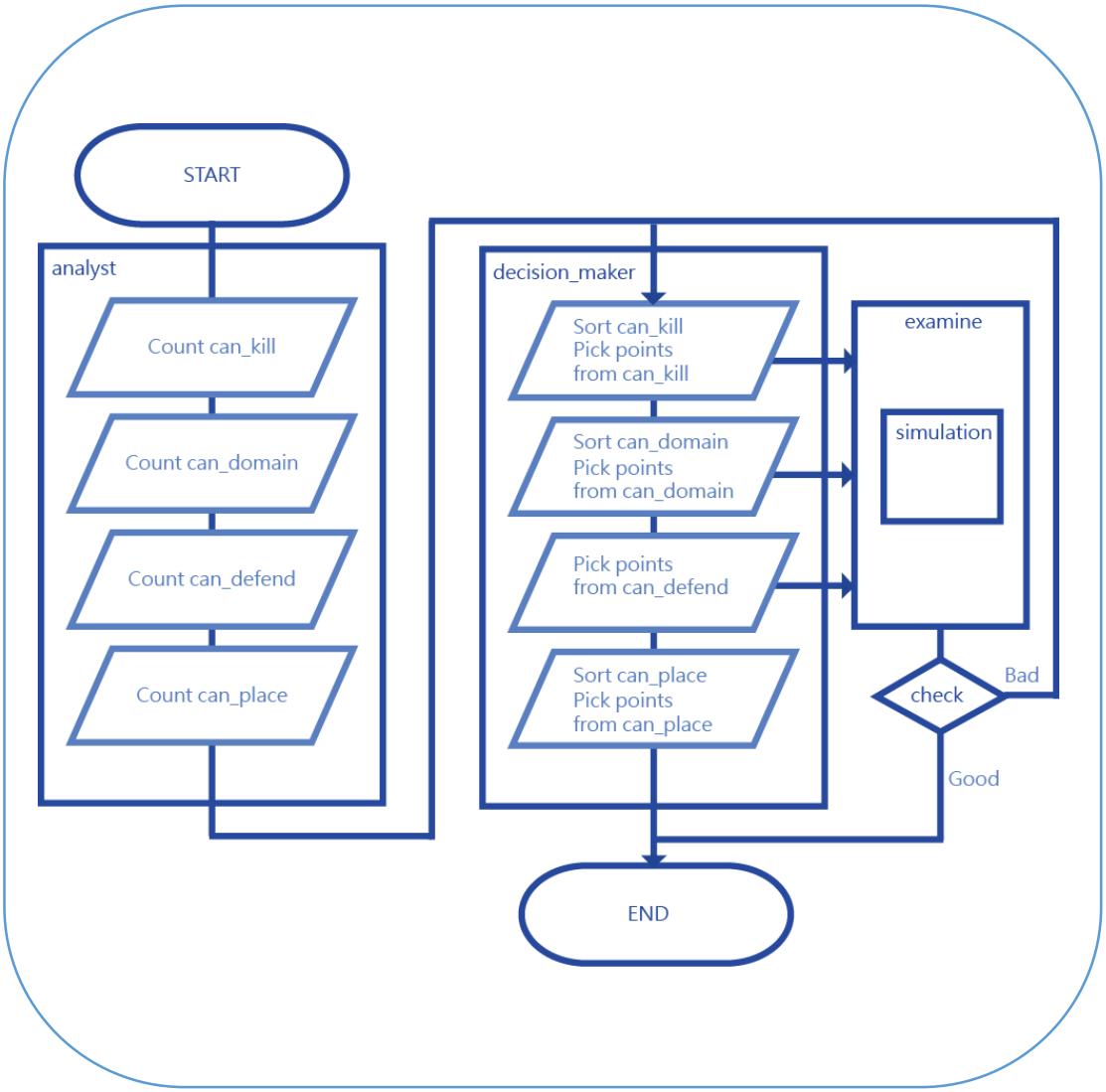
PROJECT3 – Chain Reaction

106000103 趙貞豪

Project Description

-Program Flow Chart:



-Detailed Description:

設計的架構如上圖所示，接下來我將一一細說各項目所要達成的目標以及實作方式，並闡述原因與想法。

首先，我要介紹這次 project 的流程。對照上圖來看，我們首先呼叫的是 analyst 這個 function，在這個函數真正計算我們所需要的資訊前，有一些事前

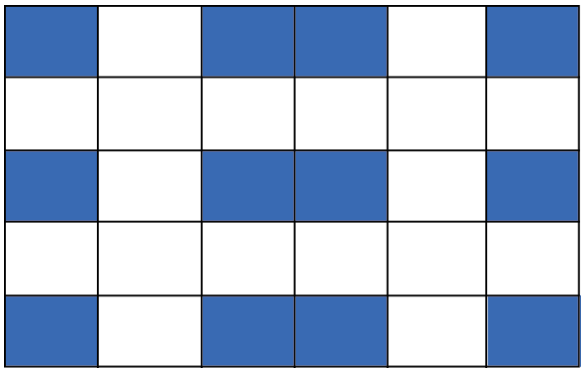
的準備需要完成：(1) attack_area 的建立、(2) threat_area 的建立、(3) defend_area 的建立、(4)conceal_area 的建立，基本上這個步驟有點像是初始化，把剛剛從棋盤上讀進來的資料做整理，以利之後作更進一步的處理。以下方表格說明個變數的意義及功能：

變數名稱	型別	功能
attack_area	*bool*	儲存我方棋子可以攻擊的座標點。
threat_area	*bool*	儲存敵方棋子可以攻擊的座標點。
defend_area	*bool*	儲存防禦敵方棋子的座標點。
conceal_area	*int*	儲存爆炸與當前棋數的差。(Max-Record)

接下來，有了這些資訊後就可以計算 Flow Chart 當中的 can_kill、can_domain、can_defend、can_place 了，以下表說明：

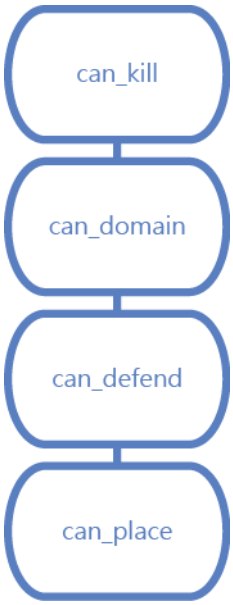
變數名稱	型別	算法
can_kill	*vector*	可殺點，針對每一個敵人點的周圍(四點)，如果不為 threat_area 且周圍點的 conceal_area<= 敵人點的 conceal_area。
can_domain	*vector*	優勢點，在優勢區域的點，周圍(四點)不能有其他點，且該點也不能有東西(White)。
can_defend	*vector*	安全點，周遭敵人的 canceal_area 都較大，或是周遭都是我方點。
can_place	*vector*	所有可以放的點。

我的算法有點像是 filter，先把所有點分類，一層層過濾，選擇最佔優勢的情況下手。首先，最好的選擇就是 can_kill 的點，是比較主動的去炸別人的情況，當然也要考慮到自己不會被反吃，因為要贏最主要還是要占地盤，所以這是第一順位。再來是 can_domain，這個是指優勢區域，顧名思義就是先搶先贏，搶到就有控制權，但是一被下完就沒有用了，所以 can_domain 只用在最初棋局剛開始的時候。can_domain 是第二順位。優勢區如下圖藍色區域：



而 can_defend 算是前面兩者的綜合版，位居第三順位。它的算法像是追蹤那些我放下去的點是不是還保有原本的優勢，如果沒有就要在該點下，取回優勢。而最後就是 can_place，這就是當前三者都沒有可以下的地方的時候，就會選在 can_place 下，儲存備用點，但這是最不理想的情況。

接下來就進到 decision_maker 的部分，在此要藉由上面所挑出的四個 vector 中取點。這裡分層進行取點，階層如下圖所示：



在進行挑點的過程會先把要選的點進 simulation 當中(我是使用 queue 來模擬連環爆)，進行檢查的動作(Flow chart 當中 examine 的部分)。如果放下去後會造成反攻的話，則會換一點進行。如果選擇確定後就會 return 要下的點，結束運算。

Screen Shots

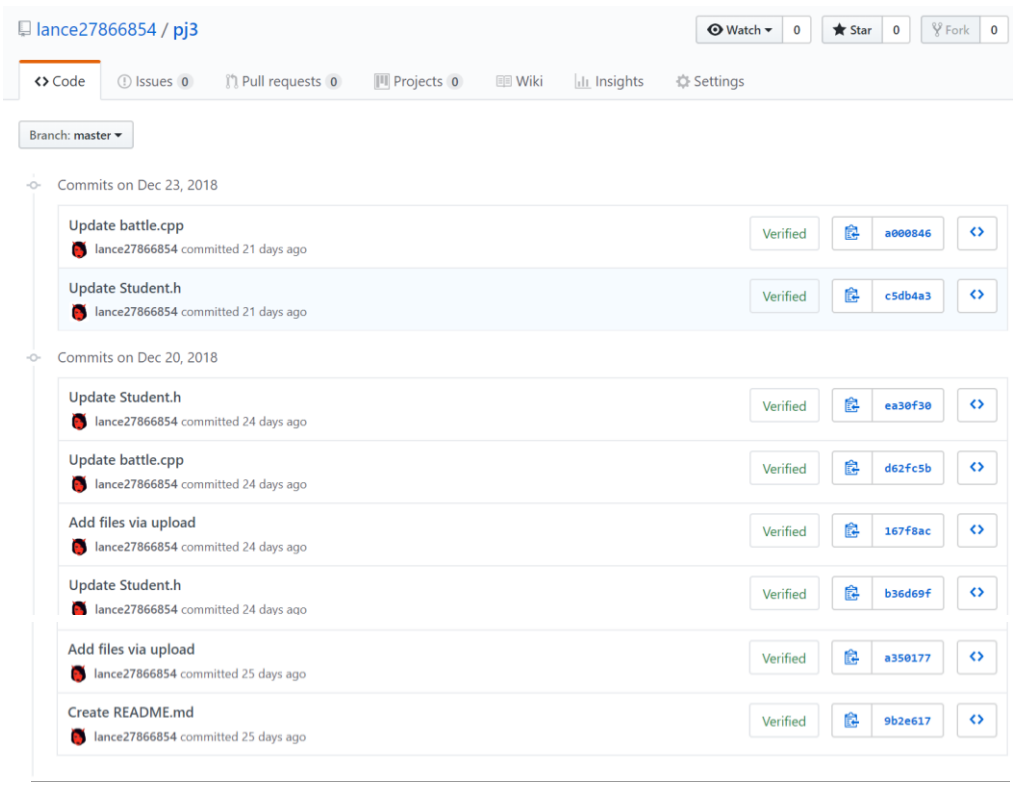
-Partial implemented Code:

```
///count can_kill
//1. enemy oriented.
//2. threat free
for(int i=0;i<5;i++){
    for(int j=0;j<6;j++){
        if(color[i][j]==enemyColor){
            int ex = conceal_area[i][j];
            int ex_u = (i!=0&&threat_area[i][j]==0&&(color[i-1][j]==inputColor||color[i-1][j]==White))? conceal_area[i-1][j] : FAIL;
            int ex_d = (i!=4&&threat_area[i][j]==0&&(color[i+1][j]==inputColor||color[i+1][j]==White))? conceal_area[i+1][j] : FAIL;
            int ex_l = (j!=0&&threat_area[i][j]==0&&(color[i][j-1]==inputColor||color[i][j-1]==White))? conceal_area[i][j-1] : FAIL;
            int ex_r = (j!=5&&threat_area[i][j]==0&&(color[i][j+1]==inputColor||color[i][j+1]==White))? conceal_area[i][j+1] : FAIL;
            if(ex>=ex_u) can_kill.push_back({ex_u+color[i-1][j], {i-1, j}});
            if(ex>=ex_d) can_kill.push_back({ex_d+color[i+1][j], {i+1, j}});
            if(ex>=ex_l) can_kill.push_back({ex_l+color[i][j-1], {i, j-1}});
            if(ex>=ex_r) can_kill.push_back({ex_r+color[i][j+1], {i, j+1}});
        }
    }
}

    ///count can_place
    for(int i=0;i<5;i++){
        for(int j=0;j<6;j++){
            if(color[i][j]==White||color[i][j]==inputColor){
                can_place.push_back({color[i][j], {i, j}});
            }
        }
    }

    ///count can_defend
    //the affiliated product of can_domain.
    for(int i=0;i<5;i++){
        if(i==1||i==3) continue;
        for(int j=0;j<6;j++){
            if(j==1||j==4) continue;
            if(color[i][j]!=inputColor) continue;
            int ex = Max[i][j]-Record[i][j];
            int ex_u = (i!=0&&!threat_area[i-1][j])? conceal_area[i-1][j] : FAIL;
            int ex_d = (i!=4&&!threat_area[i+1][j])? conceal_area[i+1][j] : FAIL;
            int ex_l = (j!=0&&!threat_area[i][j-1])? conceal_area[i][j-1] : FAIL;
            int ex_r = (j!=5&&!threat_area[i][j+1])? conceal_area[i][j+1] : FAIL;
            if(ex==ex_u||ex==ex_d||ex==ex_l||ex==ex_r){
                can_defend.push_back({0-Record[i][j], {i, j}});
            }
        }
    for(int i=0;i<5;i++){
        for(int j=0;j<6;j++){
            if(!(color[i][j]==inputColor||color[i][j]==White)) continue;
            bool ex_u = (i==0||color[i-1][j]==inputColor||conceal_area[i][j]>=conceal_area[i-1][j])? 1 : 0;
            bool ex_d = (i==4||color[i+1][j]==inputColor||conceal_area[i][j]>=conceal_area[i+1][j])? 1 : 0;
            bool ex_l = (j==0||color[i][j-1]==inputColor||conceal_area[i][j]>=conceal_area[i][j-1])? 1 : 0;
            bool ex_r = (j==5||color[i][j+1]==inputColor||conceal_area[i][j]>=conceal_area[i][j+1])? 1 : 0;
            if(defend_area[i][j]&&ex_u&&ex_d&&ex_l&&ex_r){
                can_defend.push_back({0-Record[i][j], {i, j}});
            }
        }
    }


     ///count can_domain
     for(int i=0;i<5;i++){
         if(i==1||i==3) continue;
         for(int j=0;j<6;j++){
             if(j==1||j==4) continue;
             bool color_flag = (color[i][j]==White)? 1 : 0;
             bool ex_u = (i!=0)? (Record[i-1][j]==0) : 1;
             bool ex_d = (i!=4)? (Record[i+1][j]==0) : 1;
             bool ex_l = (j!=0)? (Record[i][j-1]==0) : 1;
             bool ex_r = (j!=5)? (Record[i][j+1]==0) : 1;
             if(ex_u&&ex_d&&ex_r&&ex_l&&color_flag){
                 can_domain.push_back({Max[i][j], {i, j}});
             }
         }
     }
```

```
void decision_maker(int Record[5][6], int Max[5][6], Color color[5][6], Color inputColor,
            std::vector<t_Pair>& can_domain, std::vector<t_Pair>& can_kill, std::vector<t_Pair>& can_place,
            std::vector<t_Pair>& can_defend){
    std::sort(can_kill.begin(), can_kill.end());
    int s = can_kill.size();
    for(int i=0;i<s;i++){//std::cout<<"can_kill\n";
            int tx = can_kill[i].second.first;
            int ty = can_kill[i].second.second;
            if(examine(Record, Max, color, inputColor, tx, ty)){x = tx; y = ty; return;}
    }
```

```
    std::sort(can_domain.begin(), can_domain.end());
    s = can_domain.size();
    for(int i=0;i<s;i++){//std::cout<<"can_kill\n";
            int tx = can_domain[i].second.first;
            int ty = can_domain[i].second.second;
            if(examine(Record, Max, color, inputColor, tx, ty)){x = tx; y = ty; return;}
    }
    s = can_defend.size();
    for(int i=0;i<s;i++){//std::cout<<"can_kill\n";
            int tx = can_defend[i].second.first;
            int ty = can_defend[i].second.second;
            if(examine(Record, Max, color, inputColor, tx, ty)){x = tx; y = ty; return;}
    }
    if(!can_place.empty()){//std::cout<<"can_place\n";
        std::sort(can_place.begin(), can_place.end());
        x = can_place.front().second.first;
        y = can_place.front().second.second;
        return;
    }
    x = DEFAULT_X;
    y = DEFAULT_Y;
    return;


bool examine(int Record[5][6], int Max[5][6], Color color[5][6], Color playerColor, int x, int y){
    Color enemyColor = (playerColor == Blue)? Red : Blue;
    int sim_Record[5][6];
    int sim_Max[5][6];
    Color sim_color[5][6];
    for(int i=0;i<5;i++){
        for(int j=0;j<6;j++){
            sim_Record[i][j] = Record[i][j];
            sim_Max[i][j] = Max[i][j];
            sim_color[i][j] = color[i][j];
        }
    }
    simulation(sim_Record, sim_Max, sim_color,playerColor, x, y);
    for(int i=0;i<5;i++){
        for(int j=0;j<6;j++){
            if(sim_color[i][j]==enemyColor&&sim_Record[i][j]==sim_Max[i][j]-1){
                if(i!=0&&sim_Record[i-1][j]==sim_Max[i-1][j]-1&&color[i-1][j]==playerColor) return false;
                if(i!=4&&sim_Record[i+1][j]==sim_Max[i+1][j]-1&&color[i+1][j]==playerColor) return false;
                if(j!=0&&sim_Record[i][j-1]==sim_Max[i][j-1]-1&&color[i][j-1]==playerColor) return false;
                if(j!=5&&sim_Record[i][j+1]==sim_Max[i][j+1]-1&&color[i][j+1]==playerColor) return false;
            }
        }
    }
    return true;
}

 void simulation(int Record[5][6], int Max[5][6], Color color[5][6], Color playerColor, int x, int y){
     Color enemyColor = (playerColor == Blue)? Red : Blue;
     ///placement
     std::queue<std::pair<int, int>> chain_reaction;
     Record[x][y]++;
     color[x][y]=playerColor;
     if(Record[x][y]==Max[x][y]){
         color[x][y]=Black;
         chain_reaction.push({x-1, y});//UP
         chain_reaction.push({x+1, y});//DOWN
         chain_reaction.push({x, y-1});//LEFT
         chain_reaction.push({x, y+1});//RIGHT
     }

     while(!chain_reaction.empty()){
         //get.
         std::pair<int, int> position = chain_reaction.front();
         chain_reaction.pop();

         //chain reaction.
         int i = position.first;
         int j = position.second;
         if(i==-1||j==-1||i==5||j==6) continue;

         if(color[i][j]!=Black)color[i][j] = playerColor;
         if(color[i][j]!=Black)Record[i][j]++;

         if(Record[i][j]==Max[i][j]&&color[i][j]!=Black){
             color[i][j]=Black;
             chain_reaction.push({i-1, j});//UP
             chain_reaction.push({i+1, j});//DOWN
             chain_reaction.push({i, j-1});//LEFT
             chain_reaction.push({i, j+1});//RIGHT
         }
     }
```

- GitHub Control History:



- Compare with TA's AI Code with Student Id:

Rank

by yourself

StudentId	randomMove	noLook	heithoff	rlawrenc
106000103	Pass	Pass	Pass	Pass

- Your Rank with Student Id:（截圖時間 1/15，請助教 check 最後排名。）

106034071	3	175	171	Challenge
105021217	4	126	159	Challenge
106062223	5	51	67	Challenge
106070038	6	44	34	Challenge
106081005	7	72	27	Challenge
106000103	8	68	66	