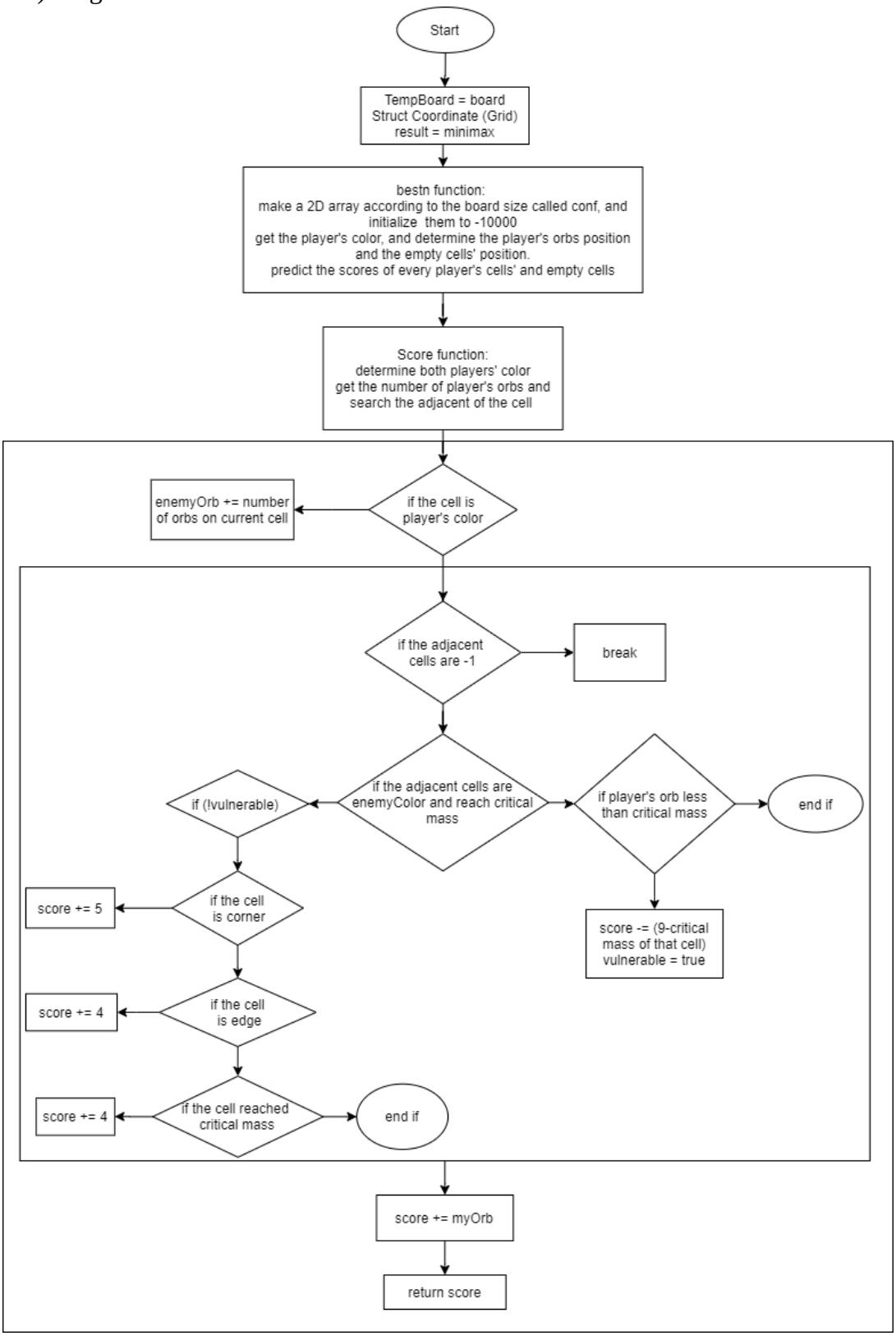
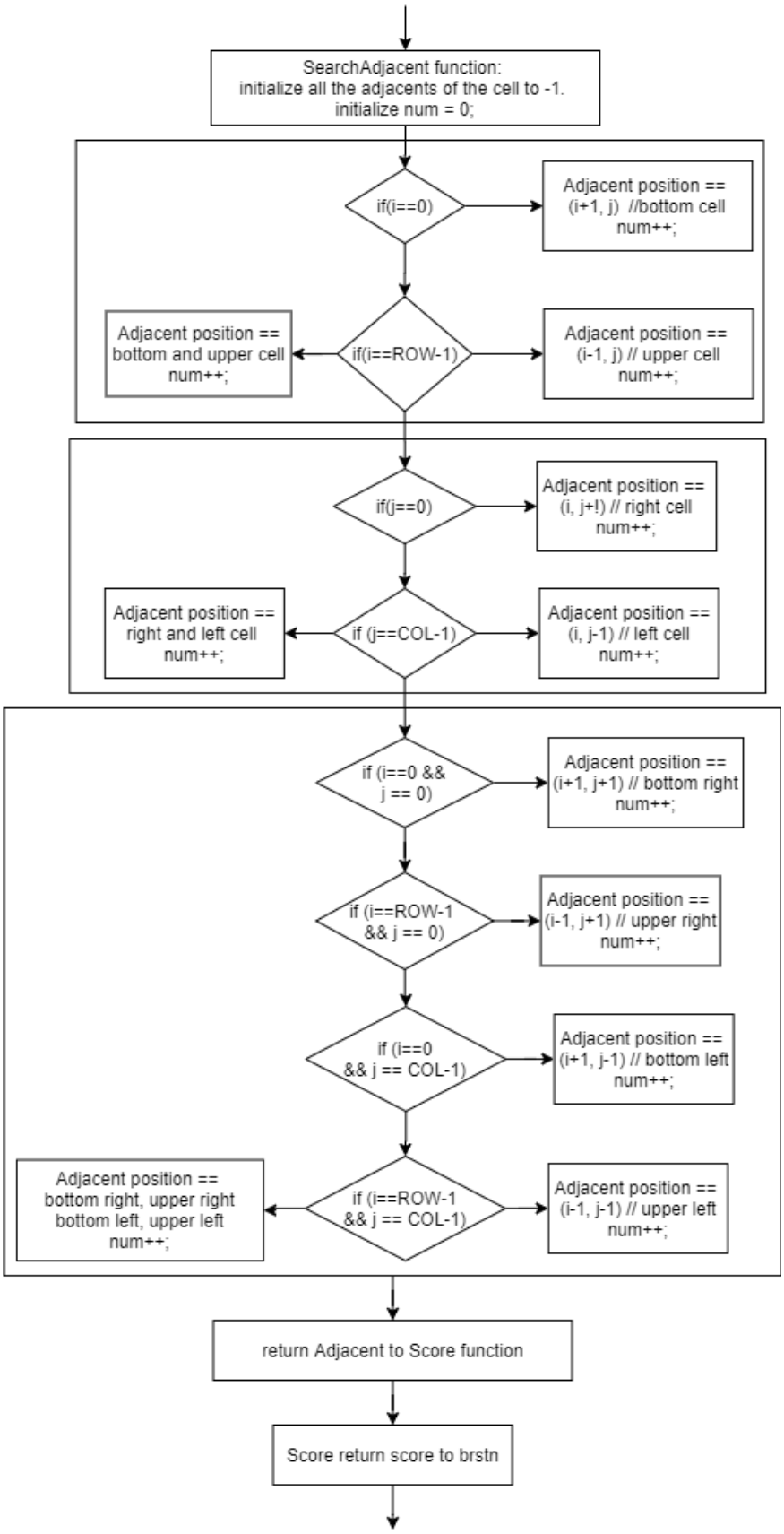
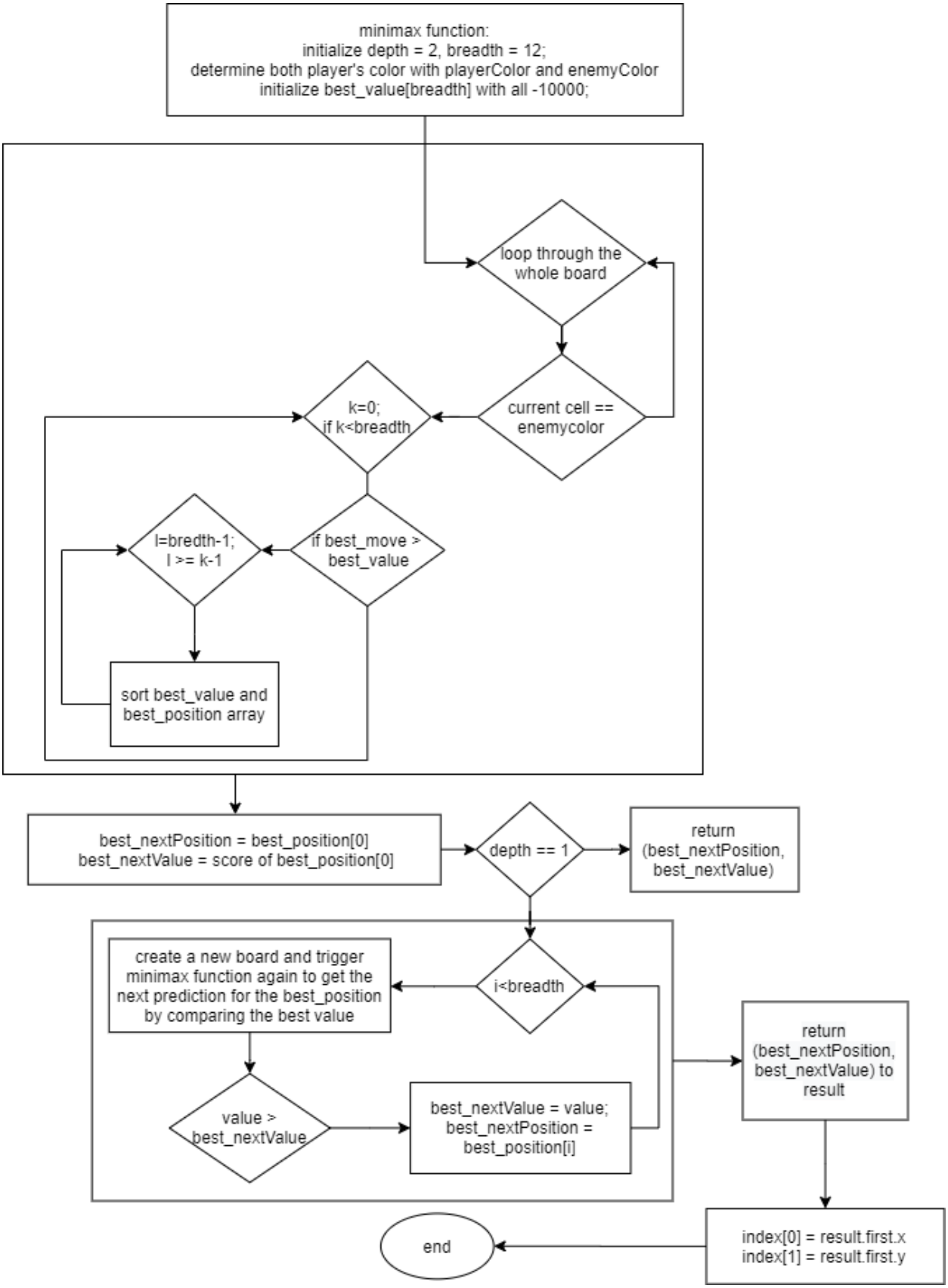
1-1) Program Flow Chart




SearchAdjacent function:
initialize all the adjacents of the cell to -1.
initialize num = 0;
if(i==0)
Adjacent position == (i+1, j) //bottom cell
num++;
if(i==ROW-1)
Adjacent position == bottom and upper cell
num++;
Adjacent position == (i-1, j) // upper cell
num++;
if(j==0)
Adjacent position == (i, j+!) // right cell
num++;
if (j==COL-1)
Adjacent position == right and left cell
num++;
Adjacent position == (i, j-1) // left cell
num++;
if (i==0 && j == 0)
Adjacent position == (i+1, j+1) // bottom right
num++;
if (i==ROW-1 && j == 0)
Adjacent position == (i-1, j+1) // upper right
num++;
if (i==0 && j == COL-1)
Adjacent position == (i+1, j-1) // bottom left
num++;
if (i==ROW-1 && j == COL-1)
Adjacent position == bottom right, upper right bottom left, upper left
num++;
Adjacent position == (i-1, j-1) // upper left
num++;
return Adjacent to Score function
Score return score to brstn


minimax function:
initialize depth = 2, breadth = 12;
determine both player's color with playerColor and enemyColor
initialize best_value[breadth] with all -10000;
loop through the whole board
current cell == enemycolor
k=0;
if k<breadth
if best_move > best_value
l=bredth-1;
l >= k-1
sort best_value and best_position array
best_nextPosition = best_position[0]
best_nextValue = score of best_position[0]
depth == 1
return (best_nextPosition, best_nextValue)
i<breadth
create a new board and trigger minimax function again to get the next prediction for the best_position by comparing the best value
value > best_nextValue
best_nextValue = value;
best_nextPosition = best_position[i]
return (best_nextPosition, best_nextValue) to result
index[0] = result.first.x
index[1] = result.first.y
end

1-2) Detailed Description

The algorithm designed is using minimax algorithm to maximize the winning chance for the game. The minimax algorithm will firstly determine the player's color and predict the score of every predicted move. Then use BFS to find the most higher winning chance 12 moves. Then sort the 12 moves according to the score ascendingly. Trigger minimax function again by the current best move position and score. If there are better next move, update it as the best position.

The score of the moves is defined by the position of the cells and the critical mass of the cell. If current cell's is surrounded by enemy orbs which reached critical mass, the score will be subtracted by minus 9 with the number of orbs of cell. If current cell is at the corner, the score will be added by 5. If current cell is at the edge, the score will be added by 4. If current cell reached critical mass, the score will be added by 4. If current cell has orbs, the score will be added by 1.

2-1) Partial Implemented Code

Below attached with the partial implemented code:

```
for(int i = 0; i < ROW; i++)
{
    for(int j = 0; j < COL; j++)
    {
        if(TempBoard.get_cell_color(i, j) == playerColor || TempBoard.get_cell_color(i, j) == 'w')
        {
            Grid position;
            position.setPosition(i, j);
            conf[i][j] = Score(move(TempBoard, position, player), player);
```

Figure 1: The code about the conditions to place the orbs

```
for (int i=0; i<ROW; i++)
{
    for(int j=0; j<COL; j++)
    {
        if(TempBoard.get_cell_color(i,j) == enemyColor) continue;
        for (int k=0; k< breadth; k++)
        {
            if(best_move[i][j] > best_value[k])
            {
                for(int l=breadth-1; l >= k+1; l--)
                {
                    best_value[l] = best_value[l-1];
                    best_position[l].setPosition(best_position[l-1].x, best_position[l-1].y);
                }
                best_value[k] = best_move[i][j];
                best_position[k].setPosition(i, j);
                break;
```

Figure 2: Compare and arrange the best moves after BFS

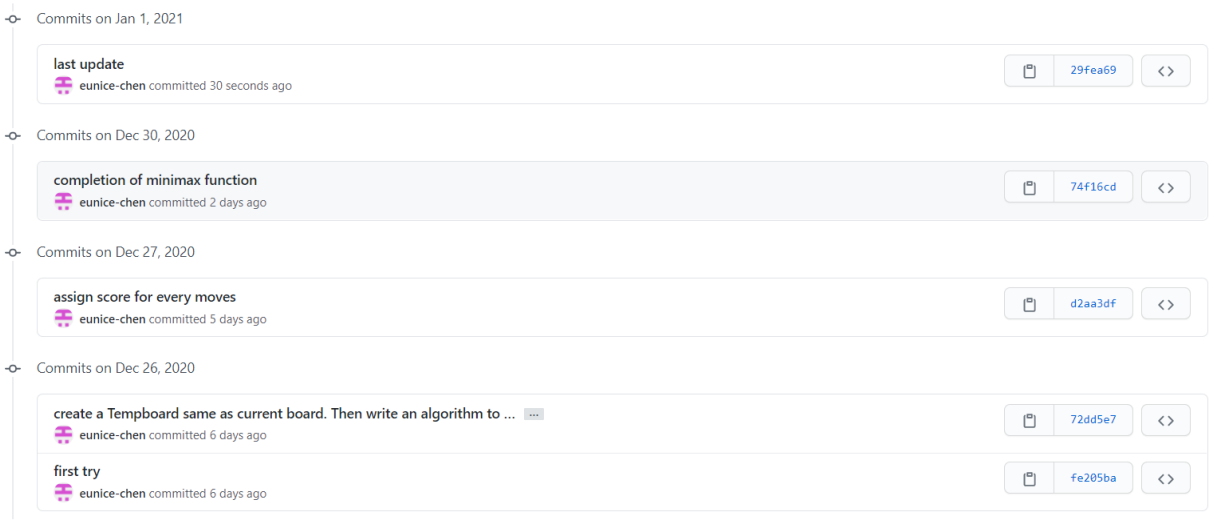
```
if(depth==1)
{
    return std::make_pair(best_nextPosition, best_nextValue);
}

for (int i=0; i<breadth; i++)
{
    Board newBoard(move(TempBoard, best_position[i], player));
    pair <Grid, int> best_move = minimax(newBoard, depth-1, breadth, player);
    int value = best_move.second;
    if(value > best_nextValue)
    {
        best_nextValue = value;
        best_nextPosition = best_position[i];
    }
}
```

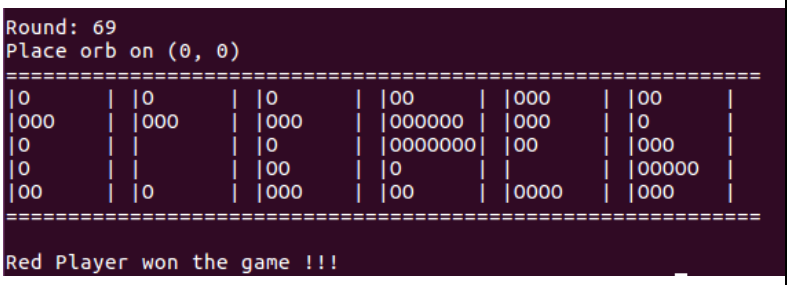
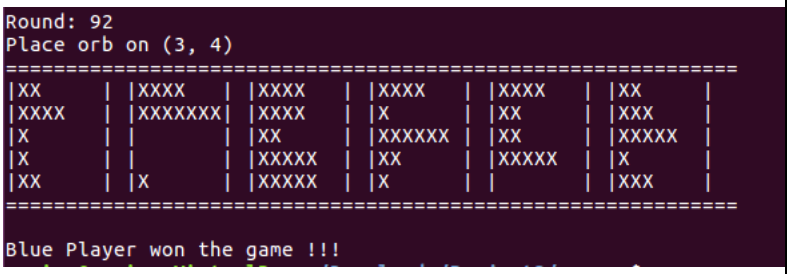
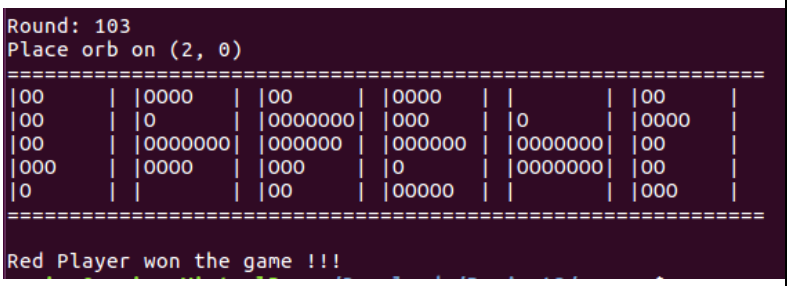
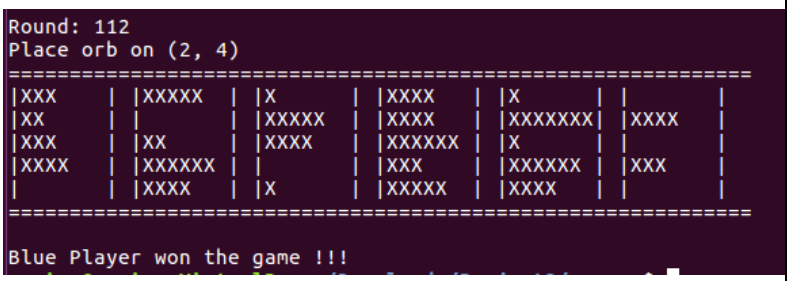
Figure 3: Use DFS with depth=2 to predict the next best moves and compare with the best move before and return the actual best move

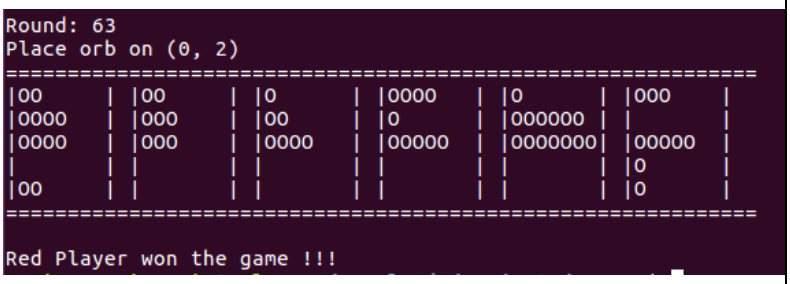
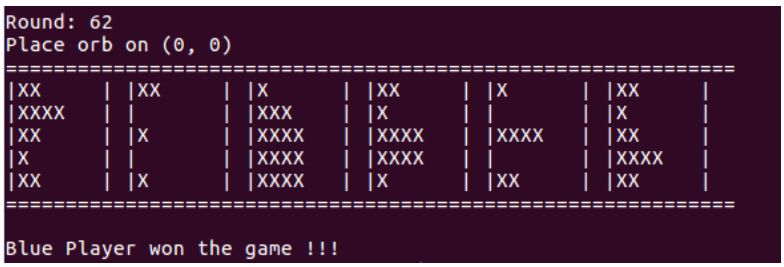
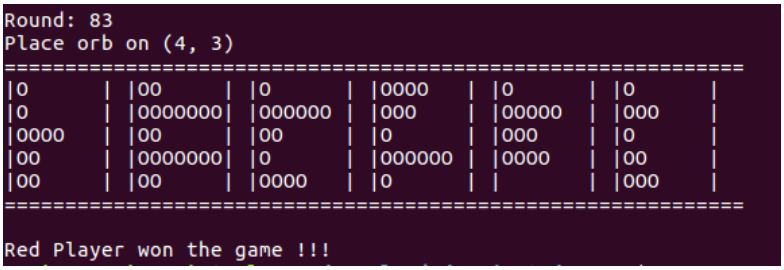
2-2) Github Control History

https://github.com/eunice-chen/Project3/commits/master



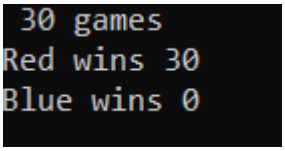
2-3) Compare with TA's AI Code (*randomMove*) for 7 results

No.	Red	Blue	
1.	algorithm_A	algorithm_B	
2.	algorithm_B	algorithm_A	
3.	algorithm_A	algorithm_B	
4.	algorithm_B	algorithm_A	

5.	algorithm_A	algorithm_B	
6.	algorithm_B	algorithm_A	
7.	algorithm_A	algorithm_B	

Result shown that when algorithm_A vs algorithm_B is 7:0

The algorithm is then test by using match.bat provided by the student, downloaded from iLMS.



Red is algorithm_A, blue is algorithm_B.

2-4) Describe the reason why you win TA's AI Code or why you can't win TA's AI Code.
Because the algorithm I designed uses the minimax algorithm. In minimax algorithm, I used BFS to obtained the best 12 results, and used the best result to go for the minimax algorithm again to get the best next result.

The random move AI is place the orbs randomly, so I can beat it with my algorithm.