AI Capstone assignment2 report

109550167 陳唯文

Environment

In this assignment, I use python as my programming language and use pyinstaller to turn python file into exe file.

Algorithm

In this assignment I choose minimax to be the algorithm of my game agent.

```
def end(cur_map):
    return not (cur_map==0).any()
```

The function end is used to decide if the game is over or not.

```
def checkRemainMove(mapStat):
    free_region = (mapStat == 0)
    temp = []
    for i in range(len(free_region)):
        for j in range(len(free_region[0])):
            if(free_region[i][j] == True):
                temp.append([i,j])
    return temp
```

The function checkRemainMove will return box which is free to go in the current map stat as a list.

```
def next_map(player, cur_map, move):
    [move_pos_x, move_pos_y] = move[0]  # expected [x,y]
    steps = move[1]  # how many step
    move_dir = move[2]  # 1~6

    next_x = move_pos_x
    next_y = move_pos_y
    cur_map[next_x][next_y] = player
    for i in range(steps - 1):
        [next_x, next_y]=Next_Node(next_x,next_y,move_dir)
        cur_map[next_x][next_y] = player

    return cur_map
```

Next_map will return map stat after apply any movement

```
def legalaction(cur_map):
    free = checkRemainMove(cur_map)
    actions = []
    visit = np.zeros([144, 144])
    for i, j in free:
        if cur_map[i][j] != 0:
            continue
        else:
            actions.append([(i,j), 1, 1])
            legal = []
            for dir in range(1,7):
                [next_x, next_y] = Next_Node(i, j, dir)
                if next_x < 0 or next_x > 11 or next_y < 0 or next_y > 11 or cur_map[next_x][next_y]!=0:
                    continue
                else:
                    if(visit[i+j*12][next_x+next_y*12] == 1):
                        continue
                    actions.append([(i, j), 2, dir])
                    visit[i+j*12][next_x+next_y*12] = visit[next_x+next_y*12][i+j*12] = 1
                    [next_x, next_y] = Next_Node(next_x, next_y, dir)
                    if next_x < 0 or next_x > 11 or next_y < 0 or next_y > 11 or cur_map[next_x][next_y]!=0:
                        continue
                    else:
                        actions.append([(i, j), 3, dir])
    # if(len(actions) > 15):
    #     actions = random.sample(actions, 1)
    return actions
```

This function legalaction will generates a list of possible actions that a player can take in the game based on the current map state. First, I call checkRemainMove to get the list of remaining positions that are free. Next, check every direction of those boxes to see the possible ones. If any direction is available, then it will check the box next to it along the same direction. If that box is still free, it means that the possible steps along the direction is 2 and 3. If not, it means that there are only 2 possible steps.
I use the 2d array visit to avoid adding actions which are actually same. For example, if there are two possible actions, [(1, 0), 2, 6] and [(1, 1), 2, 1], it will be regarded as same action and only one of them will be considered.

```
def minimax(player, depth, cur_map, alpha, beta):
    if end(cur_map):
        if player == 1:
            score = 100
        else:
            score = -100
        return [None, score]
```

Here is my implementation of my minimax. First, it'll check if the game is end, if the game has ended and the player now is 1(maximize player), it means that player 2 is the one that ended the game, so it'll return 100, if the situation is opposite, it'll return -100.

```
if depth == 0:
    actions = np.array(legalaction(cur_map), dtype=object)
    if np.all(actions[:,2] == 1):
        if(len(actions)%2 == 1):
            if player == 1:
                score = -99
            else:
                score = 99
        else:
            if player == 1:
                score =  99
            else:
                score = -99
    else:
        score = 0
    return [None, score]
```

When the depth is 0 now, it'll check the remaining positions. If it's player 1's turn , and all the remaining position is independent and the number of remaining positions is odd, we can inference that final loser will be player 1, so it'll return -99 as it's score. We can use this case to inference other situations. If the remaining positions doesn't fit any of the above four cases, the score will be 0, meaning the movement won't cause any advantages or disadvantages.

```
actions = legalaction(cur_map)
random.shuffle(actions)
scores = []
if len(actions) > 22:
    return [actions[0], 0]
```

Then it'll begin entering the part of maximizing player and minimizing player, but if the possible actions is too many, I think there's very few chances that we take any advantageous actions, also the calculation of minimax will be too big for our program to response in time. So, if the actions we can take now is over 22, it'll just return a random movement and the return score will be zero.

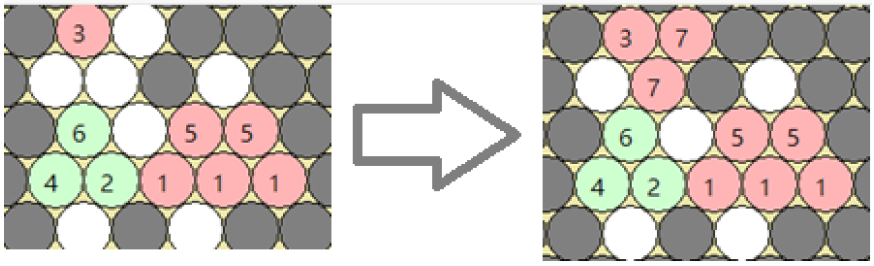
```
if player == 1:
    best_action = None
    best_score = float("-inf")
    for action in actions:
        new_map = copy.deepcopy(cur_map)
        new_map = next_map(1, new_map, action)
        _, score = minimax(2, depth-1, new_map, alpha, beta)
    #     scores.append((action, score))
    # best_action, best_score = max(scores, key=lambda item: item[1])
        if best_action is None:
            best_action = action
        if score > alpha:
            best_score = alpha = score
            best_action = action
        if beta <= alpha:
            break
    return [best_action, best_score]
```

I add alpha beta pruning in my minimax implementation so that the calculation amount can be smaller and to add more depth in minimax. After testing, the depth can change from 5 to 10.(depth – 1 every time the player changes)

Discussion

Winning percentage against sample

I played ten matches using my program and the sample program, and the result is 10W0L, showing that the strategy I take is useful. I also try matching my program with other program that uses minimax as its algorithm but with only two statuses(win and lose) in its evaluation function. The testing result is 5W0L, it proves that evaluation function that check the number and independence of remaining positions make a big difference to our game agent.



Parameters settlement

There are two parameters I try and change a lot to see how to have better impact to the game agent, which are the depth and the timing to start evaluating. I think the parameter now is enough to beat normal minimax game agent, but it could still cause timeout sometimes, so I'm not sure how often will the timeout happens in real situation, since it depends on the CPU of the computer.

Experience

This assignment is a good opportunity for us to practice our skill and enhance better understanding of reinforcement learning, but it's a shame that I only implement a basic algorithm but not more advanced algorithm such as MCTS or deep q learning, and as I progressed, I realized that minimax had its limitations, so I think the ranking won't be too good. Implementing more advanced algorithm requires deeper knowledge and machine learning concepts, I think it's a challenging task, so I will keep learning and improving my skill.