2021 Artificial Intelligence Final Project Group 15

Diary of a Wimpy Snake

-The Snake of Wrath-

0716325 曾正豪 (Zheng-Hao Tzeng), 0716328 葉眉湘 (Mei-Xiang Yeh)

College：National Chiao-Tung University, ROC

I. GitHub Link

https://github.com/banne2266/AI-FinalProject

II. Introduction

"Snake Game", which shot to frame by the prevalence of Nokia mobile phones, is a classical deterministic game. In the game, the player's goal is to maximize the score by eating the random generalized food and avoiding collision is the key to its survival.

In this game, collisions are divided into two types — hit itself or hit the boundary — once the collision occurred, the game is over. Hence, our aim is to develop an ultimate autonomous snake which accumulate more food as possible as it can based on the concept of reinforcement learning (RL) we have learned in class.

We choose Deep Q-learning which is off-policy as the main reinforcement learning algorithm with neural network to implement the game training. And we'll make a comparison between different concerning states.

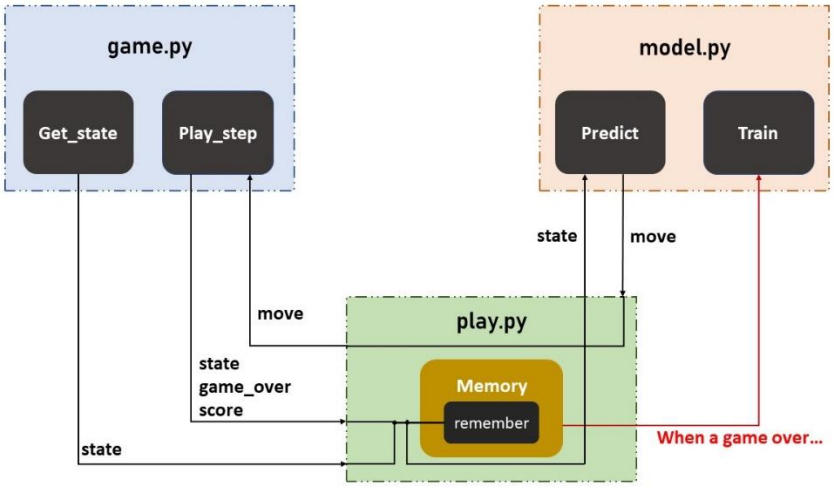

Fig.1：shows our overall idea on this project

III. Related work

There are plenty of works about the reinforcement learning experiments for Snake Game. Our work is referred to [1]. The reason we choose [1] as the baseline of our project is that its author's description is relatively clear than others and the main algorithm has a greater correlation with what we have learned in class. However, his work is not good enough, and we trust we can improve its performance.

In [1], the author used basic reinforcement learning and deep Q learning built in Pygame and Pytorch. The Fig. 2 is his architecture. He built three main classes in his project. He divided the movement into 3 directions showed in Fig. 3. Then, he planned the states into 11 categories showed in the Fig. 4. Danger means the next step is itself or boundary. Direction left/right/up/down means the direction it goes in one-hot encoding. Food left/right/up/down means the relative position to the snake's head. And the Fig. 5 is the reward he set in his model.

Actions

[1, 0, 0] -> straight
[0, 1, 0] -> right turn
[0, 0, 1] -> left turn

Fig.3

States

[danger straight, danger right, danger left,

direction left, direction right,
direction up, direction down,

food left, food right,
food up, food down]

Fig.4

Reward

Eat food ：+10
Game over： -10
Else ：0

Fig.5

The previous work only considered relative position of food and the snake's head, but in our work, we took more detail about the game into account to increase the average performance of the model (the detailed adjustment will be explained in IV. part).

IV. Methodology

A. Q Learning – Deep Q learning

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. The agent will try to adopt the optimal policy from the history of interaction with the environment. Below is the equation captured in the course material.

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\hat{Q}_{\text{opt}}(s,a) + \eta(r + \gamma \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s',a'))]$$

In the above equation, s, s' are the current and next state, r is the reward given by taken action a in state s, γ is the discount rate set by us. η is the learning rate.

In Q-learning, we need to maintain a Q-table, its size is number of possible states * number of actions. While the size of the Q-table is a vast number, which is quite a burden for memory. Hence, we need a method that have less memory usage. That is, neural network.

B. Neural Networks

We input a state into a neural network and it will return the Q-value of each actions. It has two hidden layers. We use ReLU as the activation function of hidden layers and its Loss function is mean-square error.

C. Training Strategy

First, we have a memory to save the state, action, reward and next state in each round. When the game begins, in each step, we will send the current state to the neural network and get the action with maximum Q-value or select a random action with probability ε. Then implement the action to get the reward and next state and save state, action, reward and next state into memory again. It repeats until the round is over.

After a round is ended, for each element in the memory, calculate the target Q-value with $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$ and perform the gradient descent on $(y_j - model(state_j))^2$ with respect to the neural network.

D. How Agent Define a State

In Q-learning we need to define the game states and it is different from [1]. Fig. 6 lists all the game states we have defined.

```
State =
[ Snake's head hit its body on left next step,
  Snake's head hit its body on right next step,
  Snake's head hit its body on straight next step,

  Snake's head hit boundary on left next step,
  Snake's head hit boundary on right next step,
  Snake's head hit boundary on straight next step,

  one hot encoding of direction (right/left/up/down),

  food on the left of snake's head,
  food on the right of snake's head,
  food on the up of snake's head,
  food on the down of snake's head,

  food on the eight direction of snake's head
  any body part on the eight direction of snake's head
]
```

Fig. 6：States defined on our own.

E. Binary Vision

In the last two state we defined, there are two modes to record the state. One is distance vision, and the other one is binary vision. In distance vision, the snake could directly know the practical distance to the food or to its body part on that direction. But it has a disadvantage when we train a good model. If we want to change the map size, we need to train the model again. While the binary vision doesn't need to do like this. Since it only checks if there is any target on the direction and present it as 1 or 0. What's more, the binary vision costs less time to train.

V. Experiments

Version 0.

In the version 0. We take the distance between eight directions of snake's head and the four boundaries into account.

This version is totally bullshit. Since after 5000 generations, it keeps going round and round towards the center of the map. And its average score is less than 1. Then we devise next version.

```
State =
[
  one hot encoding of direction (right/left/up/down),

  food on the eight direction of snake's head
  any body part on the eight direction of snake's head

  distance between eight directions of snake's head and boundary
]
```

Fig. 7：States defined in version 0.

Version 1.0

This version finally works. The average score comes to about 40 in a 20x20 map size. But the snake can't distinguish the body and boundary when a danger is in the next step. The major problem in this version is that the snake often ends the game in the condition showed in the Fig. 9.

```
State =
[ Snake's head is dangerous on left next step,
  Snake's head is dangerous on right next step,
  Snake's head is dangerous on straight next step,

  one hot encoding of direction (right/left/up/down),

  food on the left of snake's head,
  food on the right of snake's head,
  food on the up of snake's head,
  food on the down of snake's head,

  food on the eight direction of snake's head
  any body part on the eight direction of snake's head
]
```

Fig. 8：States defined in version 0.

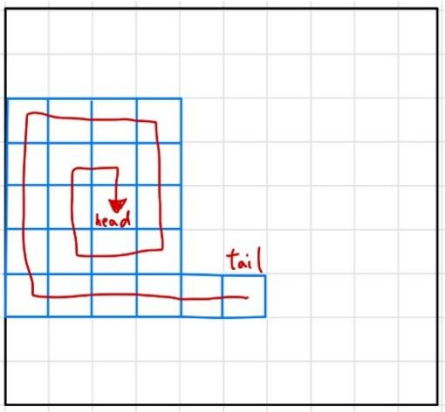

Fig. 9：One of the game over cases.

Version 2.0

Compared to version 1, the agent starts to distinguish the danger in the next step is body or the boundary. The snake starts to approach to the food like the way in Fig.11 which is more cleverer than before and is more human-intuitive. But sometimes, it still encounters the cases in Fig. 9. The average score is about 80 in a 20x20 map size.

State =
[Snake's head hit its body on left next step,
Snake's head hit its body on right next step,
Snake's head hit its body on straight next step,

Snake's head hit boundary on left next step,
Snake's head hit boundary on right next step,
Snake's head hit boundary on straight next step,

one hot encoding of direction (right/left/up/down),

food on the left of snake's head,
food on the right of snake's head,
food on the up of snake's head,
food on the down of snake's head,

food on the eight direction of snake's head
any body part on the eight direction of snake's head
]

Fig. 10：States defined in version 2.0.

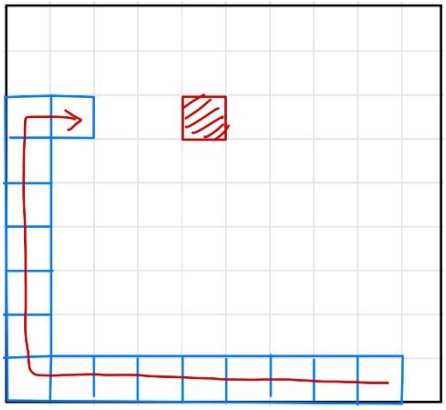

Fig. 11：One of the optimal solution to approach to the food.

Version 2.01

To solve the problem in Fig.9. We decide to double the considered area of the first 6 states in Fig.6. The snake's behavior is like Fig. 12. It learns how to get rid of the situation in Fig.9. The average score is increase to 100 in 20x20 map size. However, we think it is not good enough so we produce version 2.02.

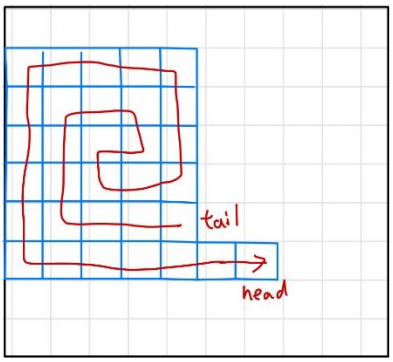

Fig. 12：A way to avoid self-collision.

Version 2.02

In the previous versions, we use the neural network with 2 hidden layer of size 256. We want to make it less computational, so we decrease the hidden layer size to 32. Something incredible happened, it performed much better than all previous versions. The average score is about 140 in 20x20 map size.

What's more, it starts to follow the optimal solution in Snake Game. That is, U-shaped strategy (Fig.13). But it still can't go in the S-shaped to eat all food and fill up all the box.

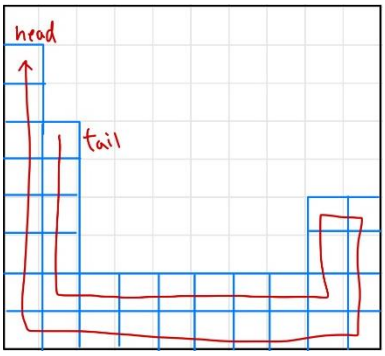

Fig. 13：U-shape is a strong strategy in Snake Game.

How we Modify Reward

In the beginning of training model, we use the reward in Fig. 5. However, after hundreds of generations, the average score usually stops increasing. And we found that the snake is so eager that it eat the food without noticing that it may collide its body. Therefore, we modify the reward to Fig.14 and we found that it starts to increase the average score after some generations.

Reward
Eat food： +10
Game over： -1000
Else：0

Fig. 14：the reward with better perormance.

VI. Conclusion

We construct a good AI in Snake Game by using Deep Q-learning. Since we totally use binary vision as the format of game state. Even we train this model in a 20x20 map size, our model could directly apply on any other map size without training again. It requires a lot patience to complete this project because it costs a lot training time.

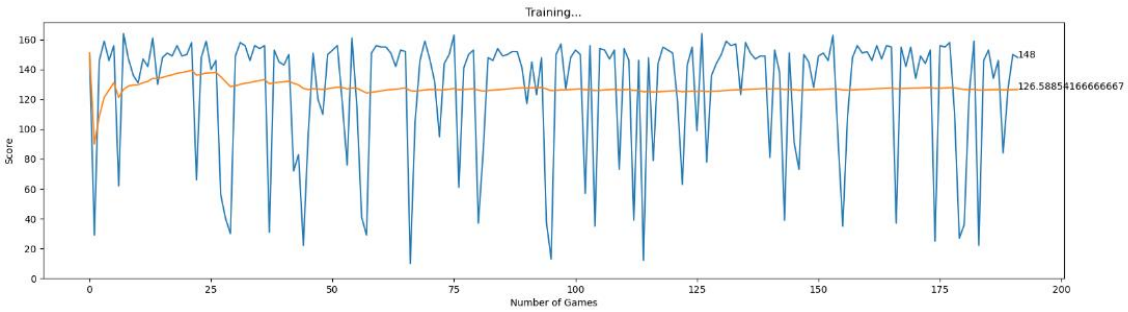

Fig.15: Blue line is the score the snake get in each round.
Orange line is the average score.

VII. References

[1] https://github.com/python-engineer/snake-ai-pytorch

[2] https://towardsdatascience.com/snake-played-by-a-deep-reinforcement-learning-agent-53f2c4331d36

[3] http://cs229.stanford.edu/proj2016spr/report/060.pdf

[4] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8460004

[5] https://www.playpcesor.com/2015/05/18-nokia-snake-app.html

[6] https://www.wpgdadatong.com/tw/blog/detail?BID=B2342

[7] http://spranesh.github.io/rl-snake/