Game Agent Report

KyooSik Lee
2016147526

May 17, 2018

1 Introduction

This report is Yonsei CSI4108 Artificial Intelligence course's HW2 report.

The subject of this report is to discuss the metheology I have implemented to get higher score in HW2(game_agent) in python.

2 Defining States

The number of states in this game is very important because it is related to the number of training. If there are too many states(for example, just concatenating all items and basket location), the possibility of one state appearing multiple times in episodes will decrease, which will require more number of training and more time.

So I decided to define states as whether each item is outside of the basket range or inside of the basket range.
The rules of defining states is as follows.

1 If the item was in the basket range, label value 1 to the item.

2 If the item was in the left side of the basket range, label value 2 to the item.

3 If the item was in the right side of the basket range, label value 3 to the item.

4 For each item value, concatenate them to make state value.

Following is an example of how I managed to give state value.

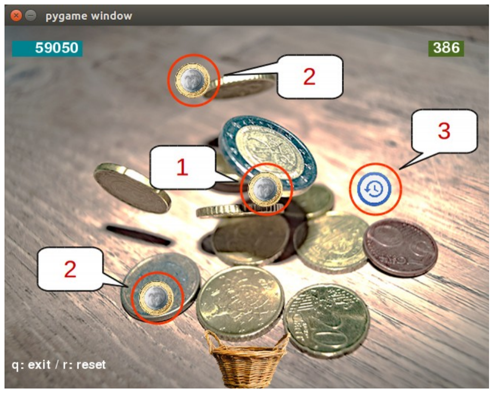

Figure 1: State Define

In Figure 1, there are 4 items on the screen. The lowest item is on the left side of the basket range, so the state value 1 is given. For two items on the same

height, the counter item is on the right side of the basket range, so value 3 is given. for the coin item, value 1 is given. And lastly, for the highest coin item, it is in the left side of the basket range so value 2 is given.
But to get higher score, it is important to have plenty of time counter. Rule 1 to 4 did not consider which item the basket gains. Therefore, the game could end unintentionally early. So I added more rules considering the remaining time counter.

5 If time item does not exist on the screen, follow the rule 1 to 4

6 If time item does exist on the screen,and the counter is under 600, then ignore other item and only define the state as whethre the time item's position is on which side of basket range.

Finally, the state is defined.
After defining states, I trained the game agent and found out that low score happens occasionally. Then I let my agent play the game and I noticed that for certain basket location, the agent couldn't catch coin or clock item. After looking up the value, I could make the following table which shows for what value of basket_location can the agent eat the item.

Table 1: Catch-able location for basket location

basket location	0	1	2	3	4	5	6	7	8
catch location	0,1,2	1,2,3	2,3	4,5,6	5,6	6,7,8	8,9,10	9,10,11	10,11

After making this table, I modified the code so that the game agent can determine if the game agent could catch the item. After the modification, the game agent worked well.

3 Defining Rewards

Defining rewards is important because the rewards will decide which movement the agent decides to go. The default rewards, which is the sum of difference of time counter and difference of score, was very straight-forward. So I have made small changes. Having basket at the end of each side can lead the agent to miss items coming from the other end, therefore I gave a small minus rewards when the basket was at the position of 0 or 8.

Also, because the game depends most on the time counter, I weighted on the difference of the time counter by multiplying 10. By doing this, the game agent will gain more rewards when the game agent catches time counter

4 Policy Function / Q-Score

I did not change any policy function or Q-score. I only changed the `get_-state()` function and `get_reward()` function The default policy function was epsilon-greedy policy. This means that at a epsilon probability, the game agent will take a random action. The `make_policy(Q, epsilon, nA)` function works by defining another function inside the `make_policy()` function. the inside function is `policy_fn(observation)` function which returns a matrix of giving the best_action 1-(2*epsilon) and other two actions epsilon. the `make_policy()` function returns `policy_fn()` function.

5 Learning Parameters

Following is what I wrote commands for training on the terminal.

```
python q_learning.py -n 30000 -e 0.001 -lr 0.01
```

Epsilon is the possibility of random action regardless of the given command.

6 Conclusion

The game agent I designed mainly focused on catching time counter. By doing this, the game agent could get higher score greater than 300000 because there were plenty of time counter left after reaching 300000.
Advantage of time-oriented game style is that the game agent can get higher score above 300000.
Disadvantage of time-oriented game style is that the game agent takes too much time to end one game.

6.1 Result

Quantitative results

The average of 14 score is 346003
My nickname for Ranking Board is BaekSanSu.
The result of one game can be checked on the following link.
https://youtu.be/i6cnWu8JcsU

6.2 Analysis

Analysis for successful case / failure case

Almost every trial using command `python game.py -s 0 -p 0` had score higher then 300000. But low score occurred few times. The low score is probably the occation when the time counter's location was unlucky to the game agent.