Theory of Computer Games 2022

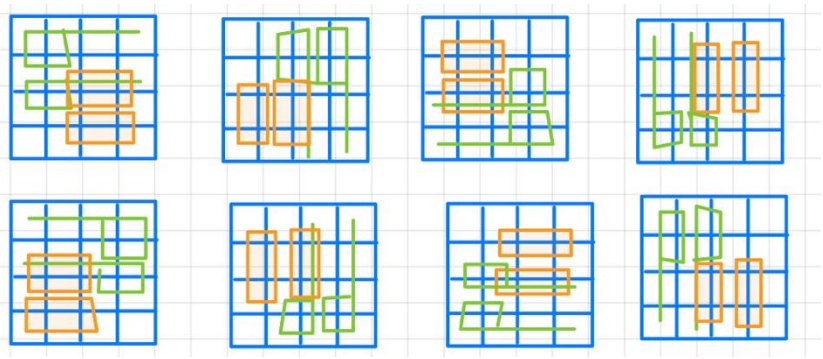
Report of Project #2

Student name: 劉子齊 Jonathan
Student ID: 311605004

In project 2, our goal is to implement a n-tuple network and the TD(0) after state learning network to improve the simple heuristics-based player implemented in the previous project. For the framework of the game, and the environment, I followed the given sample. The original given method was to play randomly. In the previous project, I made my agent to make an effort keeping the bricks in the left-bottom corner of the board, also, I took the rewards into consideration to make improvements in the previous project.

In the beginning of this project, I started with implementing an 8 * 4 tuple network with backward methods. After training its weights for 1,000, 000 times, I found that even if I make adjustments on the parameters, the performance was still not so good, which was with the score around 75. Hence, I start to implement an 8 * 6 tuple network with backward method by extending the 8 * 4 tuple network I implemented.

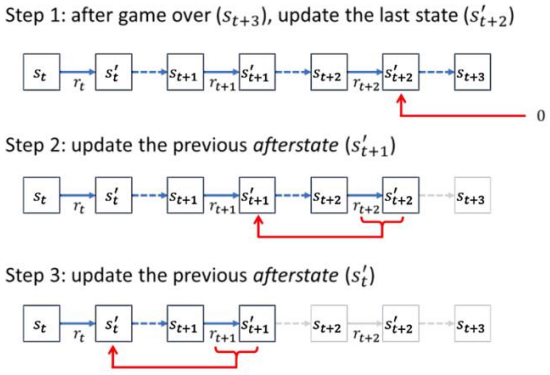
In a Threes! Game, there are 16 positions, each position has 16 possible states. If took the whole board into consideration, there will be 16^16 different situations, which is extremely resource consuming. Hence, we do not directly find out the score of the current board in the table, but find out the score of each feature, and then add up these features to represent the score of the entire board. For this project, I defined the features as show in the figure below.



With the features defined, now we figure out how to get the score of the corresponding feature. First, I took the value of the corresponding position according to the index. For example, the value of position 0 is 1, the value of position 4 is 0, the value of position 8 is 1, the value of position 9 is 8.... In this way, we can obtain this feature Corresponds to {1, 0, 1, 8, 2, 3}. In fact, the feature has been extracted in this step, but the efficiency of finding features in this way will be relatively low. The best way is of course to put {1, 0, 1, 8, 2, 3} is encoded as a number, and the feature is stored in an array, so that the search efficiency is the most efficient by its key and value. Hence, we encode the features into a number by the function shown below.

```
// 8 x 6 Tuple
int get_feature(const board& after, int vertice1, int vertice2, int vertice3, int vertice4, int vertice5, int vertice6) const{
    return after(vertice1) * 16 * 16 * 16 * 16 * 16 + after(vertice2) * 16 * 16 * 16 * 16 + after(vertice3) * 16 * 16 * 16 + after(vertice4) * 16 * 16 + after(vertice5) * 16 + after(vertice6);
}
```

With the 6 * 8 tuple network set, we move onto the TD learning. With the TD learning, we will be able to perform the seek and update trick, which we update the after states from the end to the beginning according to the result of ach states. The updating flow of the TD learning is basically the same as the flow shown in the figure below.



With the implementation all set, I start with testing different learning rates and training rounds. As a result, I obtained the best performance with the learning rate of 0.0025 and 1,000,000 game rounds, as shown in the following figure, which has the performance score of 95.8 in the Linux station provided by this course.

```
[[c311605004@tcglinux6 Project2]$ ./threes-judge --load 8x6tuple.txt --judge version=2
Threes! Judge: ./threes-judge --load 8x6tuple.txt --judge version=2

1000    avg = 130491, max = 264702, ops = 927667 (546870|3706012)
        48      100%    (0.1%)
        96      99.9%   (0.4%)
        192     99.5%   (1.6%)
        384     97.9%   (4.8%)
        768     93.1%   (7.5%)
        1536    85.6%   (39.9%)
        3072    45.7%   (45.7%)

Judging the actions... Passed
Judging the speed... Passed, expected 39598 ops
Assessment: 95.8 points
```