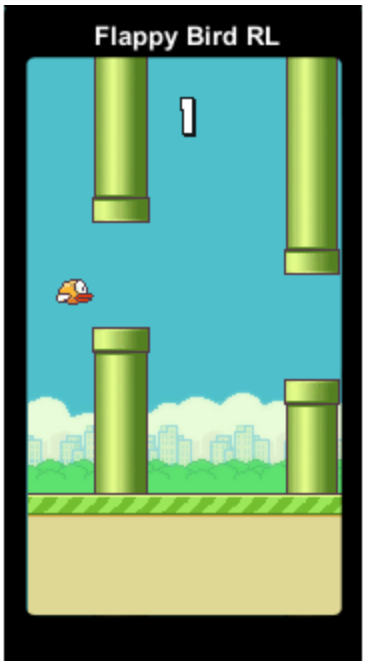


Figure 5: Screenshot

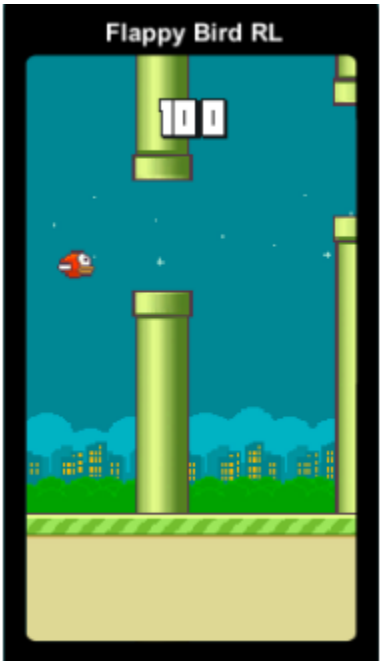


Figure 6: Screenshot