Game-AI on ‘Color Switch’ Using Q-Learning Algorithm

Introduction

For this project, the main focus is to develop an GAME-AI that can play the game of ‘Color Switch’ using Q-Learning Algorithm.

‘Color Switch’ is a game developed for smartphones. Players are a dot in the game, with each tab on the screen, the dot will move up the screen a bit. Every time when the dot eats a star, it will get one extra point. But the twist of the game is that, there are obstacles that separate the player from the star. Thus, players need to go through them to win a point, the dot can only touch obstacles with the same color, otherwise the dot will explode and game over. One addition variety of the game is a item called ‘pie’ that will switch the color of the agent randomly to the three other colors. No ending can be reach for this game, the main goal of the game is to score as much point as possible.

Motivation

A number of studies related to developing a GAME-AI on another similar open-ended game ‘Flappy Bird’ have been conducted with promising results. [https://github.com/chncyhn/flappybird-qlearning-bot](https://github.com/chncyhn/flappybird-qlearning-bot) is one of the examples that use Q-Learning as the algorithm to train the game-bot which was able to score up to 4000 points.

There are number of studies related to GAME-AI development as game is relatively stable and the result of the AI can easily be measured. It is an efficient way to practice Q-Learning through open-ended game. Implementing ‘ColorSwitch’ is also challenging due to the uncertainty of random color change and mobile obstacle.

Task Definition

As this game is an open-ended game, the criterion for a successful GAME_AI will be a bot that is able to score over 20 points after reinforcement learning.

Literature Review
Flappy Bird is a popular mobile game which the player needs to pass the bird through the obstacles in order to get the score. There are some game AI which were created for this game to get very high score. Below is a game AI for Flappy Bird, which has been released on github. https://github.com/yenchenlin/DeepLearningFlappyBird
This game AI focused on the specific game, Flappy Bird. However, we need to apply q-learning into the other reactive game which is different from Flappy Bird.

**Infrastructure**

The main language we are going to use is python with 2.7. In order to train the bot using python, first a desktop version of the game need to be recreated. A simplified version was found on the web with some bugs. After fixing those bugs the game was able to run smoothly. There are four main classes in our simulator:

<table>
<thead>
<tr>
<th>Object</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ball</td>
<td>Position (x &amp; y), Color, radius</td>
</tr>
<tr>
<td>ColorSwitch</td>
<td>Position(x &amp; y), radius</td>
</tr>
<tr>
<td>Obstacle</td>
<td>Position(x &amp; y), Color, arc_thickness, rotational velocity</td>
</tr>
<tr>
<td>Star</td>
<td>Position(x &amp; y), Color, Width, Height</td>
</tr>
</tbody>
</table>

**Ball**
The main player agent, the acceleration of the ball is default to negative one, with each action the velocity will be set to 6. Thus, creating a jumping effect.

**Colorswitch**
Item that changes the color of the ball. When collision is detected between the ball and the ‘colorswitch’, the color of the ball will change to a random color and the item will disappear and regenerate in other position.

**Obstacle**
A shallow circle that has two different colors equal divided the arc with arc_thickness. The angle of the obstacle is 0 originally, with every update of the screen the angle will be incremented by the rotational velocity (1 in this case), to mimic the rotation motion. When collision is detected with the ball, if the arc is as the same color of the ball then nothing happened, else the ball will be destroyed and game over.

**Star**
Item that increments the score by one. When collision is detected with the ball, score increment by 1 and the item will disappear and regenerate in a new position.
Main Algorithm

Reinforcement learning is one of the algorithms in machine learning, which focuses on teaching the machine to act in a way that would maximize its expected utility. RL is usually done with a trial and error basis by providing the machine with a large number of samples, also called episode.

In RL, Q-Learning is one of most frequently used. The basic idea of Q-Learning is to allow the machine to learn the expected utility (or reward) of each state of the program with respect to each possible action taken. The formula below shows how Q value is calculated.

\[
Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]
\]

\(Q_{k+1}(s,a)\) refers to the Q-value with state \(s\) of action \(a\), \(k+1\) refers to the iteration. \(T(s,a,s')\) refers to the probability of the of the next states \(s'\), given \(s\) and \(a\). \(R(s,a,s')\) refers to the reward the agent is going to obtain given \(s'\), \(s\) and \(a\). \(\gamma\) refers to the discounting factor of Q-learning algorithm. The whole term \(\gamma \max_{a'} Q_k(s',a')\), refers to getting the maximum Q-value in the next state in the previous iteration. By iterating the agent through a large number of episodes, the agent will be able to learning all the Q-value and perform optimally given the states.

GAME AI

The Game AI developed for the project is based on Q-Learning algorithm. A class bot is created to handle different tasks during the process of learning. The states of the process is defined using four parameters.

Table 2: States parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ob_x</td>
<td>The horizontal position of the obstacle arc with the same color as the ball</td>
</tr>
<tr>
<td>Ob_y</td>
<td>The vertical position of the obstacle arc with the same color as the ball</td>
</tr>
<tr>
<td>Ball_y</td>
<td>The vertical position of the ball and the star</td>
</tr>
<tr>
<td>vel</td>
<td>The velocity of the ball</td>
</tr>
</tbody>
</table>

Since the agent of the game only moves through the vertical axis of the screen therefore for star object, the x position need not to be worried. However, this is not the case of obstacle due the rotation. In order of the agent to know the location of the arc x position will need to be provided as well. For the state definition, color is being left out, even tho it is the main variation of the game, because it has already been account for in the distance between the obstacle as the agent will only see the obstacle with the same color. Another important components of Q-learning is the reward function.

Table 3: Reward function

<table>
<thead>
<tr>
<th>Event</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Error Analysis

One of the problem arise during the training was that the agent learning to stay alive without dying but the agent was not proceeding in any sense, which created a situation that the agent was jumping in its original position infinitely. In order to solve this a maximum threshold of step taken was set before the iteration is terminated.

### Result

For the Q-Learning algorithm, based on the condition given, the machine was able to achieve up 7-8 points in the first few iteration. However, after 200 iteration, the score will converge to 1-2 points. The reason of the phenomena was not found during the period of the study. Thus further study on the algorithm will need to be study.

### Baseline Comparison

To evaluation between different algorithm, a default policy was used for a agent in a trail and error basis. The agent, initially, before experiencing death, will always choose to go up. After it was killed at a position, that position with time interval will be marked as dangerous and will avoid entering that position at that moment. By accumulating the dangerous marks, the agent will be able to wait for the same colour and pass through obstacles. With this policy, the agent was able to achieve up to as much as points as possible, given a certain period of time for “try and error”. Compare to the Q-Learning agent introduced in this project, the standard policy approach seems to be a better choice to choose from.

### Conclusion

This study will need further work in designing the state and reward function.