**Gomoku Implementation applying Iterative Deepening Alpha-Beta Search**

2016320202 이다인

1. 구현 환경

A. 운영 체제 : macOS High Sierra (OS X)

B. 개발 언어 : C++

C. 개발 툴 : terminal

D. README에 적힌 대로 터미널에서 make all로 컴파일, make run으로 실행 가능하다.

2. 코드에 대한 설명

A. stuct board\_node와 struct abp\_node

오목 구현을 위하여 board\_node와 abp\_node 두 가지의 구조체를 선언하여 작성하였다. board\_node는 현재 판의 상태에 관한 정보를 모두 갖고 있는 구조체로, 게임이 끝났는지의 여부를 나타내는 boolean type game\_end, 보드 전체에 몇 칸이 남았는지 기록하는 integer type left\_tiles, 해당 판 상태에서의 heuristic score를 나타내는 integer type heuristic\_score, 판의 어떤 칸에 어떤 말이 놓여있는지 저장하는 character array type의 board\_status, 판의 몇 번째 줄과 몇 번째 열이 세어졌는지 기록하는 boolean vector type의 is\_counted\_row와 is\_counted\_col, 마지막으로 직전 움직임 혹은 진행할 움직임을 저장하는 integer pair type의 moved를 가진다. 이 때 container vector와 pair는 각각 반복자 사용의 편리함과 이차원 배열 인덱스 저장의 명확성을 위해 사용하였다.

abp\_node는 alpha beta pruning에서 사용하기 위해 따로 선언한 구조체로 내부에 해당 움직임으로 움직였을 때의 heuristic score를 기록하는 integer heuristic\_score와 움직일 위치를 나타내는 integer pair type의 move를 가진다. 이 때 container pair 는 board\_node와의 타입 일치 및 역시 이차원 배열 인덱스 저장의 명확성을 위해 사용하였다.

B. abp\_node abp\_minimax (Alpha – Beta Pruning Search with Iterative Deepening)

우선 iter\_deep func을 통해 alpha, beta abp\_node와 결과값을 저장하는 abp\_node인 select의 초기화를 진행한다. 이후 현재 시간을 측정하고, 시간 제한을 넘기기 전까지 Alpha Beta Pruning을 depth 1씩 증가시키며 반복적으로 호출한다. 과제 명세에서 주어진 대로 따로 깊이 제한을 설정하지는 않았으며 시간 제한을 기반으로 작동한다.

기본적으로 Alpha Beta Pruning을 실행하는 func abp\_minimax는 다음의 의사 코드를 기반으로 작성하였다.

01 function alphabeta(node, depth, α, β, maximizingPlayer)

02 if depth = 0 or node is a terminal node

03 return the heuristic value of node

04 if maximizingPlayer

05 v := -∞

06 for each child of node

07 v := max(v, alphabeta(child, depth – 1, α, β, FALSE))

08 α := max(α, v)

09 if β ≤ α

10 break (\* β cut-off \*)

11 return v

12 else

13 v := +∞

14 for each child of node

15 v := min(v, alphabeta(child, depth – 1, α, β, TRUE))

16 β := min(β, v)

17 if β ≤ α

18 break (\* α cut-off \*)

19 return v

(\* Initial call \*)

alphabeta(origin, depth, -[∞](https://en.wikipedia.org/wiki/Infinity), +[∞](https://en.wikipedia.org/wiki/Infinity), TRUE)

우선 시간 제한이 넘어갔을 경우, minimaxflag에 따라 alpha 또는 beta를 반환한다. Depth가 0이거나 남은 타일이 0일 경우, 그대로 해당 board\_node의 heuristic score값과 moved를 반환한다. 그렇지 않은 경우 get\_moves function 으로 생성한 다음 움직임 후보들에 대한 vector 내부의 원소들에 대해서 자기 스스로를 재귀적으로 호출하면서 진행된다. 시간 제한이 넘은 경우 cut하고 alpha 또는 beta 반환, 그 외의 경우 각각에 대해 abp\_minimax를 진행한 temp\_score node에 대해 각각 max 또는 min을 구하여 alpha의 heuristic score가 beta의 heuristic score보다 같거나 클 때 break하고 해당 값을 반환한다.

C. board\_node heuristic\_func (Heuristic Function)

Heuristic function은 기본적으로 전체 판에 대한 검토 기반으로 점수를 측정한다. 우선 빈 칸이 아닌 한 말에 대해서, 세로줄과 가로줄, 윗방향 대각선과 아랫방향 대각선으로 각각 몇 줄이 연속되어 있는지 갯수를 센다. 점수 책정에 대해서는 이 방향에 대해서는 무관히 갯수와 관련하여 진행된다.

|  |  |
| --- | --- |
| 5칸 | 181818 (game\_end true로 설정하고 현재 노드 반환) |
| 4칸 | 한 칸 열려있을 때 Score 2018  두 칸 열려있을 때 Score 2018+2018 (한 칸 열려있을 때에 누적하여 +2018) |
| 3칸 | 한 칸 열려있으나 판 가장 끝부분에 있어 다섯칸까지 진행할 수 없을 때 -118  한 칸 열려있고 진행 할 수 있는 칸이 한 칸(닫힌 삼)일 때 3  한 칸 열려있고 진행 할 수 있는 칸이 두 칸(열린 삼)일 때 돌 놓을 공간이 있으면 218, 돌 놓을 공간이 없으면 -118  그 외의 경우에는 -118 |
| 2칸 | 두 칸과 두 칸 사이에 빈 칸이 있을 때 2018 |

그 외 board에 빈 칸이 없으면 game\_end를 true로 설정하고 노드를 반환한다.

D. 게임 내부의 aux function

1) vector<board\_node> get\_moves

만일 is\_counted\_col과 is\_counted\_row vector가 비어있다면, 즉 빈 판이라면 판의 중간에 말을 놓는다. 그 외의 경우 판의 빈 칸들을 모두 integer pair type의 vector get\_list에 push\_back한다. get\_list의 반복자를 통해 해당 빈 칸에 말이 놓였을 때를 가정한 각 판의 상태를 board\_node로 생성하며, 생성된 board\_node들의 vector를 반환한다.

2) board\_node make\_random\_move

겹치는 칸에 돌이 놓여졌을 때 처리를 위해 만든 함수로, 임의의 빈 행과 열에 말을 업데이트하는 함수이다.

3) board\_node move\_aux

board의 상태와 말의 위치를 입력해주기 위한 함수로 게임이 끝났는지, 보드에 말이 꽉 차 있는지, 올바른 위치에 놓인 돌인지, 중복된 돌은 아닌지에 대한 체크도 함께 진행한다.

3) void print\_aux

판의 상태를 가시적으로 표현하는 함수이다.

4) void game\_start

직접적으로 게임을 시작하는 함수로 사용자에게 놓을 말의 행과 열을 입력 받는다. 또한 한 플레이어의 턴이 끝날 때마다 player\_flag의 반전을 통하여 다음 플레이어의 차례로 변경해준다.

E. 그 외 게임 실행에 대한 부가 사항

1) 쌍삼에 대한 규칙을 구현하려고 했으나, 시간의 부족으로 미처 구현하지 못했다.

2) AI player의 경우 빈 판의 중간에 말을 놓고, 그 다음 AI의 턴에 0, 0에 말을 놓는다. 이는 판의 0, 0에서부터 탐색을 시작하여 나오는 결과로 보인다. 그러나 상대 말이 2칸 이상 말을 연속하게 놓을 때부터 상대편 말을 수비하기 시작하기 때문에 이에 대한 추가적인 구현은 하지 않았다.