인공지능 중간 프로젝트

Artificial Intelligence Midterm Project Report

- [인공지능괴롭조]
- 조원 : 2015410055 김연
  : 2015410056 김지윤
  : 2015410100 임수경

- 제출 일자 : 2017년 5월 10일
- 담당 교수 : 김현철 교수님

[목차]

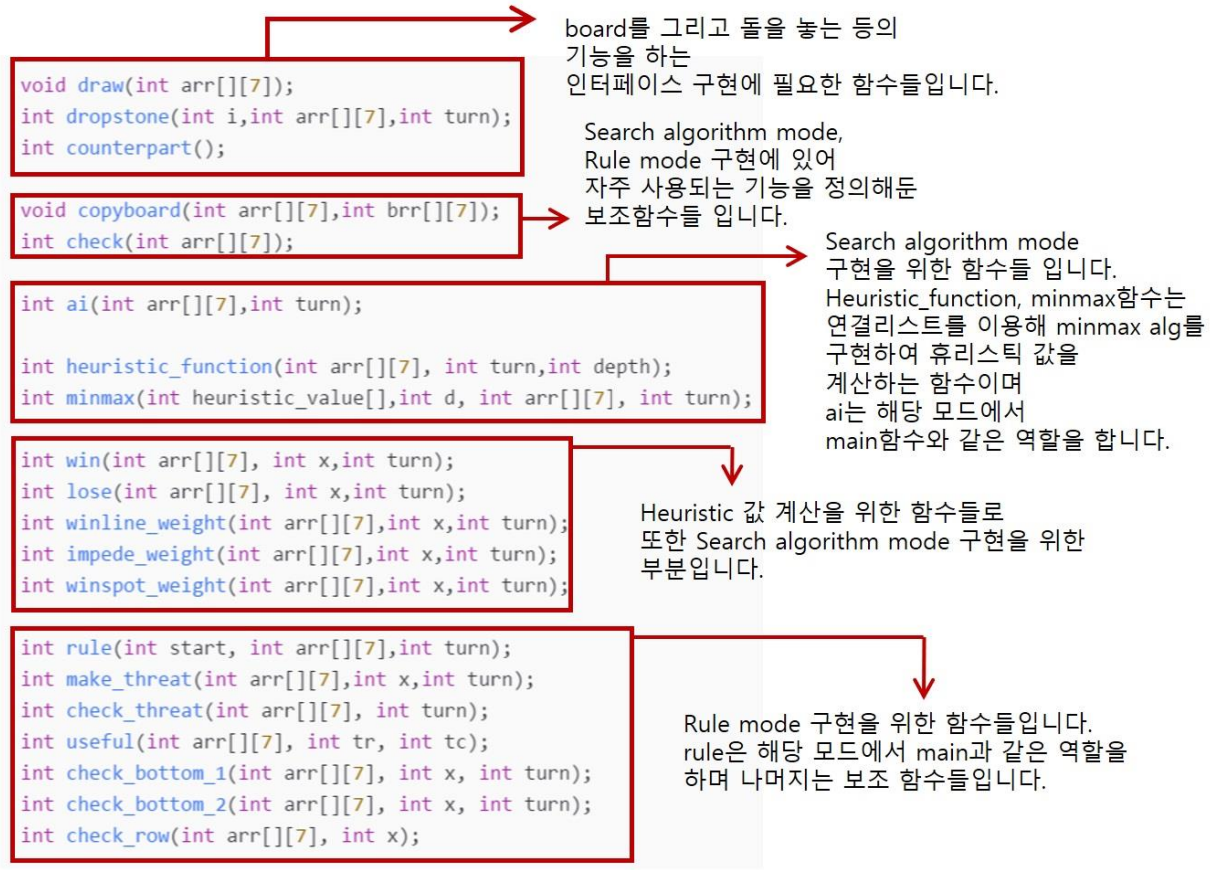
1. Connect 4 AI

Connect 4는 두 사람이 진행하는 게임으로 [6x7]의 게임board에 번갈아 가며 column을 선택해 착수하며 가로, 세로, 양 대각선 방향으로 돌 4개의 connection을 먼저 완성하는 사람이 이기는 것이 규칙입니다. 아래에 돌이 있어야 착수가 가능하므로 착수점은 column값 만으로 선택하게 됩니다.

일정 수 앞을 내다보고 착수점을 결정하는 Search Algorithm과 현재 board상의 돌 배치 만을 보고 정해진 규칙에 따라 착수점을 결정하는 Rule의 두 가지 mode를 구현했고 저희 조는 Search Algorithm mode에서 최대 일곱 수 앞까지 내다보도록 했습니다. 이때 connect 4에서 선택은 서로 돌아가면서 동등하게 진행되므로 MINMAX 알고리즘을 이용했습니다.

A. code structure

전체 코드는 크게 interface와 Search Algorithm mode, Rule mode 각각을 구현한 부분, 그리고 main 함수 이렇게 네 파트로 나눌 수 있습니다. 코드에 포함되어 있는 함수들의 prototype과 간략한 설명은 아래와 같습니다.

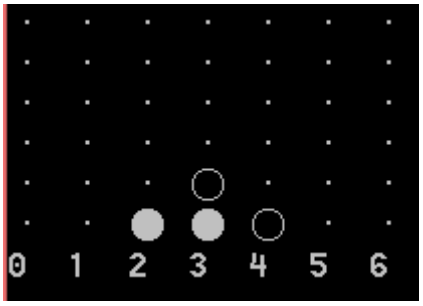


각 파트 구현에 대한 구체적인 내용과 함수의 기능에 대해서는 아래에 자세히 설명하겠습니다.

B. Interface

a. draw( )

connect 4 경기를 위한 board를 그려주는 함수입니다. 배열의 value가 -1일 경우 즉 초기화 된 값 그대로 가지고 있는 경우 아직 돌이 놓이지 않은 상태이므로 ' .'으로 나타냈고 0 또는 1의 값을 가지면 원으로 돌을 두되 색으로 구분해주었습니다. 코드를 실행시키면 오른쪽과 같이 board가 그려지게 됩니다.



b. dropstone( )

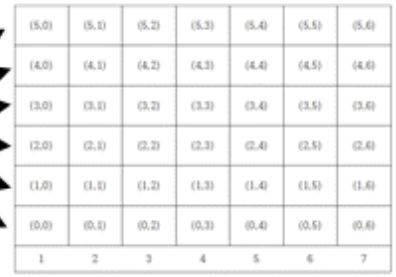
board에 돌을 두는 함수입니다. 가장 아래 row 부터 검사하여 돌이 없는 칸(배열 해당 칸의 value가 -1)에 대해 값을 바꿔줍니다. arr는 turn%2의 값을 가지게 되는데 배열에 짝수 번째 turn이라면 0, 홀수 번째 turn이라면 1을 저장하게 되고 상대와 내가 0과 1을 번갈아 저장하도록 작동합니다.

c. counterpart( )

상대팀이 돌을 어디에 두는지 column값을 입력 받는 간단한 함수입니다.

C. main 함수

```
37  int main(){
38      int arr[6][7]={
39        {-1,-1,-1,-1,-1,-1,-1},
40          {-1,-1,-1,-1,-1,-1,-1},
41          {-1,-1,-1,-1,-1,-1,-1},
42          {-1,-1,-1,-1,-1,-1,-1},
43          {-1,-1,-1,-1,-1,-1,-1},
44          {-1,-1,-1,-1,-1,-1,-1}};
45      int brr[6][7];
46      char start,a;
47      int mode;
48      int turn=0;
49      printf("상대팀이 먼저 플레이 합니까?(Y/N) : ");
50      scanf("%c",&start); a=getchar();
```



메인 함수에서는 우선 돌을 놓을 board로 arr[6][7]을 선언해줍니다. 초기화는 -1으로 되어있으며 돌을 두게 되면 0 또는 1의 값을 갖게 됩니다. 그리고 arr를 인자로 받는 함수 선언을 위해 arr를 복사할 새로운 배열 brr[6][7]을 만들었습니다.

먼저 상대방이 먼저 시작할지 내가 먼저 시작할지 start변수에 입력 받아 Y일 경우 상대방이 먼저, N일 경우 connect 4 AI가 말을 두도록 하였습니다.

```
52      while(1){
53         printf("search algorithm 모드는 1, rule 모드는 2를 입력하세요: ");
54          scanf("%d",&mode); a=getchar();
```

위의 while문은 경기가 끝날 때까지 반복되며 매 turn마다 search algorithm mode인지 rule mode인지 확인합니다.

Search algorithm mode를 선택할 경우 ai 함수를 호출하여 어느 column을 선택할지 Heuristic 값을 계산하여 return하고 rule mode를 선택할 경우 rule 함수를 호출하여 규칙에 따라 최적의 column을 찾아 return합니다.

만약 start값이 N으로 AI가 첫 수를 놓을 수 있게 된다면 첫 수는 가운데 column, (1,4)를 선택하는 것이 가장 유리하므로 이부분은 main함수에서 처리합니다.

D. 보조 함수

a. copyboard( )

Search Algorithm과 Rule mode 구현을 위해 쓰이는 함수들 중 다수는 현재 board 상태를 나타내는 arr를 인자로 전달받습니다. 이때 기존 arr 변형시키지 않기 위해서는 새로운 배열에 복사해서 인자로 전달하는 과정이 필요한데 copyboard()가 배열을 복사하는 기능을 합니다.

b. check( )

board의 배열을 인자로 받아 상대방의 돌이든 나의 돌이든 관계없이 같은 종류의 돌이 연속 4개 연결되어 connection을 완성하는지 확인하는 함수입니다. 가로, 세로, 양대각선 방향을 모두 체크하며 해당 기능은 Search Algorithm, Rule mode상관없이 굉장히 자주 사용되었기에 따로 함수로 정의해주었습니다.

2. Search Algorithm Mode

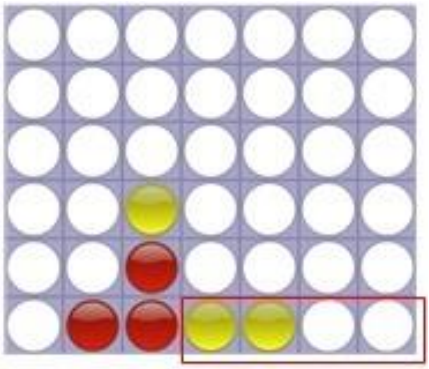
Search Algorithm은 fixdepth만큼의 수를 내다보는 방식으로 모든 게임 판의 연결리스트를 저장하여 각 상황별로 여러 함수를 이용해 heuristic value를 계산합니다. 이때, 몇몇 상황의 경우 예외처리를 하여 특정 column을 무조건 선택하게 하거나 혹은 절대로 선택하지 않도록 heuristic value를 다시 계산해 줍니다. 이렇게 계산된 heuristic value를 토대로 MINMAX 알고리즘을 실행시켜 최종적으로 우리 팀이 놓을 column값을 return합니다.

A. Heuristic 함수

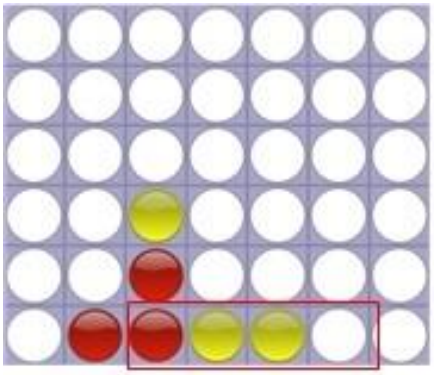
Heuristic 값은 다음 네 가지 함수 결과값의 합으로 표현됩니다.

a. winline_weight( )

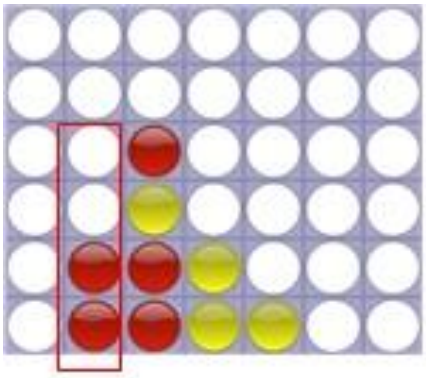
winline_weight 함수는 특정 라인으로 이길 가능성이 얼마나 높을 지 확인해보고 가중치를 두는 함수입니다. 인자로 받은 column에 내 돌을 둬보고 연속 4칸 상대방 돌이 없다면 그 줄은 나에게 유리하다고 볼 수 있으므로 가중치 w를 올려줍니다. 반대로 연속 4칸에 내 돌이 없이 이 위치에 두어도 connection을 완성할 가능성이 낮다고 판단되면 w를 낮춰줍니다.



예를 들어 왼쪽과 같은 상황에서 노란 색 돌이 상대방이라고 하면 빨간 네모 칸 안의 부분은 가로 연속 4개가 내 돌이 아니므로 w값을 낮춰주게 됩니다.



이렇게 4칸을 체크하는 경우 상대방 돌과 내 돌이 라인 안에 섞여 있으므로 w값을 변화시키지 않습니다.



이렇게 4칸을 체크하는 경우에는 라인 내에 상대방 돌이 하나도 없으므로 w값을 증가시켜줍니다.

가로, 세로 양쪽 대각선 모든 방향의 라인에 대하여 w값을 확인하고 이렇게 계산한 w값을 return하여 최적의 column을 찾게 합니다.

b. winspot_weight( )

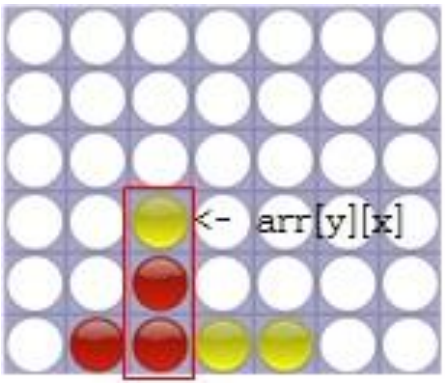
이 함수는 앞으로의 두 수까지 내다봐서 내가 이 위치에 돌을 두면 이길 수 있는지, 상대가 이 위치에 돌을 두고 그 위에 내 돌을 두면 이길 수 있는지 등의 가능성을 계산하여 가중치를 올려주는 기능을 합니다.

Spot 변수를 통해 이 위치가 이길 위치라면 spot값을 올려주고 질 위치라면 값을 내

려줍니다. spot 값이 여러 번 더해지고 빼질 수 있기 때문에 최종 spot 값이 0이 나오지 않도록 351과 같은 값을 사용하였고, 지게 되는 경우는 무조건 피해야하므로 지게 될 위치에서 spot 값을 더 크게 낮춰줍니다.

c. impede_weight( )

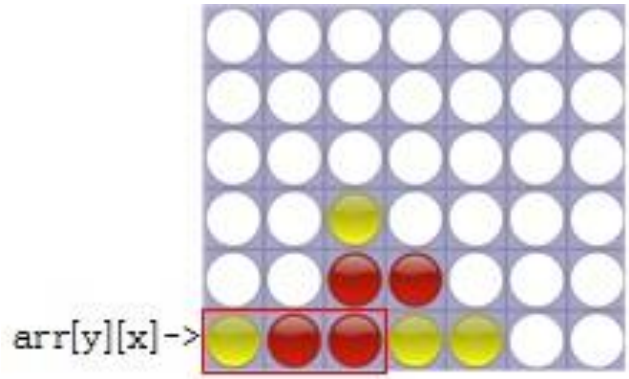
이 함수는 상대방을 막아줄 수 있는 위치에 가중치를 두는 함수입니다. 연속 3개인 말을 막아주는 것은 ai함수에서 처리하기 때문에, 이 impede함수는 연속 2개의 말을 막을 수 있다면 가중치를 주어 최대한 상대가 3목 이상을 만드는 것을 막아주는 함수입니다.



왼쪽과 같은 상황에서 내 돌이 노란색 돌이라면 빨간 네모 칸 안에서 세로 연속 두개인 상대방 돌을 내 돌이 막았기 때문에 가중치를 올려줍니다.

arr[y][x]라는 위치에 내 돌이 놓인다고 가정했기 때문에 그 주변으로만 체크하면 됩니다.

마찬가지로 가로로 연속 2개 있는 상대방 돌을 arr[y][x]에 내 돌을 둠으로써 막아준다면 가중치를 올려줍니다.



d. win( )

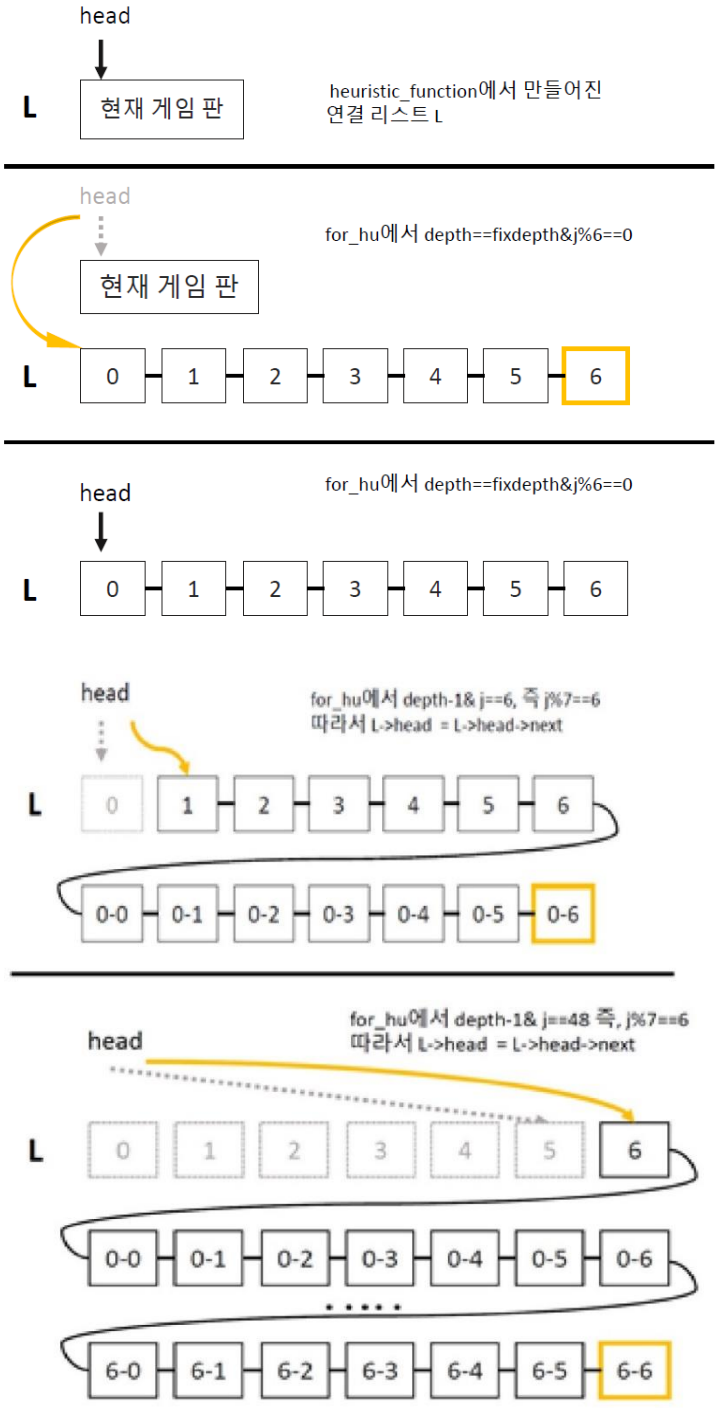
column을 인자로 받아 그 자리에 내 돌을 둬보고 check함수를 호출해 connection이 완성되는지 확인합니다. Connection이 이루어진다면 우선적으로 선택해야 하므로 1000이라는 큰 가중치를 줍니다.

B. heuristic_fuction( )

이 함수는 for_hu 함수와 minmax 함수를 연결 시켜주는 역할로 heuristic값이 가장 큰 column을 return합니다. 연결리스트를 생성하고 현재 게임 판을 복사한 후, 연결리스트의 첫 번째 노드의 data[6][7]에 넣어줍니다. 이후에 for_hu함수를 수행하고 depth와 fixdepth가 같다면 minmax함수를 실행시킵니다.

C. for_hu( )

heuristic_function에서 만들어진 연결리스트 L을 이용해 휴리스틱 값을 계산하는 함수입니다. fixdepth는 몇 수 앞을 내다볼지를 나타내는 변수이며 연결리스트는 아래의 그림과 같이 전체 트리의 일부 노드만 가지고 있습니다. 각 칸에 나타난 숫자는 현재 게임 판에 놓여질 column의순서 열이며 왼쪽이 먼저 입력된 값입니다.



1) depth <= fixdepth

   depth 당 총 7^(fixdepth-depth+1)개의 노드가 생성됩니다. 특히 j%7==6일 때 L의 head는 다음 노드를 가리키게 됩니다.

2) depth == 1

   총 7^7개의 노드가 생성됩니다. 각 노드의 게임 board 상황마다 heuristic value를 계산합니다.

변수 num은 생성된 노드의 게임 board에 놓인 전체 돌의 수이며 poss는 현재 게임 board에 놓인 전체 돌의 수입니다. State는 해당 노드의 게임 판에 7수가 놓인 순서를 십진수로 나타낸 것인데 예를 들어 0-6-0-0-0-1-2 순으로 입력되었다면 7진수 0600012를 십진수로 표현한 값인 6*7^6+1*7^1+2*7^0이 state가 됩니다. 0-0-0-0-0-0-3의 경우 state값은 3 입니다.

1) num-poss == fixdepth-1일 경우

   이 상황은 생성된 노드의 게임 board에 놓여진 전체 돌의 수가 각 column당 놓을 수 있는 최대 돌의 개수 (5개)를 넘지 않았을 경우입니다. 이때 상대가 connection을 완성했으면 패배한 상황이기 때문에 impossible값을 줍니다. 그렇지 않다면 heuristic_value[state]에 heuristic 값을 계산하여 넣어주는데 heuristic 값을 계산하는 함수h(x)는 위에서 설명한 heuristic 함수 결과 값의 합 winline_weight + winspot_weight + winspot_weight + impede_weight + win로 나타냅니다..

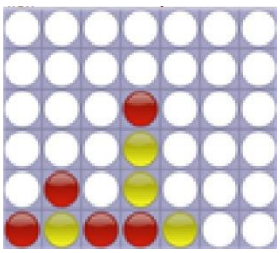
2) num-poss != fixdepth-1일 경우

   이 경우는 각 column당 놓을 수 있는 최대 돌의 수를 넘은 것이기 때문에 impossible값을 줍니다.

7^7개 노드의 heuristic 값이 모두 결정되면 함수를 종료합니다.

한편, 아래와 같은 상황에 대해서는 예외처리를 해서 heuristic_value를 max 혹은 min으로 나오게 만들어 처리했습니다.

1) 상대가 자신의 차례일 때 우리 팀이 직전에 놓은 돌과 같은 column을 선택해 승리할 경우 모든 heuristic_value[state]에 대해 (단, state = 7진수 izz_zzzz / i : 해당 column값, z : 0~6 사이의 수) impossible로 값을 바꿔줍니다. 오른쪽 그림은 해당 상황을 보여주는 예시입니다. 만약 내가 세 번째 column을 착수점으로 선택한다면 상대방은 바로 connection을 완성



할 수 있는 조건을 가지게 됩니다.

2) Priority 함수를 이용해, 게임 board 상황이 priority에서 요구된 조건을 만족하는 상황일 경우 특정한 column값을 반환합니다. Priority 함수는 이번 차례에 무조건 놓아야하는 column값을 반환하기 때문에 모든 heuristic_value[state] (단, state = 7진수 izz_zzzz / i : 해당 column값, z : 0~6 사이의 수)를 must로 바꿔줍니다.

D. minmax( )

For_hu함수를 통해서 얻은 heuristic_value[x]를 이용해 minmax알고리즘을 실행하여 우리 팀이 선택할 column값을 return하는 함수입니다.

```
if(i%2==1){
    if(max<heuristic_value[j]){
        max=heuristic_value[j];
    }
}
else{
    if(min>heuristic_value[j]){
        min=heuristic_value[j];
    }
}
```

```
if(j%7==6){
    if(i%2==1){
        if(max==-1234||max==-1000){
            heuristic_value[(j+1)/7-1]=99999;
        }
        else if(max==9999)
        {
            heuristic_value[(j+1)/7-1]=-9999;
        }
        else{
            heuristic_value[(j+1)/7-1]=max;
        }
    }
    else{
        if(min==-1234||min==99999){
            heuristic_value[(j+1)/7-1]=-99999;
        }
        else if(min==-9999)
        {
            heuristic_value[(j+1)/7-1]=9999;
        }
        else{
            heuristic_value[(j+1)/7-1]=min;
        }
    }
    max=-1000;min=99999;
}
```

* 왼편의 두 코드는 모두 같은 j에 대해서 일어나는 for문 내 코드입니다.

* i: 전체 트리 상의 depth를 나타냅니다.

* heuristic_value[j]를 heuristic_value[0]부터 ① i가 홀수이면 max을, ② i가 짝수이면, min 값을 계산합니다.

* j%7==6일 때, ① i가 홀수이면 계산된 max을, ② i가 짝수이면, 계산된 min 값을 heuristic_value[(j+1)/7-1]에 저장합니다.

* ①impossible한 상황 중 상대방이 우리 팀이 놓은 column x와 같은 column을 선택 할 경우

i==홀수 일 때, heuristic_value[(j+1)/7-1]에 99999를 넣어주고.

I==짝수 일 때, min==99999이므로 heuristic_value[(j+1)/7-1]에 –99999를 넣어줍니다.

이를 반복하면, 최종적으로 heuristic_value[x]는 –99999가 되어

해당 column을 선택하지 않도록 합니다.

*②priority 함수에 의해 무조건 놓아야 하는 column이 있을 때, 모든 heuristic_value[state] (단, state = 7진수 xzz_zzz, i : column x, z : 0~6 사이의 수)가 must 이므로

i ==홀수 일 때, heuristic_value[(j+1)/7-1]에 -9999를 넣어주고

i ==짝수 일 때, min==-9999이므로 heuristic_value[(j+1)/7-1]에 9999를 넣어줍니다.

이를 반복하면 최종적으로 heuristic_value[x]는 9999가 되어 해당 column을 반드시 선택하게 됩니다.

*③그 외, 일반적인 minmax 알고리즘이 실행되는 방식으로 값을 계산하여 최종적으로 얻은 heuristic_value[0]~heuristic_value[6] 중 최댓값을 갖는 heuristic_value[x]의 인텍스 x를 반환합니다.

E. ai( )

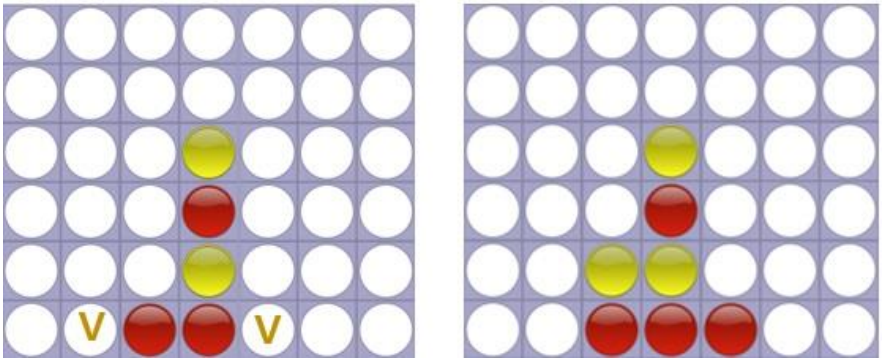
ai 함수는 Search Algorithm mode에서 메인 함수와 같은 역할을 합니다.

먼저 현재 게임 board의 놓여있는 돌의 총 개수를 계산해 poss라는 변수에 저장하고 poss값에 따라 (빈칸의 개수에 따라) fixdepth값을 지정해줍니다. 이후에 fixdepth를 인자로 받는 heuristic_function을 호출하고 결과값을 return합니다.

이때 함수 결과 값이 -3일 경우 (모든 column의 값이 동일하게 -99999일 때)는 ai 함수 내에서 현재 board 상황에 대해 column별로 heuristic 값을 계산해 (winline_weight + winspot_weight + winspot_weight + impede_weight + win) 최댓값을 가지는 column을 return합니다.

F. priority( )

아래의 그림의 상황과 같이 반드시 선점해야 하는 column이 있는 경우를 처리해주는 함수입니다.

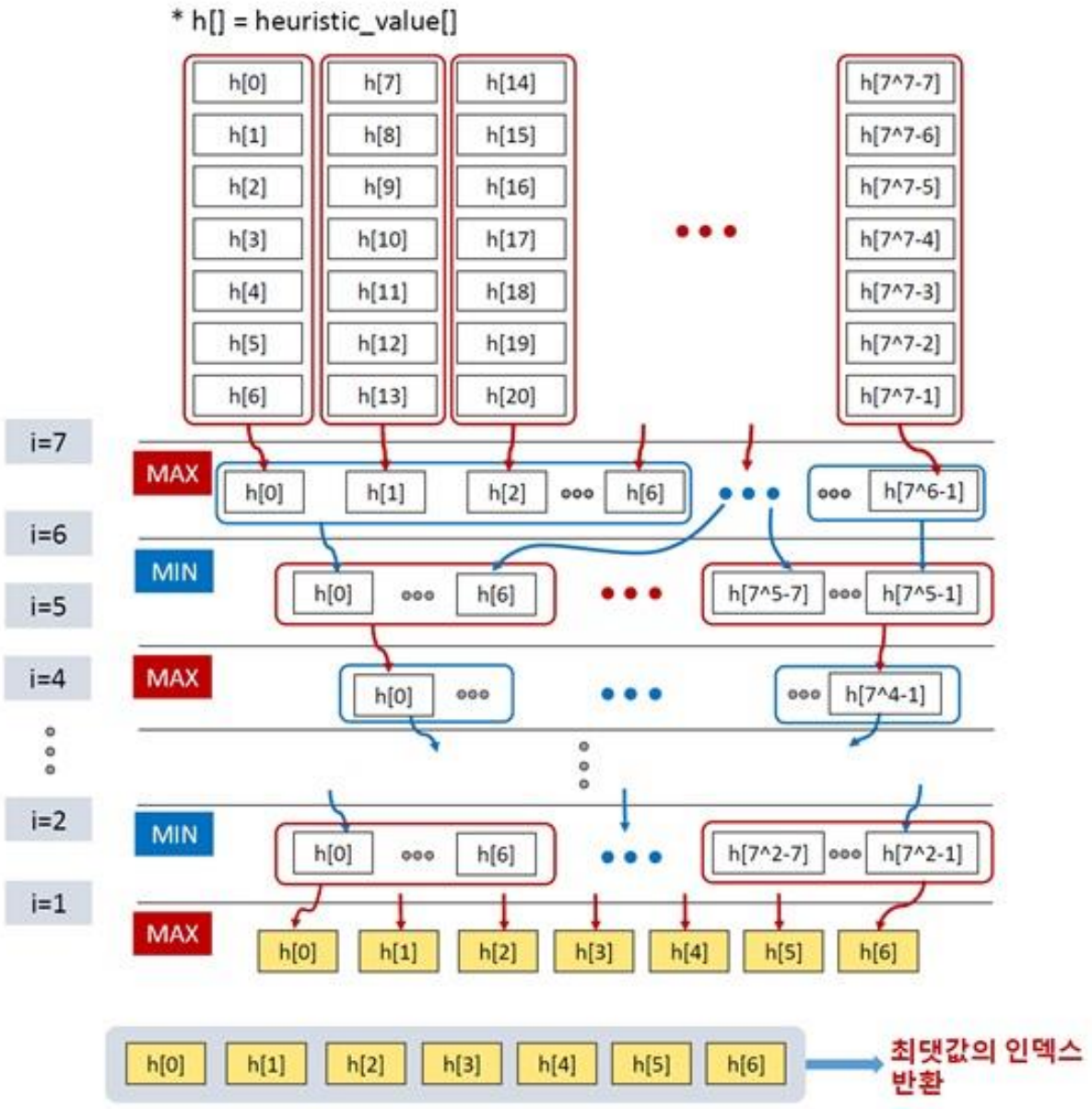


그림의 왼쪽과 같은 경우 내 turn에서 (노란 돌) 체크된 부분을 선점하지 않으면 오른쪽 그림처럼 상대방의 connection완성을 막지 못하게 되는 경우가 발생합니다.

따라서 상대방의 돌 패턴을 분석해 위와 같은 상황이 올 때 해당 column을 선점하게 합니다.

해당 상황에 대한 처리를 수행하는 함수는 rule mode에 더 세밀하게 구현이 되어있지만 Search Algorithm mode의 execution time을 최대한 줄이기 위해 단순하게 패턴 분석이 가능한 가로줄 connection에 대한 처리만 if else문으로 구현했습니다.

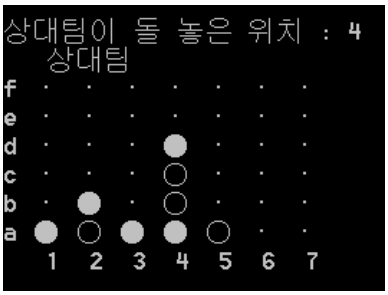
G. Search Algorithm mode의 개략적인 실행모습



H. 시행착오

1) 내가 특정 column을 선택했을 때 상대방이 다음 turn에서 같은 column을 선택하여 승리하게 되는 상황에 대해 해당 column을 선택하지 않도록 하는 기능을 초반엔 losespot_weight라는 함수를 만들어 처리했습니다. 하지만 이 부분은 가중치를 아주 낮게 (-99999) 주어야 하는 부분이기 때문에 heuristic 알고리즘 내에서 따로 체크하는 것이 더 적절하다고 판단해 해당 함수는 사용하지 않게 되었습니다.

```
int losespot_weight(int arr[][7],int x,int turn){
    int i,j,w=0,y=0,mine,enemy;

    //i,x에 말이 있나 없나 확인 없으면 거기다 놓을라고 i,x에 말이있으면 그 위에 두어야 함
    for(i=5;i>=0;i--){
        if(arr[i][x]!=-1){
            y=i+1;
         break;
        }
    }
    //y=6이면 그 줄이 꽉 찬것 더이상 못놓으니까 임파서블
    if(y==6){return impossible;}

    mine=turn%2;
    enemy=(turn+1)%2;

  //y,x에 내가 놓는 상황 만듦
    arr[y][x]=mine;

  //내가 두었는데 4개가 만만들어졌는데
    if(check(arr)!=1){
      //상대가 두니까 4개가 만들어지면 절대 여기두면안됨
    arr[y+1][x]=enemy;
    if(check(arr)==1){
      arr[y][x]=-1;
      arr[y+1][x]=-1;
      return -2000;
    }
  }
    //원래대로 해줌
    arr[y][x]=-1;
    arr[y+1][x]=-1;

    return 0;
}
```



오른쪽 그림은 위에서 서술한 상황을 보여줍니다. 내가 패배할 수 밖에 없는 상황을 막기 위해서는 해당 column을 절대 선택하지 않도록 해야하고 이를 heuristic 함수 내에서 impossible 값을 부여하여 대처하도록 했습니다.

2) 이전 코드에서 상대방이 세 개 돌을 연결해서 수비를 해야하는 상황, 혹은 우리 팀이 세 개의 돌을 연결해서 공격이 가능한 상황은 기존의 heuristic 값을 계산하는 함수로는 가중치가 충분히 주어지지 않아, 따로 체크하여 heuristic_value에 must값을 부여하였습니다.

3) 노드의 데이터 값이 현재 board의 돌 배치 뿐만이 아니라, 각 노드의 heuristic을 저장하려 하였으나 현재 코드는 모든 트리를 구현하는 것이 아닌 fixdepth만큼의 수 앞의 상황을 담은 7^fixdepth개의 state노드만 형성하는 것이기 때문에 이 값에 관한 부분을 저장하기 위해 heuristic_value[]라는 배열을 따로 만들어 사용했고 minmax() 에서의 계산을 용이하게 만들었습니다.

3. Rule Mode

Rule Mode 선택 시에 돌을 어디에 두어야 할지에 대한 판단은 오직 현재 board의 상태를 토대로 해야 합니다. 사람 대 사람으로 게임을 할 때에도 각 선수는 현재 상황에 기반해 수를 결정할 수 밖에 없습니다. 따라서 사람이 착수점을 계산할 때의 판단 기준과 최대한 비슷하게 rule을 구현한다면 rule mode에서의 승률을 높일 수 있습니다. Connect 4 게임을 직접 여러 번 해보면서 수를 둘 때 고려했던 점은 무엇인지, 어떤 사항을 우선적으로 고려했는지 최대한 파악하려고 했고 몇 가지 조건으로 나눠 코드로 구현했습니다.

**threat: 돌을 둔다면 connection을 완성할 수 있는 위치

**useful threat: threat에 해당되는 자리가 맨 밑의 row거나 바로 아래칸이 빈칸이 아니어서 바로 control할 수 있는 threat

**useless threat: threat에 해당되는 자리 바로 아래칸이 비어 있는 상태라 바로 control하지 못하는 (의미 없는) threat

A. 판단 기준과 우선순위

Rule mode에서는 일정 수 앞을 내다볼 수 없기 때문에 각 column에 대한 판단 기준이 Search Algorithm의 heuristic 함수와는 어느 정도 다릅니다. 하지만 최우선적으로 고려되어야 하는 부분, 예를 들어 상대방이 두면 connection을 완성할 수 있는 자리를 선점 해야 한다는 규칙 등은 동일하게 적용됩니다.

Rule mode에서는 아래의 같은 순서로 각 규칙의 조건에 부합하는 column이 있는지 체크하고 있다면 그 값을 return합니다.

1) 선택한다면 내 connection을 완성 할 수 있는 column이 있다면 바로 선택합니다. 그리고 상대방이 선택한다면 상대방의 connection을 완성할 수 있는 column이 있다면 상대방이 승리하지 못하도록 선점합니다.

   ••• case 0

2) 한 칸 위에 상대방의 useless threat 존재하는 column은 다음 turn에서 상대방이 connection을 완성할 수 있으므로 선택하지 않도록 합니다.

3) 조건 2)에서 걸러낸 column에 대하여 상대방이 선택했을 때 useful threat 2개를 생성할 수 있는 column이 있다면 선점합니다. 상대방이 useful threat을 2개 생성한다면 앞으로 두 turn 이내로 무조건 지게 되므로 우선적으로 고려해야 합니다.

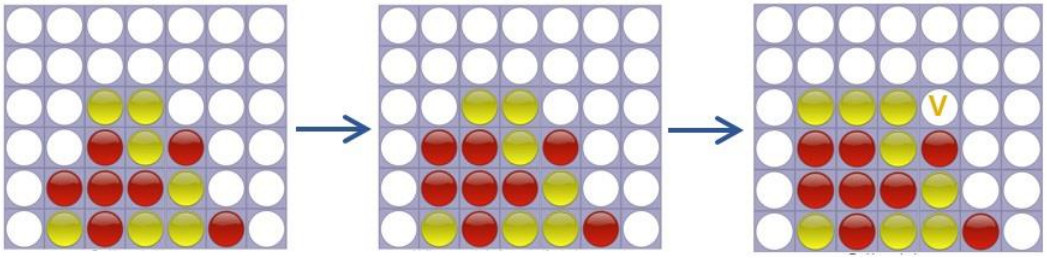
   ••• case 1

4) 조건 2)에서 걸러낸 column에 대하여 useful threat을 만들어 낼 수 있는 column이

있다면 선택합니다. 이는 다음 turn때 상대방으로 하여금 내 connection 완성을 막기 위해 해당 threat을 선점하도록 이끌기 때문에 상대방의 수를 조종할 수 있습니다.

••• case 2, case 3, case 4

5) 상대방의 useful threat 생성을 돕는 column은 선택하지 않도록 합니다.



위 그림에서 노란 돌이 상대방이라고 할 때 내가 두 번째 column을 선택하여 (3,2) 자리에 돌을 두게 된다면 상대방은 그 위, (4,2)자리에 돌을 둘 수 있게 되고 따라서 (4,5)자리에 useful threat을 생성할 수 있게 됩니다. 따라서 위 상황의 두 번째 column과 같은 것은 선택하지 않도록 걸러냅니다.

6) 조건 5)에서 걸러낸 column에 대하여 useless threat을 만들어 낼 수 있는 column이 있다면 선택합니다. 이들을 지금 당장 control할 수는 없지만 게임이 진행되면서 useful threat이 되므로 유리한 것으로 판단하고 선택합니다.

••• case 5, case 6

7) 조건 5)에서 걸러낸 column에 대하여 조건 6)에 부합하는 것이 하나도 없다면 홀수 번째 row에 착수되는 column을 선택하도록 합니다. 이 조건은 유리하거나 불리하지 않은 column이 두 개 이상 존재할 때 대체로 홀수 번째 row에 돌을 두는 것이 이후에 threat 생성, connection 완성에 도움이 되는 것 같아 추가했습니다.

••• case 7

8) 위의 조건에 부합하는 column이 없다면 조건 2)에 해당되지 않는 column에 대해 랜덤으로 선택하고, 남아있는 column이 모두 조건 2)에 해당된다면 돌을 둘 수 있는 column에 대해 랜덤으로 선택합니다.

••• case 8

각 case의 동작과 구현은 아래 **3.C.b**에서 자세하게 설명하도록 하겠습니다.

B. Turn이 3 이하일 경우

착수된 돌의 수가 선수 별 2개 이하일 경우에는 위에서 설명한 고려사항들이 제대로 적

용되지 않아 case7로 넘어가서 column을 랜덤으로 선택하게 됩니다. 이는 상대방에게 유리하게 돌아갈 수 있기 때문에 turn이 3이하의 경우를 따로 빼서 대체로 connection을 완성하기에 유리한 자리를 선점하도록 처리합니다.

C. Turn이 3 이상일 경우

a. 호출 함수

Rule의 조건 판단에는 내 돌 혹은 상대방의 돌을 가정해 useful / useless threat을 계산하는 부분이 반복해서 나오기 때문에 특정 계산을 처리해주는 함수를 따로 정의해 rule함수 내에서 호출하게 하였습니다.

1) *make_threat( )*

column값을 인자로 받아 그 column을 선택한다면 몇 개의 useful threat, useless threat을 생성하는지 계산하는 함수입니다. 인자로 받은 배열을 복사해서 해당 column에 착수하고 (시뮬레이션) *check_threat( )*을 호출합니다. return값으로는 *check_threat( )*의 결과값을 그대로 return합니다.

2) *check_threat( )*

board를 나타내는 배열과 turn을 인자로 받아 배열의 모든 빈칸에 대하여 차례대로 돌을 둬보고 해당 칸이 threat인지 판단하는 함수입니다. 이 함수는 현재 상황에서 turn 차례의 선수가 몇 개의 useful threat과 몇 개의 useless threat을 가지는지 계산하는데, useful threat은 하나당 4점, useless threat은 하나당 1점으로 하고 총 threat value를 return 합니다. 만약 return 값이 5라면 useful, useless threat을 각각 하나씩 가지고 있는 것입니다.

3) *useful( )*

threat이 useful인지 useless인지 판단하기 위해 정의된 함수입니다. row, column을 나타내는 변수와 board의 배열을 인자로 받아 row가 첫 번째 거나 바로 아래 칸이 빈칸이 아니라면 useful이라고 판단합니다.

4) *check_bottom_1( )*

특정 column에 돌을 둔다면 상대방이 connection을 완성할 수 있는지 판단하는 함수입니다. 인자로 받은 column에 내 돌을, 그 바로 위칸에 상대방의 돌을 둬보고 상대방의 connection이 완성되는지 *check( )*함수를 호출해 계산합니다. 조건 2)에 대한 처리를 이 함수가 수행합니다.

5) *check_bottom_2( )*

특정 column에 돌을 둔다면 상대방이 threat을 생성할 수 있는지 판단하는 함수

입니다. *check_botom_1( )*과 같이 전달받은 column에 차례로 내 돌과 상대방 돌을 착수해보고 *check_threat( )*함수를 호출해 몇 개의 useful / useless threat을 생성하는지 계산합니다.

6) *check_row( )*

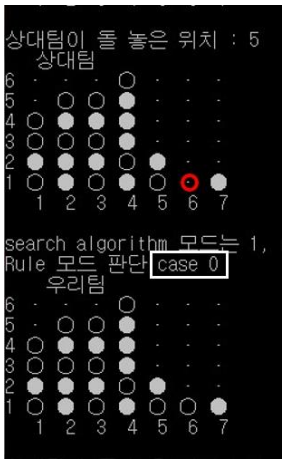
전달받은 column에 돌을 둘 수 있는지, 그렇다면 착수점의 row는 몇 번째인지 계산하는 함수입니다. 만약 돌을 둘 수 없다면 (=해당 column에 빈칸이 존재하지 않는다면) -1을, 둘 수 있다면 착수점의 row index를 return합니다.

b. Case별 처리 (case 0 ~ case 8)

Turn이 3이상일 때 Rule mode에서의 처리는 **3.A**에서 서술한 조건에 따라 좀 더 세부적으로 9가지, case 0 ~ case 8으로 나뉘며 각 번호는 우선순위를 나타냅니다. 각 case의 구현과 동작은 아래와 같습니다.

**실행 화면에서 O O O 는 각각 상대방의 useful threat, 나의 useful threat, 나의 useless threat 지점을 나타냅니다.

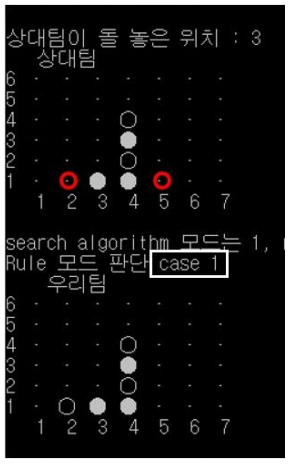
1) case 0



조건 1)에 대한 처리입니다. 첫 번째 column부터 일곱 번째 column까지 차례대로 *win( )*과 *lose( )*를 호출합니다. 만약 특정 column에서 함수가 의미 있는 값(*win( )* : 1000, *lose( )* : 500)을 return한다면 그 column을 선택합니다.

(1,6)은 상대방이 connection을 완성할 수 있는 자리이므로 해당 칸을 선점하여 상대방이 승리하지 못하도록 합니다. (*win( )*의 경우도 마찬가지)

2) case 1

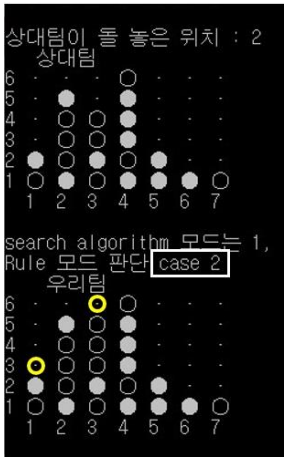


조건 3)에 대한 처리입니다. 우선 돌을 둘 수 없는 column과 조건 2)에 해당 되는 column의 temp_arr에 -100의 값을 줘서 선택하지 않도록 하고 걸러진 column에 대해 차례대로 상대방 turn으로 가정해 *make_threat( )*을 호출합니다. return 값이 8이상이면 useful threat이 2개 생성된 것이므로 이 자리를 선점합니다.

상대방이 (1,2) 또는 (1,5)에 돌을 둔다면 useful threat을 2개 생성하므로 이 자리를 선택해야 합니다. column 1부

터 차례대로 검사하기 때문에 두 번째 column을 착수점으로 선택하게 됩니다.

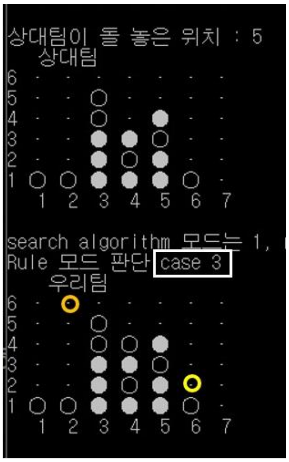
3) case 2



여기서부터 case 4까지는 조건 4)에 대한 처리입니다. 먼저 첫 번째 column부터 차례대로 *make_threat( )*함수를 호출하고 return 값을 비교해 최대값을 threat_value에, 해당 column을 temp_col에 저장합니다. useful threat 2개 생성한다면 (threat_value가 8이상이라면) temp_col을 return합니다.

(5,3)자리에 돌을 둔다면 (6,3)과 (3,1)에 useful threat을 생성하게 되므로 다섯 번째 column을 선택합니다.

4) case 3



case 2와 같습니다. temp_col의 column이 useful threat, useless threat을 각각 하나씩 생성한다면 그것을 return합니다.

오른쪽 첫번째 그림에서 (4,4)를 착수점으로 선택한다면 (6,2)에 useless threat을, (2,6)에 useful threat을 생성할 수 있습니다.

5) case 4

useful threat 하나를 생성할 경우에 대해 처리합니다.

위의 두번째 그림과 같은 상황에서 착수점으로 (3,4)를 선택한다면 (2,5)자리에 useful threat을 하나 만들어 낼 수 있습니다.

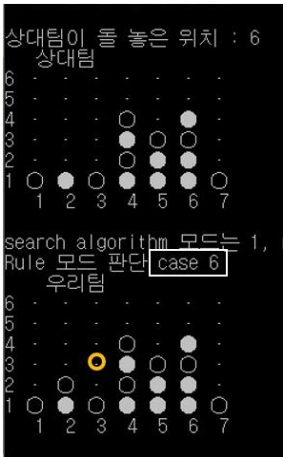
6) case 5

먼저 column마다 *check_bottom_2( )*를 호출해 결과값을 temp_arr에 저장하고 조건 5)에 따라 column을 한번 더 걸러냅니다. 함수의 결과값이 4이상이면 (상대방이 useful threat을 하나 이상 생성할 수 있게 해주는 밑받침 역할을 한다면) 불리한 것으로 판단하고 선택하지 않도록 합니다.

나머지 column에 대해 useless threat을 두 개 이상 생성

하는 column이 있다면 선택합니다.

위 그림에서 (3,5)를 착수점으로 선택한다면 (3,6), (4,6)자리에 각각 useless threat을 만들 수 있습니다.



7) case 6

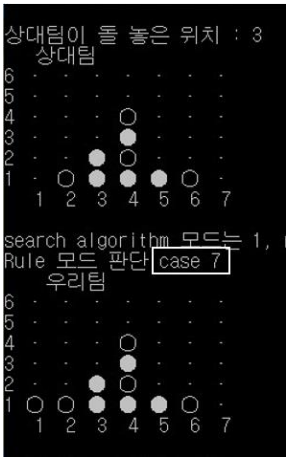
case 5와 동일합니다. 나머지 column에 대해 useless threat을 하나 생성하는 column이 있다면 선택합니다.

(2,2)를 착수점으로 선택함으로써 (3,3)에 useless threat을 생성할 수 있습니다.

8) case 7

나머지 column중 threat을 생성하는 column이 하나도 없다면 홀수 번째 row에 착수되는 column을 선택합니다. 착수점의 row는 *check_row( )*를 호출하여 판단합니다.

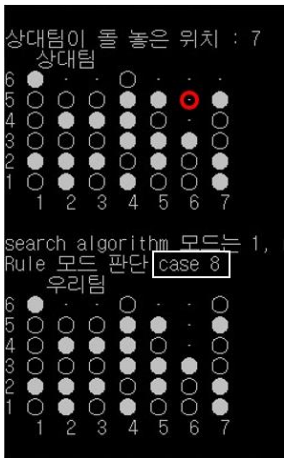
오른쪽 그림과 같은 상황에서는 어떤 column을 선택하나 특별히 유리하거나 불리하지 않습니다. 따라서 홀수 번째 row에 착수되는 column, 여기서는 1, 3, 4, 7에 대해 랜덤하게 선택합니다.

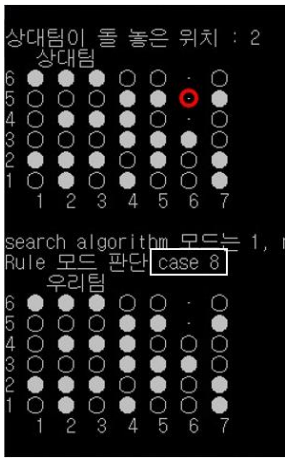


9) case 8

위의 여덟 가지 경우에 부합하는 column이 없다면 랜덤으로 선택하게 되는데 조건2)에 해당하지 않는 column의 유무에 따라 다르게 처리해줍니다.

오른쪽의 첫번째 그림은 조건 2)에 해당하지 않는 column이 존재할 때, 두번째 그림은 존재하지 않을 때에 대한 처리 동작을 나타냅니다. 두 번째 상황에서는 무조건 다음 turn에 지게 되지만 return값을 내야 하므로 어쩔 수 없이 6번째 column을 선택하게 됩니다.





D. 시행착오

1) 처음엔 rule mode를 블랙보드에 올려 주신 중간 프로젝트 관련 문서에 나온 내용

① If there is a winning move, take it

② If your opponent has a winning move, take the move so he can't take it

③ Take the center square over edges and corners

④ Take corner squares over edges

⑤ Take edges if they are the only thing available

을 기반으로 규칙을 세우려고 했습니다. 첫 번째 두 번째 항목에 대한 부분은 최우선적으로 처리해주는게 맞지만 나머지 항목은 유리한 착수점을 결정하는데 거의 도움이 되지 못했고 오히려 상대방에 유리하도록 column을 선택하기도 했습니다.

뒤에 나온 odd, even threat에 관한 내용 또한 구현해봤습니다.

다음 코드에서 *threat_1( )*과 *threat_2( )*는 내 돌을 임시로 둬보고 각각 홀수, 짝수 번째 row에 threat을 생성하는지 체크하는 함수였습니다.

```
//연결된 라인이 가로 라인인 경우
//-->threat을 생성하는 돌의 row와 threat이 위치할 row를 같게 생각하면 됨
for(i=0;i<7;i++){
    if(threat_1(arr, i, turn)>=0{
       if((threat_1(arr, i, turn)%2)==0){
         //제일 아래 라인은 홀수 row지만 index0이므로
         //나눈 값이 0이어야 홀수 row인것임
          return i;
       }
    }
}

//연결된 라인이 양방향 대각선, 세로인 경우
for(i=0;i<7;i++){
    if(threat_2(arr, i, turn)>=0{
       if((threat_1(arr, i, turn)%2)==1){
         //위에 언급한 이유로 홀수 짝수가 뒤바뀌어야 하므로
         //나눈값이 1이어야 odd threat을 생성하는 것임
          return i;
       }
    }
}
```

이때 threat 생성에 대한 체크는 세 개의 돌이 연결되는지 판단했고 이 과정에서 모든 threat 배치 패턴을 파악하지 못하는 문제점이 발생했습니다.

이부분에 대한 내용은 아래에 다시 서술하도록 하겠습니다.

2) 상대방이 3개의 돌이 연속되는 라인을 생성하면 무조건 막도록 하는 rule은 꼭 필요하다고 생각하였었습니다. 그래서 가로, 세로, 양 대각선 방향으로 상대방의 돌이 3개 연속되면 무조건 막도록 했으나 상대방의 돌이 ○, 내 돌이 ●일 때 ○○_○와 같은 상대방의 threat은 파악하지 못하는 문제점이 있었습니다. 따라서 이 경우에 대해 ○○●○ 이렇게 처리해주는 룰을 또다시 만들었습니다. 이렇게 if else문으로 board의 패턴을 토대로 threat을 파악하려고 하다 보니 룰이 너무 복잡해지게 되고 threat

을 파악하는 새로운 방법이 필요했습니다.

아래의 코드는 상대방이 3개의 돌을 연결했을 때 막아주도록 하는 rule을 구현했던 것입니다.

```
//3개 연속이 상대편 무조건 막아줌
 //세로 연속 3개 확 인
 for(i=0;i<=2;i++){
   for(j=0;j<=6;j++){
       if(arr[i][j]==enemy&&arr[i+1][j]==enemy&&arr[i+2][j]==enemy&&arr[i+3][j]==-1) dropstone(j,arr,turn);
   }
 }

 //가로 3개 연속인가 확인 0 0 0 x
 for(i=0;i<=5;i++){
   for(j=0;j<=3;j++){
       if(arr[i][j]==enemy&&arr[i][j+1]==enemy&&arr[i][j+2]==enemy&&arr[i][j+3]==-1) dropstone(j+3,arr,turn);
   }
 }
 //가로 3개 연속인가 확인 x 0 0 0
 for(i=0;i<=5;i++){
   for(j=1;j<=4;j++){
       if(arr[i][j]==enemy&&arr[i][j+1]==enemy&&arr[i][j+2]==enemy&&arr[i][j-1]==-1) dropstone(j-1,arr,turn);
   }
 }
 //슬래시 방향 대각선으로  3 개 연속인가 확인
  for(i=0;i<=2;i++){
   for(j=0;j<=3;j++){
       if(arr[i][j]==enemy&&arr[i+1][j+1]==enemy&&arr[i+2][j+2]==enemy&&arr[i+3][j+3]==-1&&arr[i+2][j+3]!=-1) dropstone(j+3,arr,turn);
   }
  }
  //역슬래시 방향 대각선으로 3개 연속인가 확인
 for(i=0;i<=2;i++){
   for(j=3;j<=6;j++){
       if(arr[i][j]==enemy&&arr[i+1][j-1]==enemy&&arr[i+2][j-2]==enemy&&arr[i+3][j-3]==-1&&arr[i+2][j-3]!=-1) dropstone(j-3,arr,turn);
   }
```

3) 규칙 일부를 작성하고 테스트로 실행해보는 과정에서 게임 초반에는 정의해 놓은 규칙이 거의 적용되지 않는 것을 발견했고 turn이 작을 때 와 클 때를 구분해줘야 한다는 생각이 들었습니다. Turn이 10 이하인 경우엔 세 개 돌의 연속 연결이 완성될 확률은 매우 적고 따라서 상대방의 두 돌 연결을 미리 막아준다든가 내 connection을 만들기 위한 작업을 미리 해주는게 유리하겠다고 생각했습니다.

```
//상대가 연속 두개있으면 그거 막아주게 해주는 룰

//가로 2개 연속인가 확인 0 0 x
for(i=0;i<=5;i++){
  for(j=0;j<=4;j++){
      if(arr[i][j]==enemy&&arr[i][j+1]==enemy&&arr[i][j+2]==-1) dropstone(j+2,arr,turn);
  }
}

//가로 2개 연속인가 확인 x 0 0
for(i=0;i<=5;i++){
  for(j=1;j<=5;j++){
      if(arr[i][j]==enemy&&arr[i][j+1]==enemy&&arr[i][j-1]==-1) dropstone(j-1,arr,turn);
  }
}

//세로 연속 2개 확 인
for(i=0;i<=2;i++){
  for(j=0;j<=6;j++){
      if(arr[i][j]==enemy&&arr[i+1][j]==enemy&&arr[i+2][j]==-1) dropstone(j,arr,turn);
  }
}


//슬래시 방향 대각선으로  2개 연속인가 확인
 for(i=0;i<=2;i++){
  for(j=0;j<=3;j++){
      if(arr[i][j]==enemy&&arr[i+1][j+1]==enemy&&arr[i+2][j+2]==-1&&arr[i+1][j+2]!=-1&&arr[i][j+2]!=-1) dropstone(j+2,arr,turn);
  }
```

```
//상대방 연속 두개가 없는경우 그냥 내 맘위에 두자
for(k=0;k<=6;k++){
    for(i=5;i>=0;i--){
        if(arr[i][k]!=-1){
          y=i+1;
    break;
      }
  }
  if(arr[y-1][k]==mine) {
dropstone(k,arr,turn);
break;
}
```

4) 위와 같이 각 경우에 대해 하나하나 룰을 만들어주다 보니 각 룰도 제대로 처리를 못할뿐더러 너무 복잡해져 우선순위에 대한 판단도 모호해지고 중복되는 부분도 상당히 많았습니다. 그래서 turn 이 3 이하인 부분은 단순하게 매칭 시키는 식으로 만들었고 그 이후의 부분에선 threat을 찾거나 생성하는 기능의 함수를 최대한 구현해 호출하는 식으로 바꾸었습니다. 이렇게 하니 우선순위에 대한 처리도 명확해지고 결과적으로 실행해 보았을 때도 승률이 높아졌습니다.

**조원 별 기여부분

- 2015410055 김연

  Heuristic 값 계산 함수 부분 – winline_weight(), winspot_weight(), impede_weight(), win()

  Rule mode에서 turn이 3 이하일 경우에 대한 처리 부분

  main함수와 interface구현부분, 전체 코드 내용 정리

  구현 부분에 대한 보고서 내용 정리

- 2015410056 김지윤

  ai(), priority()

  Rule mode에서 turn이 3 이상일 경우에 대한 처리 부분

  Rule 보조 함수 – make_threat(), check_threat(), useful(), check_bottom_1,2(), check_row()

  구현 부분에 대한 보고서 내용 정리와 전체 보고서 정리 및 작성

- 2015410100 임수경

  Search algorithm 모드를 위한 자료구조와 MINMAX 알고리즘 구현 부분

  heuristic_function(), for_hu(), minmax()

  구현 부분에 대한 보고서 내용 정리

  나머지 코드 부분은 함께 작성하였습니다.