Team 9 Final Report
Game AI using Reinforcement Learning: CookieRun

Jangwon Moon
20171105
ntw0926@unist.ac.kr

Hyeonseo Kim
20191084
hyeonseo@unist.ac.kr

Yonghyeon Kim
20201064
yonghyeon@unist.ac.kr

Abstract

Digital gaming and artificial intelligence (AI) have been driving each other's development for a long time. In particular, AI has had a major impact on the development of the gaming industry, with applications in areas such as non-playable characters (NPCs) and game QA (beta testing). As an extension of this, this final project aims to train an AI player of the online game "CookieRun" to perform similarly to human players by using reinforcement learning. The AI is designed to be trained on different stages of the game so that results can be compared during the QA phase. Our team organized an environment to train the AI and conducted actual training, but due to various difficulties, we were unable to train the AI to the level of a human player within the project period. However, we made progress in the training process, and we expect that the AI's learning progress can be further improved with the proper training period and environment.

1 Introduction

The significant relationship between digital game and artificial intelligence (AI) has been a catalyst of innovation for each other. Notably, AI has left an indelible mark on the gaming industry, influencing the creation of non-playable characters (NPCs) and contributing to the rigorous testing processes, such as game quality assurance (QA)(*i.e.* beta test).Our final project embarks on the ambitious goal of training an AI player for the online game "CookieRun" using reinforcement learning.

The motivation behind this project lies in the aspiration to equip the AI player with the capability to emulate the performance of human players. The intention is to train the AI in various stages of the game, allowing for comprehensive comparisons during the QA phase. Our team designed an environment tailored for training the AI player and undertook the actual training process. Regrettably, we encountered a spectrum of challenges that impeded our progress, preventing us from attaining parity with human players within the designated project period.

Despite these challenges, our endeavor yielded valuable insights and progress in the training process. This report encapsulates our journey, highlighting the difficulties faced, the advancements made, and the potential avenues for future refinement. While the AI's performance fell short of the targeting point, we remain optimistic that, given an extended training period and an optimized environment, the AI's learning trajectory can be further elevated. This report aims to explain our approach, the hurdles encountered and the lessons learned in the pursuit of enhancing the AI player in "CookieRun" through reinforcement learning.

Figure 1: CookieRun Play Screenshot. There are various obstacles that a player should avoid by 3 kinds of actions.

2 Method

1. **DQN (Deep Q Network)**
   DQN is an algorithm that combines deep learning and reinforcement learning to achieve

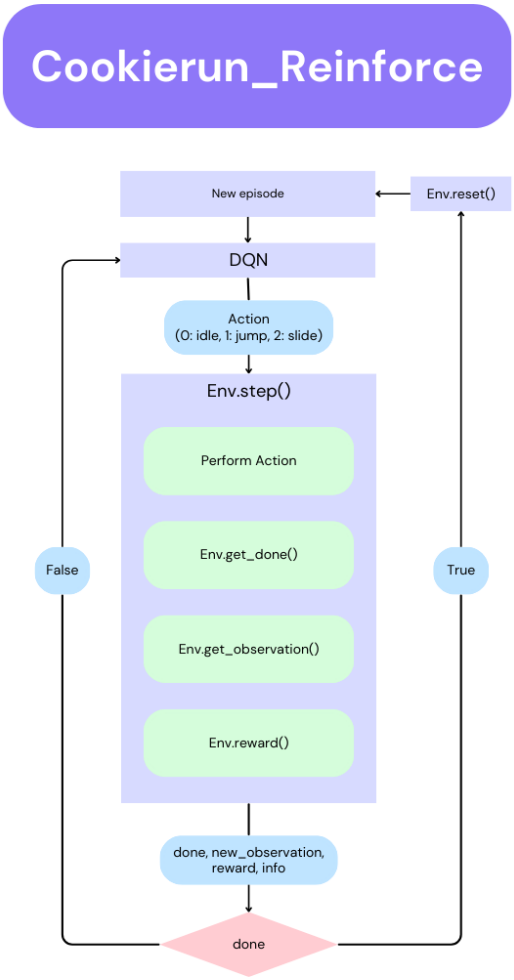

Figure 2: Flow Chart of the Gymnasium

high performance. Q-learning, one of the reinforcement learning techniques that learns without a conventional model, is learned by storing the state (s) of the environment and the action (a) of the agent as a Q-table (Q(s, a)). This has the disadvantage that when the state space and action space become large, a lot of memory and exploration time are required. DQN is a deep learning solution to this problem. DQN approximates the Q-function corresponding to the Q-table of Q-learning using a deep neural network. This enables reinforcement learning with high accuracy.

2. **Gymnasium - Env**
   The gymnasium environment consist of following functions. Step function will get input to make cookie do one of the three action: jumping, sliding, or do nothing. Also step will call following functions.

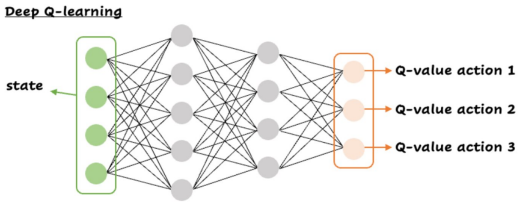

Figure 3: DQN Workflow

Get_observation scans the screen of the current frame. Get_done function tries to check whether game is over or not. The reset function is triggered when get_done function returns true. Then reset function will reset the game and start a new one. Other than that, we got some in functions that will help calculating rewards.

3. **Interaction of DQN and Env**
   The value returned by Env.step(), new_observation, serves as the input state for the DQN. Based on this new_observation, DQN runs through a deep neural network and returns one of three actions. Depending on the value of this action, Env.step() manipulates the game.

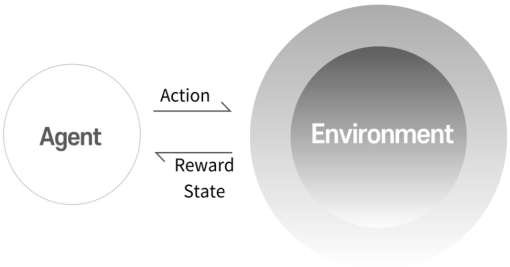

Figure 4: Interaction of DQN and Env

3 Implementation

1. **Env.reward()**

   1.1. Health bar recognition
   The health bar can be tested when we change the health bar into image. When setting into BGR image, we can check the image's R value; which is high when the color is red and low when the color is gray. Since the remaining health is red color and health used is gray, we can use this information to convert health into numbers. When the cookie is hit by the object, it will lose a chunk of the health

so we can calculate if they got hit or not by comparing previous health by current health.

1.2. Score recognition

The game score is located at the top center of the game screen. The goal of the Cookie Run is not just to survive as long as possible, but to get a high game score by eating objects. We needed to get this score as an int value. To do this, we utilized pytesseract, a library that extracts text from images.

However, we couldn't reliably get the int value in every frame due to the various borders of the game background. To solve this problem, we preprocessed the image containing the game score by extracting the RGB values (0, 0, 0). By performing pytesseract after preprocessing, we were able to reliably obtain the game score.

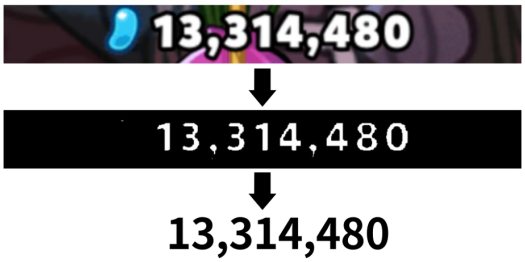

Figure 5: Score recognition process

1.3. Merge health bar and score

We added both health reward and score reward into one. When we hit by obstacle, we will set to health reward to -30. After hit by the object, the cookie will have some invincible time that is not affected by any objects; so we set next five frames to 0. Otherwise, the cookie is not damaged, so we reward them with 10 points. Meanwhile, the score is way higher than the score reward, we will divide by 100,000 for the score factor since health score will be more important.

2. **Env.get_done()**

The following screen is when the game is done. We can see the green rectangle here, which will pop up in the same location wherever you died. So we will use the image comparison by comparing histograms between two image. If histograms are similar, it means images are similar, and Opencv2 compareHist function can compare two histograms into from 0 to 1 where closer to 0 meaning images are not similar and closer to 1 meaning images are similar.

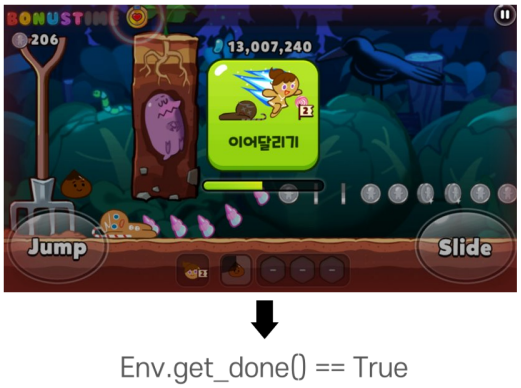

Figure 6: Game done screen

3. **Env.reset()**

The reset function is consist of several clicking popup screen in sequence. It's not perfectly handle all the situation but it will handle the most of the situation when popups appears.

4. **Env.get_observation()**

get_observation not only scans the game screen, but also preprocesses the scanned screen to provide data(new_observation). The purpose of this preprocessing is :

1. reduce the size of the training data
2. make learning more efficient.

The shape of the data we get when we scan the game is (3 * 800 * 1500). Each pixel of (800 * 1500) has an RGB value. We basically reshape this (800 * 1500) image shape to (90 * 120). We then tried different methods to find the most suitable preprocessing.

4.1. Gray Scale

We changed the RGB scale of the image to gray scale. The speed of 1 iteration was increased by reducing the amount of data. However, since we trained with a single value of gray scale, it was difficult to distinguish between obstacles and reward objects.

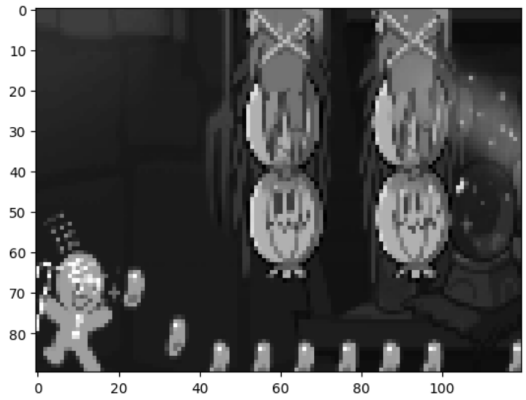

Figure 7: Gray scale data (1 * 90 * 120)

4.2. Canny Algorithm
Extracted the boundaries from the image using the canny algorithm. The purpose was to increase the recognition rate of objects with clear boundaries while using gray scale. However, the disadvantage is that the boundaries are not consistently extracted, and it becomes difficult to identify the boundaries when objects overlap.

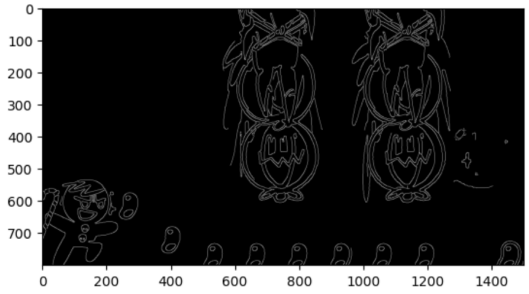

Figure 8: Canny algorithm data (1 * 800 * 1500)

4.3. RGB Scale
We used an RGB scale image resized to 90 * 120 as data.

4.4. K-mean
We applied k-means to improve the recognition rate of objects in the existing RGB scale. However, the final mean varies from frame to frame, so the input data was not uniform.

4.5. K-mean (mean fixed)
We provided the initial mean to address the non-uniformity in the k-means. By not taking the step of changing the mean, we were able to get uniform results across all frames. However, this had the disadvantage of increasing the time

Figure 9: RGB scale data (3 * 90 * 120)

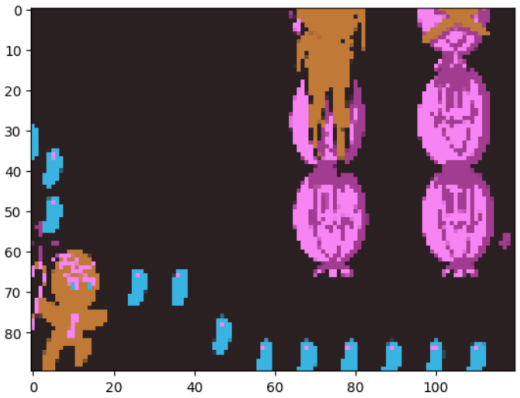

Figure 10: K-mean data (3 * 90 * 120)

it took to get the data.

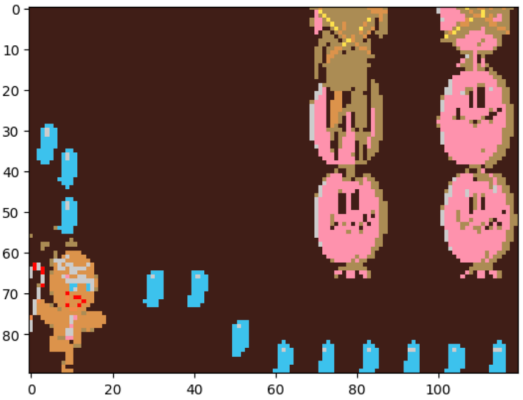

Figure 11: Fixed K-mean data (3 * 90 * 120)

These methods were evaluated based on the efficiency of learning by data generation time and object recognition rate. As a result, we used RGB scale for training.

4 Result

1. **Experimental Environments (day of week, timeout, popup)**

We had some limitations when it comes to training the DQN model. First Cookierun is online game still servicing now, we can't play the game infinitely. Only on Tuesday, Thursday and Sunday allows us to play infinitely, so it means only these three days are available for training the model. //

Figure 12: Limitation of play date, We could only test only on Tuesday, Thursday and Sunday

Also since random popups appears time to time, while we didn't fully implemented all the possible cases in the reset function, there are some cases that we had to undo some training progress while they are stuck in the popup.

2. **Experimental results**
   Three graph shows the performance of the stage 1(red) and stage 2(cyan); where stage 2 is the harder level than stage 1. The first graph shows the length of the episode which directly related to the survival time for each game. The next graph shows the total reward average of each timestamp. The third graph shows the loss while training, where loss here means the difference between predicted reward value and actual reward value.

3. **Analyze graphs**
   The first two graph shows that, at timestamp 80000, we can see performance of stage 1 start to increase while performance of stage 2 still struggles. It can meaning that DQN struggles more in stage 2 than stage 1; implying stage 2 is harder than stage 1. Although, we have some issues in stage 2 in the conclusion, we could spot the difficulty of the level can be affect the performance of DQN. Meanwhile the loss of the Stage 1 dramatically moves a lot

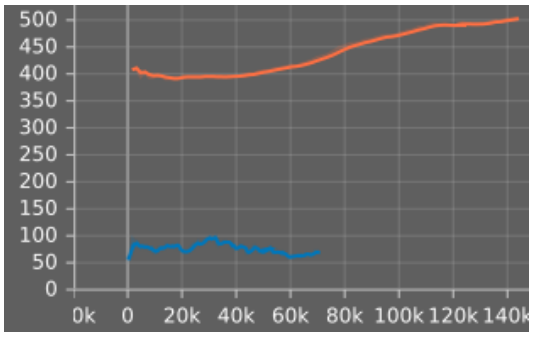

Figure 13: Episode Length mean

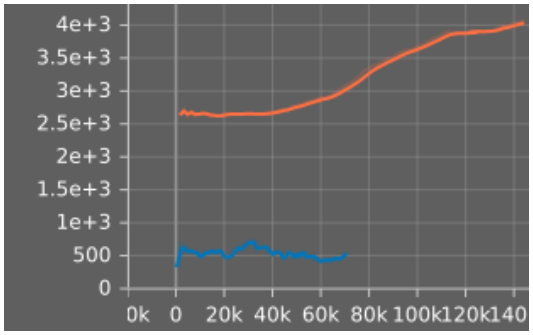

Figure 14: Reward Average mean

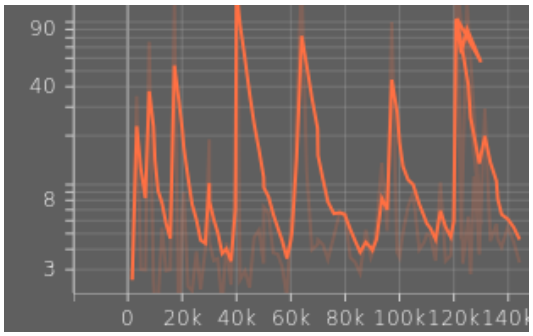

Figure 15: Loss of Stage 1

time to time. We suspect that this happens because of the bonus time; when cookie collects all the alphabet jellys. Sometimes cookie can go to bonus time and sometimes they can't, but it will result in several seconds of game difference and it will grant high score when cookie goes to the bonus time.

5 Conclusion

1. **Implication**
   In this study, we wanted to show the potential of using reinforcement learning to QA games. We expected that AI could be used to replace a lot of human QA effort, and that it could find ways to play that humans couldn't think. You can also compare the difficulty of the stages numerically by comparing the time it takes

the reinforcement learning AI to master each stage.

In fact, the reinforcement learning we ran on Cookie Run yielded the results we expected. The reinforcement learning AI played differently from humans. Also, as you can see in the results section, we were able to compare the difference in difficulty between stage1 and stage2 with the difference in the degree of reinforcement learning.

2. **Recommendation**

   2.1. Error of score function
   The stage 2 had white background, so it sometimes failed to recognize the image into numbers. That's why stage 2 got high loss and have failed to train properly for each frames. More advanced score recognition methods will be needed to eliminate these errors.

   2.2. Popup window issue
   Cookie Run has various pop-ups in between games. These notify you of achievements, events, and the time in the game. These pop-ups continued to appear during reinforcement learning. Some of the pop-ups were removed by recognizing image similarities, but there were so many that it was not possible to remove them all. It would be more efficient to remove the dummy data from the reinforcement learning caused by these pop-ups.

   2.3. Time limitation
   Due to Cookie Run's policy, there are only certain days of the week that reinforcement learning can take place. This limited the amount of time we could spend on reinforcement learning, and more time would have yielded better results.

   2.4. Other variable for reward or observation
   There may be other variables to consider when making an observation or earning a reward.

Github Link

6 Acknowledgement

This report is final project report of UNIST CSE362 Artificial Intelligence course in 2023 fall semester.