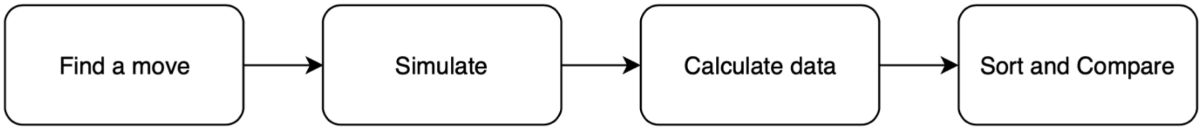
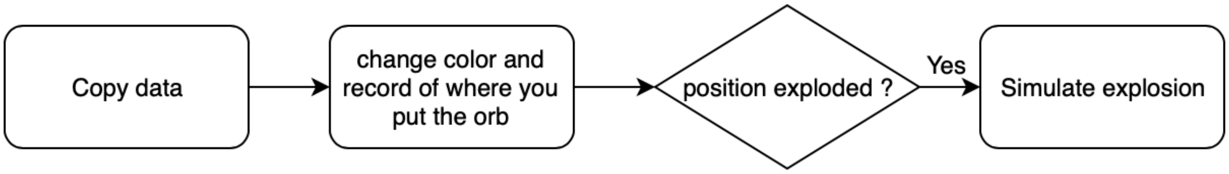
Project 3 Report
106062202

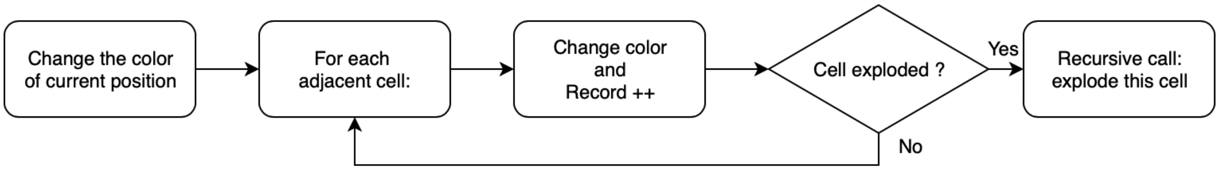
1. Program Description
    a. Program Flow Chart
        i. Core concept:



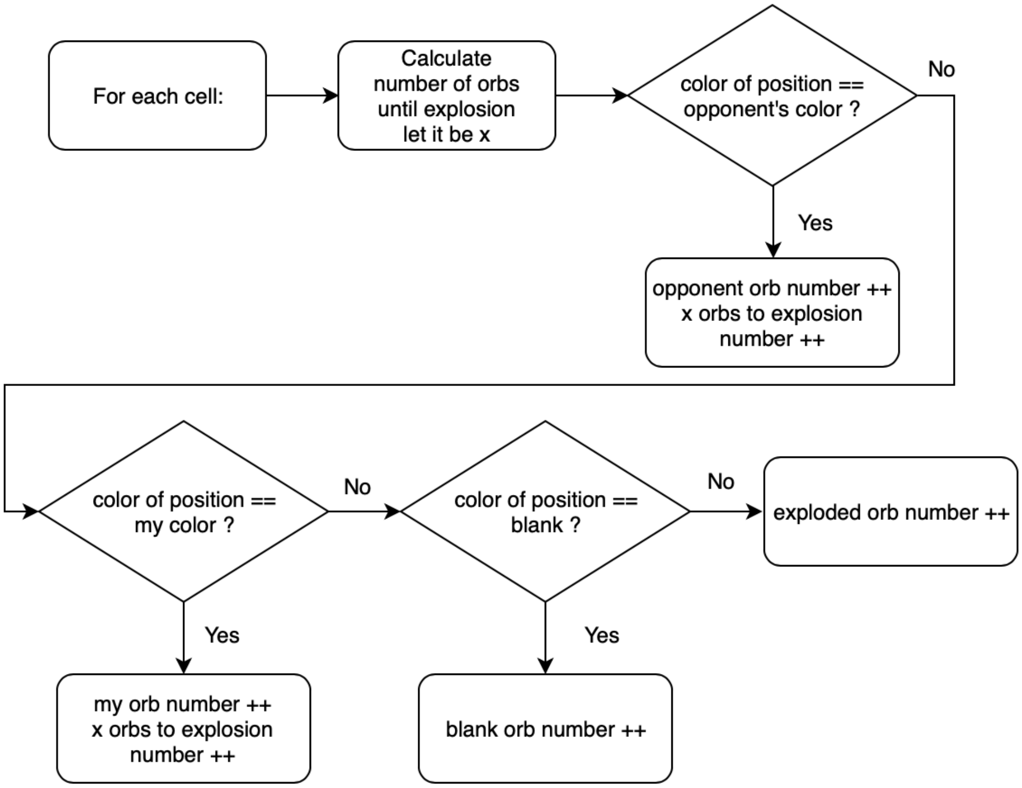
        ii. Simulation:



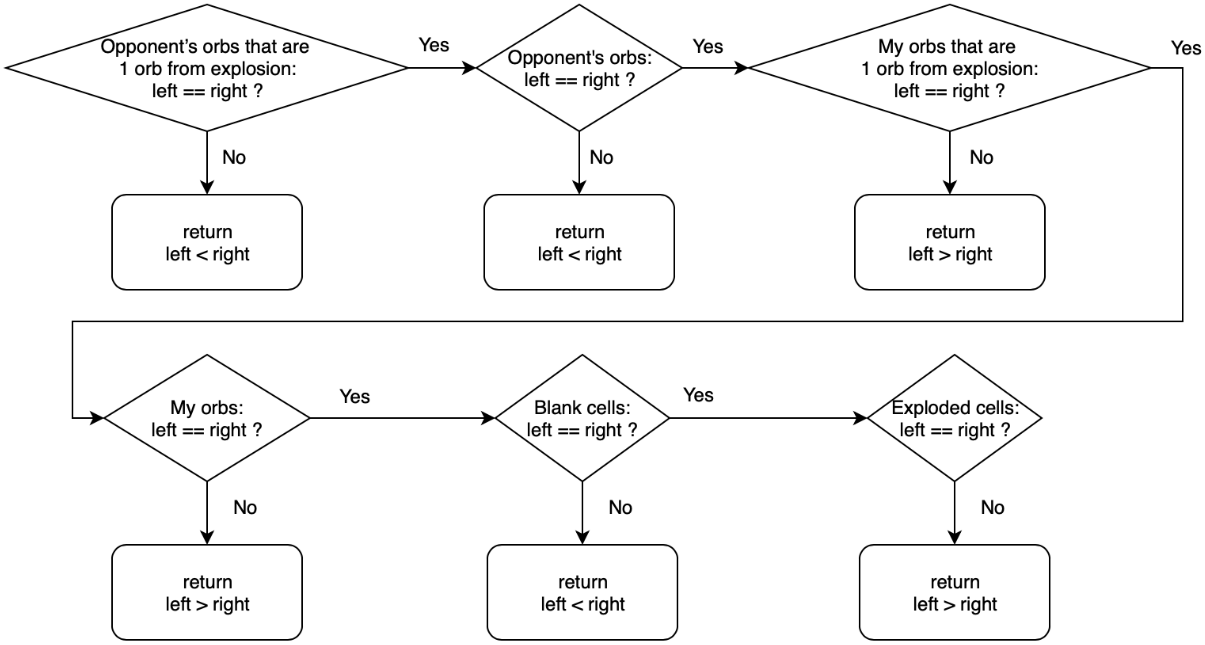
        iii. Explosion simulation:


Change the color of current position
For each adjacent cell:
Change color and Record ++
Cell exploded ?
Yes
Recursive call: explode this cell
No

        iv. Calculate Data:


For each cell:
Calculate number of orbs until explosion let it be x
color of position == opponent's color ?
No
Yes
opponent orb number ++ x orbs to explosion number ++
color of position == my color ?
No
color of position == blank ?
No
exploded orb number ++
Yes
Yes
my orb number ++ x orbs to explosion number ++
blank orb number ++

v. Operator overloading:


Opponent's orbs that are 1 orb from explosion: left == right ?
Yes
No
return left < right
Opponent's orbs: left == right ?
Yes
No
return left < right
My orbs that are 1 orb from explosion: left == right ?
Yes
No
return left > right
My orbs: left == right ?
Yes
No
return left > right
Blank cells: left == right ?
Yes
No
return left < right
Exploded cells: left == right ?
No
return left > right

b. Detailed Description

i. Core concept: Find all the possible moves and determine which move is most likely to lead to a win.

ii. Simulation: simulate if a move is taken, what the result would look like.

iii. Calculate data: gather information about the simulation results, including the number of orbs that are 1, 2, and 3 orbs away from explosion, the number of opponent's orbs and my orbs, the number of blank cells, and the number of exploded cells.

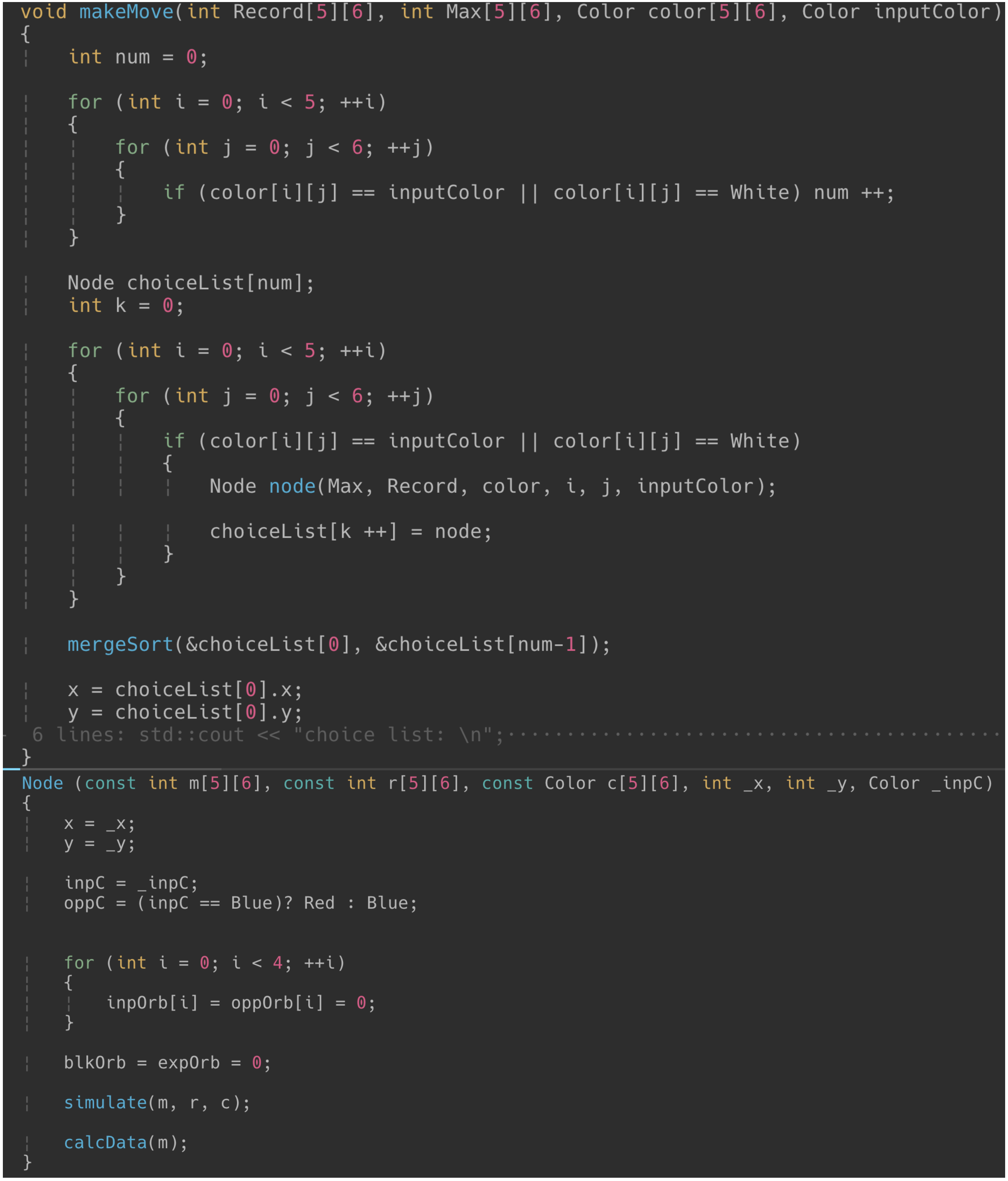
iv. Sorting: uses merge sort with the "greater than" operator overloaded.

v. Comparison: Determining which factor is more important.

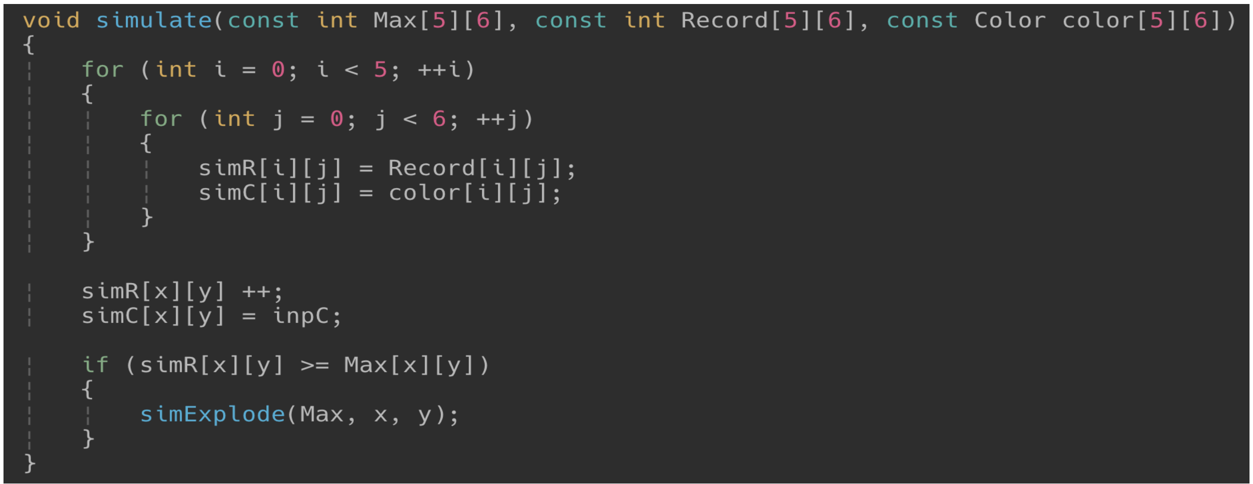
1. Minimum number of opponent's orbs that are 1 orb from explosion.
2. Minimum number of opponent's orbs.
3. Maximum number of my orbs that are 1 orb from explosion.
4. Maximum number of my orbs.
5. Minimum number of blank cells.
6. Maximum number of exploded cells.

2. Screenshots

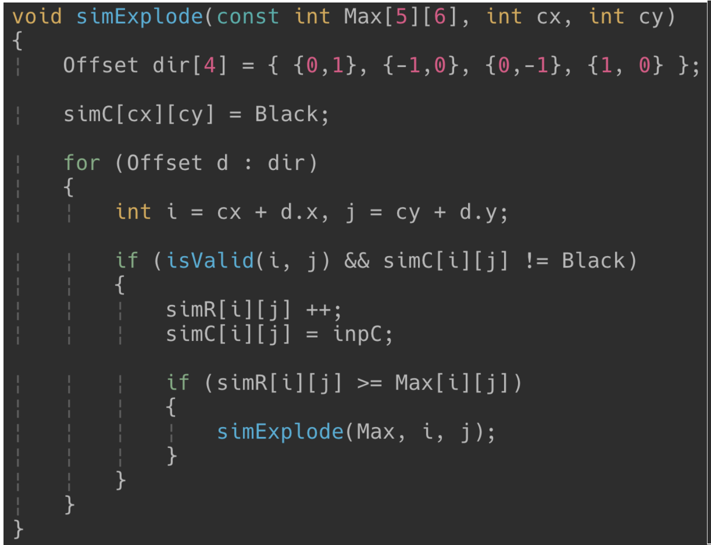
   a. Partial Implemented Code



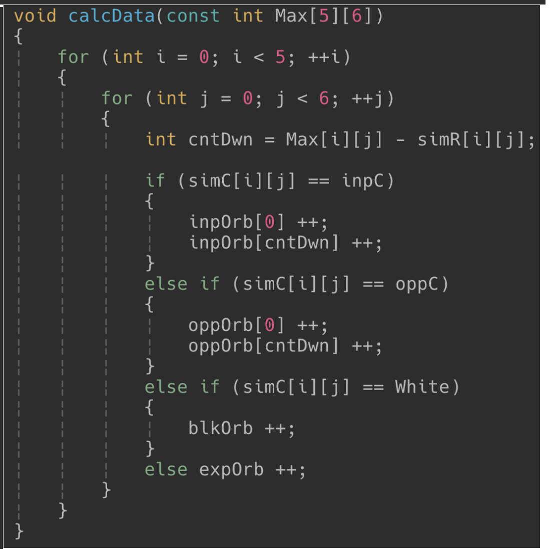
```
void makeMove(int Record[5][6], int Max[5][6], Color color[5][6], Color inputColor)
{
    int num = 0;

    for (int i = 0; i < 5; ++i)
    {
        for (int j = 0; j < 6; ++j)
        {
            if (color[i][j] == inputColor || color[i][j] == White) num ++;
        }
    }

    Node choiceList[num];
    int k = 0;

    for (int i = 0; i < 5; ++i)
    {
        for (int j = 0; j < 6; ++j)
        {
            if (color[i][j] == inputColor || color[i][j] == White)
            {
                Node node(Max, Record, color, i, j, inputColor);

                choiceList[k ++] = node;
            }
        }
    }

    mergeSort(&choiceList[0], &choiceList[num-1]);

    x = choiceList[0].x;
    y = choiceList[0].y;
 6 lines: std::cout << "choice list: \n";·····································
}
Node (const int m[5][6], const int r[5][6], const Color c[5][6], int _x, int _y, Color _inpC)
{
    x = _x;
    y = _y;

    inpC = _inpC;
    oppC = (inpC == Blue)? Red : Blue;


    for (int i = 0; i < 4; ++i)
    {
        inpOrb[i] = oppOrb[i] = 0;
    }

    blkOrb = expOrb = 0;

    simulate(m, r, c);

    calcData(m);
}
```



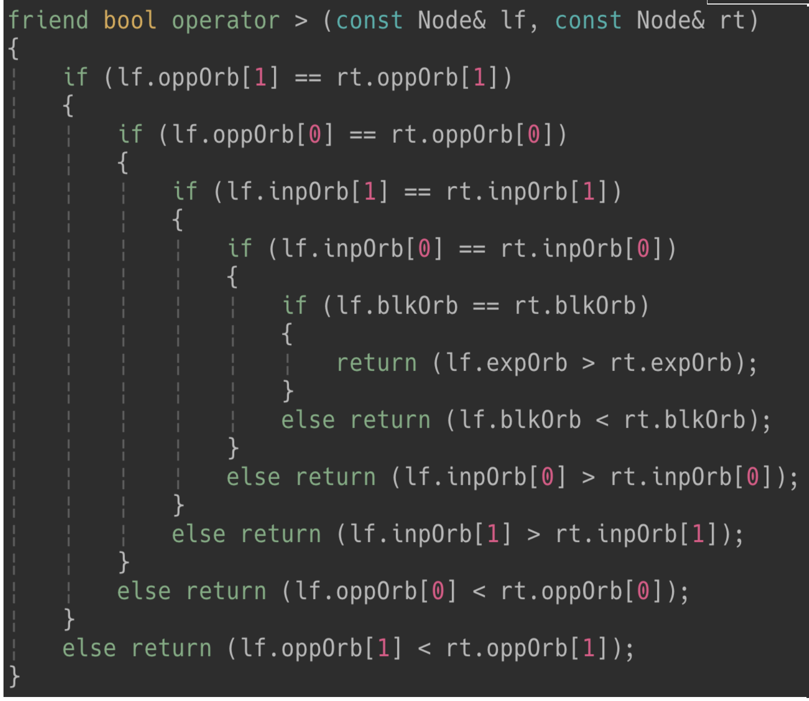
```
void simulate(const int Max[5][6], const int Record[5][6], const Color color[5][6])
{
    for (int i = 0; i < 5; ++i)
    {
        for (int j = 0; j < 6; ++j)
        {
            simR[i][j] = Record[i][j];
            simC[i][j] = color[i][j];
        }
    }

    simR[x][y] ++;
    simC[x][y] = inpC;

    if (simR[x][y] >= Max[x][y])
    {
        simExplode(Max, x, y);
    }
}
```



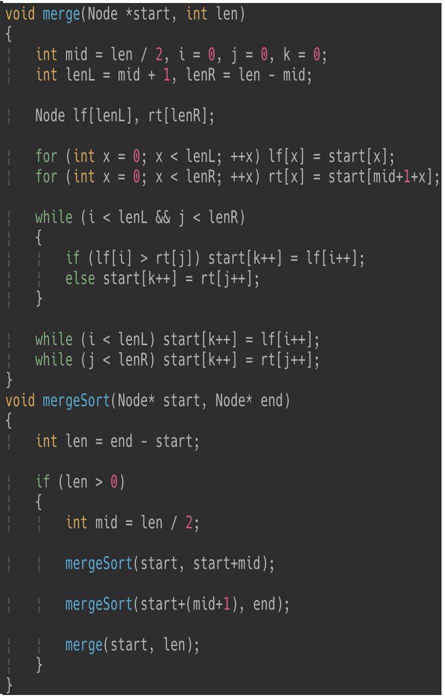
```
void simExplode(const int Max[5][6], int cx, int cy)
{
    Offset dir[4] = { {0,1}, {-1,0}, {0,-1}, {1, 0} };

    simC[cx][cy] = Black;

    for (Offset d : dir)
    {
        int i = cx + d.x, j = cy + d.y;

        if (isValid(i, j) && simC[i][j] != Black)
        {
            simR[i][j] ++;
            simC[i][j] = inpC;

            if (simR[i][j] >= Max[i][j])
            {
                simExplode(Max, i, j);
            }
        }
    }
}
```



```
void calcData(const int Max[5][6])
{
    for (int i = 0; i < 5; ++i)
    {
        for (int j = 0; j < 6; ++j)
        {
            int cntDwn = Max[i][j] - simR[i][j];

            if (simC[i][j] == inpC)
            {
                inpOrb[0] ++;
                inpOrb[cntDwn] ++;
            }
            else if (simC[i][j] == oppC)
            {
                oppOrb[0] ++;
                oppOrb[cntDwn] ++;
            }
            else if (simC[i][j] == White)
            {
                blkOrb ++;
            }
            else expOrb ++;
        }
    }
}
```

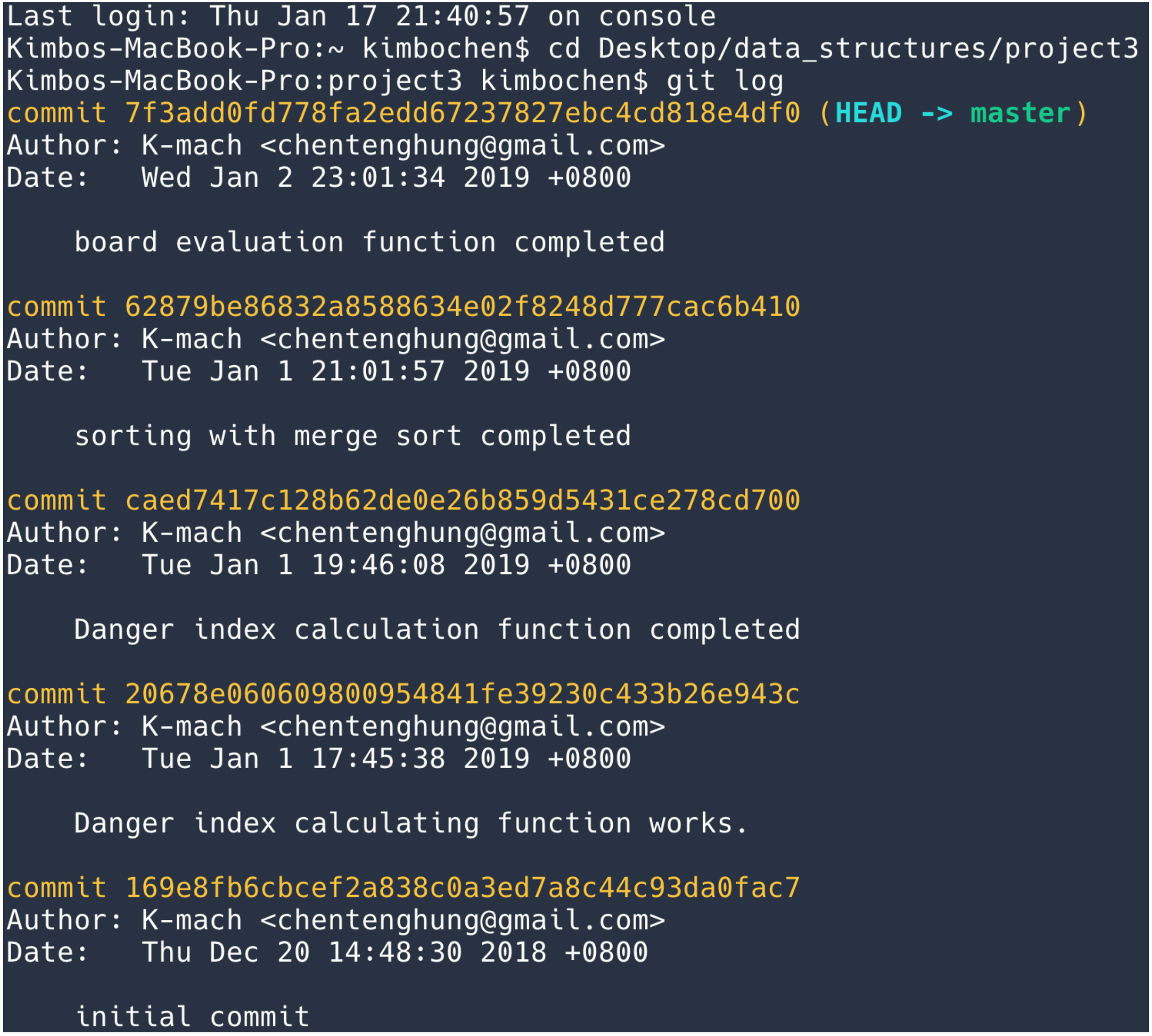


```
friend bool operator > (const Node& lf, const Node& rt)
{
    if (lf.oppOrb[1] == rt.oppOrb[1])
    {
        if (lf.oppOrb[0] == rt.oppOrb[0])
        {
            if (lf.inpOrb[1] == rt.inpOrb[1])
            {
                if (lf.inpOrb[0] == rt.inpOrb[0])
                {
                    if (lf.blkOrb == rt.blkOrb)
                    {
                        return (lf.expOrb > rt.expOrb);
                    }
                    else return (lf.blkOrb < rt.blkOrb);
                }
                else return (lf.inpOrb[0] > rt.inpOrb[0]);
            }
            else return (lf.inpOrb[1] > rt.inpOrb[1]);
        }
        else return (lf.oppOrb[0] < rt.oppOrb[0]);
    }
    else return (lf.oppOrb[1] < rt.oppOrb[1]);
}
```



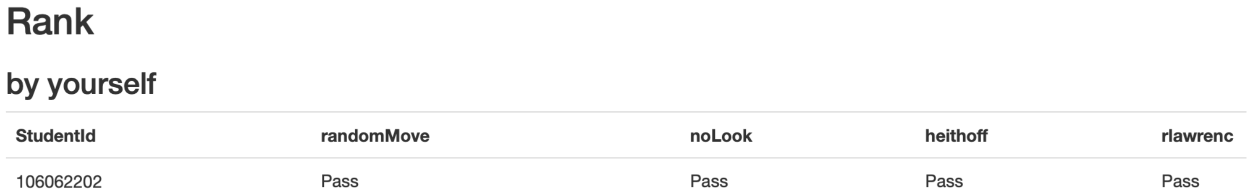
```
void merge(Node *start, int len)
{
    int mid = len / 2, i = 0, j = 0, k = 0;
    int lenL = mid + 1, lenR = len - mid;

    Node lf[lenL], rt[lenR];

    for (int x = 0; x < lenL; ++x) lf[x] = start[x];
    for (int x = 0; x < lenR; ++x) rt[x] = start[mid+1+x];

    while (i < lenL && j < lenR)
    {
        if (lf[i] > rt[j]) start[k++] = lf[i++];
        else start[k++] = rt[j++];
    }

    while (i < lenL) start[k++] = lf[i++];
    while (j < lenR) start[k++] = rt[j++];
}
void mergeSort(Node* start, Node* end)
{
    int len = end - start;

    if (len > 0)
    {
        int mid = len / 2;

        mergeSort(start, start+mid);

        mergeSort(start+(mid+1), end);

        merge(start, len);
    }
}
```

b. GitHub Control History



```
Last login: Thu Jan 17 21:40:57 on console
Kimbos-MacBook-Pro:~ kimbochen$ cd Desktop/data_structures/project3
Kimbos-MacBook-Pro:project3 kimbochen$ git log
commit 7f3add0fd778fa2edd67237827ebc4cd818e4df0 (HEAD -> master)
Author: K-mach <chentenghung@gmail.com>
Date:   Wed Jan 2 23:01:34 2019 +0800

    board evaluation function completed

commit 62879be86832a8588634e02f8248d777cac6b410
Author: K-mach <chentenghung@gmail.com>
Date:   Tue Jan 1 21:01:57 2019 +0800

    sorting with merge sort completed

commit caed7417c128b62de0e26b859d5431ce278cd700
Author: K-mach <chentenghung@gmail.com>
Date:   Tue Jan 1 19:46:08 2019 +0800

    Danger index calculation function completed

commit 20678e060609800954841fe39230c433b26e943c
Author: K-mach <chentenghung@gmail.com>
Date:   Tue Jan 1 17:45:38 2019 +0800

    Danger index calculating function works.

commit 169e8fb6cbcef2a838c0a3ed7a8c44c93da0fac7
Author: K-mach <chentenghung@gmail.com>
Date:   Thu Dec 20 14:48:30 2018 +0800

    initial commit
```

c. Competition with TA's AI

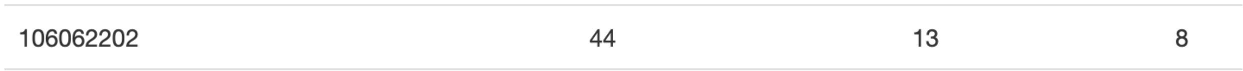


Rank

by yourself

StudentId	randomMove	noLook	heithoff	rlawrenc
106062202	Pass	Pass	Pass	Pass

d. My Rank



106062202	44	13	8