**1. 서론**

1.1 연구주제

Minimax algorithm을 활용한 오목 인공지능 모델 개발

1.2 연구목적

미니맥스 알고리즘 모델을 실제로 구현하고 이해해보기 위해서이다.

1.3 연구범위

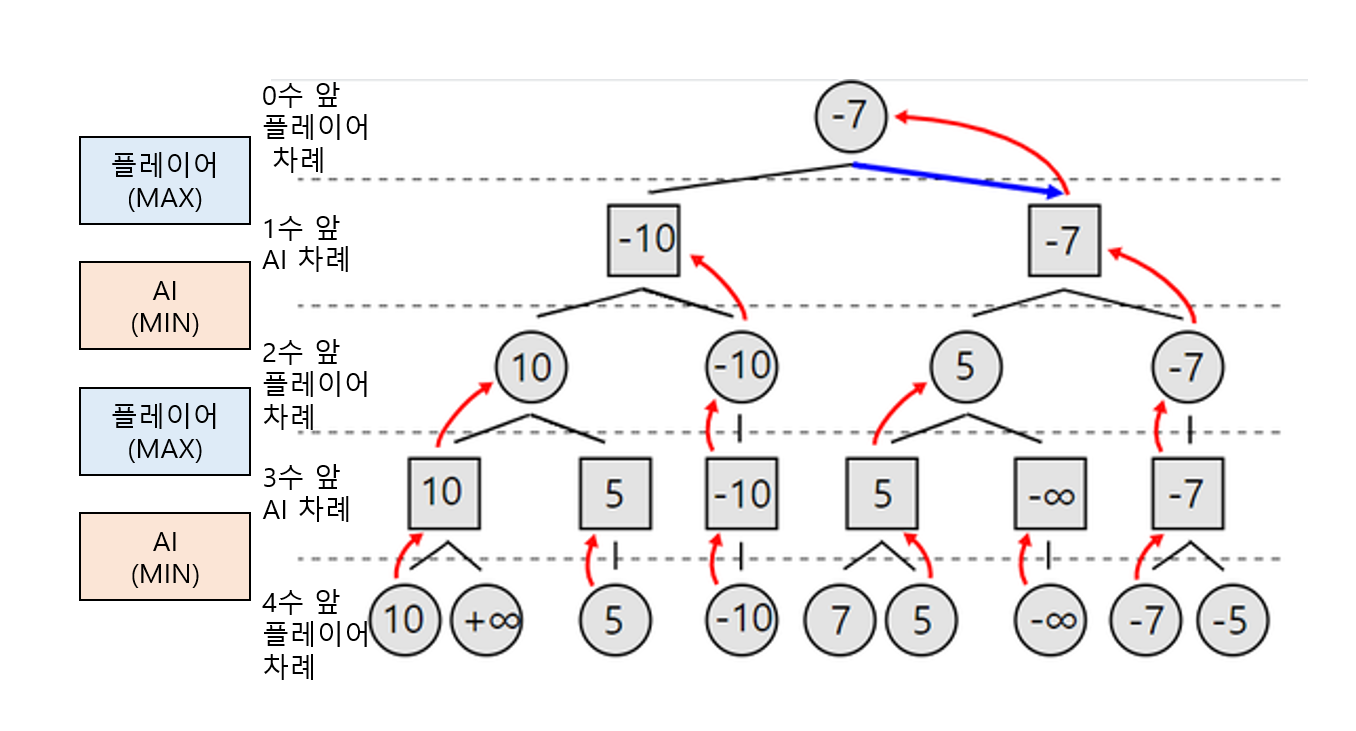
c와 c++을 활용

**2. 연구 수행 내용**

2.1 이론적 배경 및 선행 연구

최소최대 알고리즘은 예상되는 최대의 손실을 최소화하기 위해 사용하는 이론 중 하나다. 최소 최대 원리에 따라 어떤 계획의 성공에 의한 효과를 생각하는 게 아니라, 실패했을 때 어떻게 될지를 생각하여 그 손실이 최소가 되도록 세우는 전략이다.[[1]](http://wiki.hash.kr/index.php/%EC%B5%9C%EC%86%8C%EC%B5%9C%EB%8C%80_%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98#cite_note-1) 바둑, 체스와 같은 두 명의 게임 참여자가 서로 번갈아 행동하거나 동시에 움직이는 경우를 모두 다루는 제로섬 게임 이론으로부터 시작하였으나, 더 복잡한 게임과 불확실성이 존재하는 일반적인 의사결정을 포함해 널리 쓰이고 있다.

최대의 손실을 최소화하기 위해서 모든 경우의 수 중에서 상대는 최적의 수(MAX,상대에게 가장 유리한 수)를 선택한다고 가정하고, AI는 모든 경우의 수 중에서 가장 덜 상대에게 유리한 수(MIN, )를 선택하게 된다.

4수 앞까지 계산하는 그래프를 예시로 들어보자. 각 경우의 수에서 숫자가 클수록(양수) 플레이어에게 유리하고, 숫자가 작을수록 (음수) AI에게 유리한 경우이다. 최대최소알고리즘에서는 우선 4수 앞 경우의 수까지 그래프를 전개하고, 평가함수를 통해 4수 앞의 경우들의 유리한 정도를 평가한다. 그 다음 3수 앞의 경우의 수를 평가하기 위해서는 4수 앞의 경우들(평가해둔 값들)을 사용한다. 3수 앞은 AI의 차례이기 떄문에 AI는 플레이어에게 가장 덜 유리한 수(MIN,값이 가장 작은 경우의 수)를 선택하게 될 것이다. 그 뒤 2수 앞의 경우의 수를 평가하기 위해서는 마찬가지로 3수 앞의 경우들의 값을 사용한다. 2수 앞은 플레이어의 차례이기 때문에 플레이어는 플레이어에게 가장 유리한 경우의 수(MAX, 값이 가장 큰 수)를 선택할 것이다. 1수 앞은 AI의 차례이기 때문에 MIN을 선택하고, 0수 앞은 플레이어의 차례이기에 MAX를 선택한다.

이와 같이 최대최소알고리즘을 적용하면 AI는 가장 피해가 적은 방향으로 게임을 진행할 수 있다.

2.2 연구방법

이전에 개발해둔 모델을 활용하여 코드를 최적화함

2.3 활동 및 과정

게임 환경(오목 판)은 사이즈를 15\*15로 제한하고, 탐색트리가 3수앞까지 보도록 설정하였다. 그리고 오목의 모든 공간을 다 계산하면 탐색량이 대략 (225)^3이 되기에 가장 최근에 놓은 돌에서 거리가 5이하인 곳(5개를 이으면 게임이 끝나기에)만 탐색하도록 제한하고, 오목판을 넘어가거나 이미 돌이 있는 경우에는 더이상 트리를 전개하지 않도록하여 코드를 최적화 했다.

그리고 평가함수 메모이제션 알고리즘을 활용하여서 오목판 전체를 탐색하여 점수를 부여하는 평가함수1과 메모제이션을 활용하여서 부모 노드의 점수 + 가장 최근에 둔 돌 근처의 점수(마찬가지로 거리가 5이하인 곳)으로 점수를 부여하는 평가함수2를 활용하여서 루트 노드에서는 평가함수 1, 나머지 노드에서는 평가함수 2(평가함수1보다 탐색 양이 적음)를 사용하게 하여서 그래프를 탐색하는데 걸리는 시간을 줄였다.

**3. 연구결과**

3.1 이론적 배경 및 선행 연구

#include<bits/stdc++.h>

#include<windows.h>

#include<conio.h>

// 키보드 값

#define UP 0

#define DOWN 1

#define LEFT 2

#define RIGHT 3

#define ENTER 4

using namespace std;

int dx[121] = {0, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -3, -3, -3, -3, -3, -3, -3, -3,

-3, -3, -3, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4,

4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5};

int dy[121] = {0, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, -5, -4

, -3, -2, -1, 0, 1, 2, 3, 4, 5, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, -5, -4, -3,

-2, -1, 0, 1, 2, 3, 4, 5, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, -5, -4, -3, -2, -1

, 0, 1, 2, 3, 4, 5, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};

int board[1742621][16][16]={},cnt=1;

long long int score[1742621], im = -10000000000, AI[10] = {0, 1, 100, 10000, 1000000, 100000000, 100000000, 100000000, 100000000, 100000000};

void input();

int tree(void){

int i, temp = -1, s;

for(i = 120+14401; i <= 1728000+120+14400; i++){

if(score[i/120] == im){

i+=119;

continue;

}

score[i/120] = max(score[i/120], score[i]);

}

for(i = 121; i <= 120+14400; i++){

if(score[i] == im){

continue;

}

score[i/120] = min(score[i/120], score[i]);

}

s = im -1;

for(i = 1; i <= 120; i++){

printf("s[%d] : %d ", i, score[i]);

if(score[i] >= s){

temp = i;

s = score[i];

}

}

printf("\n");

return temp;

}

// 평가 함수 고려해야할 것 : 돌 개수, 최근에 둔 돌 근처에 있는 돌 수

// 평가함수 1(오목 판(15\*15)전체 탐색, root노드에서만 활용)

long long int sc(int index){ //평가함수 플레이어(1) 에게 유리할때는 음수, AI(2)에게 유리할때는 양수

int i, j, s = 0,temp, tf;

for(i = 1; i <= 15; i++){ //AI기준 가로 세로 연결된 돌 고려

temp = 0; tf = 0;

for(j = 1; j <= 15; j++){

if(board[index][i][j] == 2){

temp++;

tf = 1;

}

else{

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

temp = 0; tf = 0;

for(j = 1; j <= 15; j++){

if(board[index][j][i] == 2){

temp++;

tf = 1;

}

else{

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

}

for(i = 1; i <= 15; i++){ //플레이어기준 가로 세로 연결된 돌 고려, 플레이어는 인공지능보다 가중치 2 배

temp = 0; tf = 0;

for(j = 1; j <= 15; j++){

if(board[index][i][j] == 1){

temp++;

tf = 1;

}

else{

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(board[index][i][j] == 2){

s += AI[temp]\*3;

}

if(j-temp-1 > 0){

if(board[index][i][j-temp-1] == 2){

s += AI[temp]\*3;

}

}

}

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s -= 4\*AI[temp];

temp = 0;

tf = 0;

}

temp = 0; tf = 0;

for(j = 1; j <= 15; j++){

if(board[index][j][i] == 1){

temp++;

tf = 1;

}

else{

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(board[index][j][i] == 2){

s += AI[temp]\*3;

}

if(j-temp-1 > 0){

if(board[index][j-temp-1][i] == 2){

s += AI[temp]\*3;

}

}

}

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s -= 2\*AI[temp];

temp = 0;

tf = 0;

}

}

for(i = 1; i <= 15; i++){ //AI기준 대각선 고려(조선 아님)

temp = 0; tf = 0;

for(j = 1; j <= i; j++){

if(board[index][16-j][i+1-j] == 2){

temp++;

tf = 1;

}

else{

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

}

}

temp = 0; tf = 0;

for(j = 1; j <= i; j++){

if(board[index][j][15-i+j] == 2){

temp++;

tf = 1;

}

else{

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

}

}

}

for(i = 1; i <= 15; i++){ //AI기준 대각선 고려(조선 아님)

temp = 0; tf = 0;

for(j = 1; j <= i; j++){

if(board[index][16-j][15-i+j] == 2){

temp++;

tf = 1;

}

else{

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

temp = 0; tf = 0;

for(j = 1; j <= i; j++){

if(board[index][j][i-j+1] == 2){

temp++;

tf = 1;

}

else{

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s += AI[temp];

temp = 0;

tf = 0;

}

}

for(i = 1; i <= 15; i++){ //player기준 대각선 고려(조선 아님)

temp = 0; tf = 0;

for(j = 1; j <= i; j++){

if(board[index][16-j][i+1-j] == 1){

temp++;

tf = 1;

}

else{

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(board[index][16-j][i+1-j] == 2){

s += AI[temp]\*3;

}

if(i+1-j+temp+1 < 15 && 16-j+temp+1 < 15){

if(board[index][16-j+temp+1][i+1-j+temp+1] == 2){

s += AI[temp]\*3;

}

}

}

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s -= AI[temp]\*4;

temp = 0;

tf = 0;

}

temp = 0; tf = 0;

for(j = 1; j <= i; j++){

if(board[index][j][15-i+j] == 1){

temp++;

tf = 1;

}

else{

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(board[index][j][15-i+j] == 2){

s += AI[temp]\*3;

}

if(15-i+j-temp-1 > 0 && j-temp-1 > 0){

if(board[index][j-temp-1][15-i+j-temp-1] == 2){

s += AI[temp]\*3;

}

}

}

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s -= AI[temp]\*4;

temp = 0;

tf = 0;

}

}

for(i = 1; i <= 15; i++){ //player기준 대각선 고려(조선 아님)

temp = 0; tf = 0;

for(j = 1; j <= i; j++){

if(board[index][16-j][15-i+j] == 1){

temp++;

tf = 1;

}

else{

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(board[index][16-j][15-i+j] == 2){

s += AI[temp]\*3;

}

if(15-i+j-temp-1 > 0 && 16-j+temp+1 < 15){

if(board[index][16-j+temp+1][15-i+j-temp-1] == 2){

s += AI[temp]\*3;

}

}

}

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s -= AI[temp]\*4;

temp = 0;

tf = 0;

}

temp = 0; tf = 0;

for(j = 1; j <= i; j++){

if(board[index][j][i-j+1] == 1){

temp++;

tf = 1;

}

else{

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(board[index][j][i+1-j] == 2){

s += AI[temp]\*3;

}

if(i+1-j+temp+1 < 15 && j-temp-1 > 0){

if(board[index][j-temp-1][i+1-j+temp+1] == 2){

s += AI[temp]\*3;

}

}

}

temp = 0;

tf = 0;

}

}

}

if(tf == 1){

s -= AI[temp]\*4;

temp = 0;

tf = 0;

}

}

return s;

}

long long int sc2(int index){ //평가함수 2(memozation을 활용한 경량화 버전, root노드가 아닌 경우에 활용)

int i, j, s = 0,temp, tf;

int x = board[index][0][0];

int y = board[index][0][1];

//평가함수 플레이어(1) 에게 유리할때는 음수, AI(2)에게 유리할때는 양수

////////////////////////////////////////플레이어 기준////////////////////////////////////////

temp = 0; tf = 0;

for(i = -4; i < 5; i++){ //플레이어(1) x+-

if(x+i < 1 || x+i > 15){

continue;

}

if(board[index][x+i][y] == 1){

tf = 1;

temp++;

}

else{

s -= AI[temp]\*4;

if(temp >= 3){

if(x+i-temp-1 >= 1 && board[index][x+i-temp-1][y] == 2){

s += AI[temp]\*5;

}

if(board[index][x+i][y] == 2){

s += AI[temp]\*5;

}

}

tf = 0;

temp = 0;

}

}

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(x+i-temp-1 >= 1 && board[index][x+i-temp-1][y] == 2){

s += AI[temp]\*5;

}

if(x+i <= 15 && board[index][x+i][y] == 2){

s += AI[temp]\*5;

}

}

tf = 0;

temp = 0;

}

temp = 0; tf = 0;

for(i = -4; i < 5; i++){ //플레이어(1) y+-

if(y+i < 1 || y+i > 15){

continue;

}

if(board[index][x][y+i] == 1){

tf = 1;

temp++;

}

else{

s -= AI[temp]\*4;

if(temp >= 3){

if(y+i-temp-1 >= 1 && board[index][x][y+i-temp-1] == 2){

s += AI[temp]\*5;

}

if(board[index][x][y+i] == 2){

s += AI[temp]\*5;

}

}

tf = 0;

temp = 0;

}

}

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(y+i-temp-1 >= 1 && board[index][x][y+i-temp-1] == 2){

s += AI[temp]\*5;

}

if(y+i <= 15 && board[index][x][y+i] == 2){

s += AI[temp]\*5;

}

}

tf = 0;

temp = 0;

}

temp = 0; tf = 0;

for(i = -4; i < 5; i++){ //플레이어(1) x+- y+-

if(y+i < 1 || y+i > 15 || x+i < 1 || x+i > 15){

continue;

}

if(board[index][x+i][y+i] == 1){

tf = 1;

temp++;

}

else{

s -= AI[temp]\*4;

if(temp >= 3){

if(y+i-temp-1 >= 1 && x+i-temp-1 >= 1 && board[index][x+i-temp-1][y+i-temp-1] == 2){

s += AI[temp]\*5;

}

if(board[index][x+i][y+i] == 2){

s += AI[temp]\*5;

}

}

tf = 0;

temp = 0;

}

}

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(y+i-temp-1 >= 1 && x+i-temp-1 >= 1 && board[index][x+i-temp-1][y+i-temp-1] == 2){

s += AI[temp]\*5;

}

if(y+i <= 15 && x+i <= 15 && board[index][x+i][y+i] == 2){

s += AI[temp]\*5;

}

}

tf = 0;

temp = 0;

}

temp = 0; tf = 0;

for(i = -4; i < 5; i++){ //플레이어(1) x+- y+-

if(y+i < 1 || y+i > 15 || x-i < 1 || x-i > 15){

continue;

}

if(board[index][x-i][y+i] == 1){

tf = 1;

temp++;

}

else{

s -= AI[temp]\*4;

if(temp >= 3){

if(y+i-temp-1 >= 1 && x-i+temp+1 >= 1 && board[index][x-i+temp+1][y+i-temp-1] == 2){

s += AI[temp]\*5;

}

if(board[index][x-i][y+i] == 2){

s += AI[temp]\*5;

}

}

tf = 0;

temp = 0;

}

}

if(tf == 1){

s -= AI[temp]\*4;

if(temp >= 3){

if(y+i-temp-1 >= 1 && x-i+temp+1 >= 1 && board[index][x-i+temp+1][y+i-temp-1] == 2){

s += AI[temp]\*5;

}

if(y+i <= 15 && x-i >= 1 && board[index][x-i][y+i] == 2){

s += AI[temp]\*5;

}

}

tf = 0;

temp = 0;

}

////////////////////////////////////////플레이어 기준////////////////////////////////////////

////////////////////////////////////////AI 기준////////////////////////////////////////

temp = 0; tf = 0;

for(i = -4; i < 5; i++){ //플레이어(1) x+-

if(x+i < 1 || x+i > 15){

continue;

}

if(board[index][x+i][y] == 2){

tf = 1;

temp++;

}

else{

s += AI[temp];

if(temp >= 3){

if(x+i-temp-1 >= 1 && board[index][x+i-temp-1][y] == 1){

s -= AI[temp]\*2;

}

if(board[index][x+i][y] == 1){

s -= AI[temp]\*2;

}

}

tf = 0;

temp = 0;

}

}

if(tf == 1){

s += AI[temp];

if(temp >= 3){

if(x+i-temp-1 >= 1 && board[index][x+i-temp-1][y] == 1){

s -= AI[temp]\*2;

}

if(x+i <= 15 && board[index][x+i][y] == 1){

s -= AI[temp]\*2;

}

}

tf = 0;

temp = 0;

}

temp = 0; tf = 0;

for(i = -4; i < 5; i++){ //플레이어(1) y+-

if(y+i < 1 || y+i > 15){

continue;

}

if(board[index][x][y+i] == 2){

tf = 1;

temp++;

}

else{

s += AI[temp];

if(temp >= 3){

if(y+i-temp-1 >= 1 && board[index][x][y+i-temp-1] == 1){

s -= AI[temp]\*2;

}

if(board[index][x][y+i] == 2){

s -= AI[temp]\*2;

}

}

tf = 0;

temp = 0;

}

}

if(tf == 1){

s += AI[temp];

if(temp >= 3){

if(y+i-temp-1 >= 1 && board[index][x][y+i-temp-1] == 1){

s -= AI[temp]\*2;

}

if(y+i <= 15 && board[index][x][y+i] == 1){

s -= AI[temp]\*2;

}

}

tf = 0;

temp = 0;

}

temp = 0; tf = 0;

for(i = -4; i < 5; i++){ //플레이어(1) x+- y+-

if(y+i < 1 || y+i > 15 || x+i < 1 || x+i > 15){

continue;

}

if(board[index][x+i][y+i] == 2){

tf = 1;

temp++;

}

else{

s += AI[temp];

if(temp >= 3){

if(y+i-temp-1 >= 1 && x+i-temp-1 >= 1 && board[index][x+i-temp-1][y+i-temp-1] == 1){

s -= AI[temp]\*3;

}

if(board[index][x+i][y+i] == 1){

s -= AI[temp]\*3;

}

}

tf = 0;

temp = 0;

}

}

if(tf == 1){

s += AI[temp];

if(temp >= 3){

if(y+i-temp-1 >= 1 && x+i-temp-1 >= 1 && board[index][x+i-temp-1][y+i-temp-1] == 1){

s -= AI[temp]\*3;

}

if(y+i <= 15 && x+i <= 15 && board[index][x+i][y+i] == 1){

s -= AI[temp]\*3;

}

}

tf = 0;

temp = 0;

}

temp = 0; tf = 0;

for(i = -4; i < 5; i++){ //플레이어(1) x+- y+-

if(y+i < 1 || y+i > 15 || x-i < 1 || x-i > 15){

continue;

}

if(board[index][x-i][y+i] == 2){

tf = 1;

temp++;

}

else{

s += AI[temp];

if(temp >= 3){

if(y+i-temp-1 >= 1 && x-i+temp+1 >= 1 && board[index][x-i+temp+1][y+i-temp-1] == 1){

s -= AI[temp]\*3;

}

if(board[index][x-i][y+i] == 1){

s -= AI[temp]\*3;

}

}

tf = 0;

temp = 0;

}

}

if(tf == 1){

s += AI[temp];

if(temp >= 3){

if(y+i-temp-1 >= 1 && x-i+temp+1 >= 1 && board[index][x-i+temp+1][y+i-temp-1] == 1){

s -= AI[temp]\*3;

}

if(y+i <= 15 && x-i >= 1 && board[index][x-i][y+i] == 1){

s -= AI[temp]\*3;

}

}

tf = 0;

temp = 0;

}

////////////////////////////////////////AI 기준////////////////////////////////////////

if(board[index][x][y] == 1){

for(j = 0; j < 3; j++){

for(i = 1+44+4+11\*j; i <= 1+44+4+3+11\*j; i++){

if(x+dx[i] < 0 || x+dx[i] > 15 || y+dy[i] < 0 || y+dy[i] > 15){

continue;

}

if(board[index][x+dx[i]][y+dy[i]] != 0){

s += 200;

}

}

}

}

return s;

}

void copy(int a, int b){ //b 보드에 a보드를 복사함

int i, j;

for(i = 1; i <= 15; i++){

for(j = 1; j<= 15;j++){

board[b][i][j] = board[a][i][j];

}

}

}

int key(){ //키 입력 함수

int key;

key=getch();

if(key==72) return UP; // 방향키 위

else if(key==80) return DOWN; // 방향키 아래

else if(key==13) return ENTER; // 엔터키

}

void xy(int x, int y){ // x,y 좌표설정

HANDLE consoleHandle = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD pos;

pos.X = x;

pos.Y = y;

SetConsoleCursorPosition(consoleHandle, pos);

}

void init(){ //크기및 커서 숨기는 설정

system("title OMOK BOT");

system("mode con cols=53 lines=15");

HANDLE consoleHandle = GetStdHandle(STD\_OUTPUT\_HANDLE);

CONSOLE\_CURSOR\_INFO ConsoleCursor;

ConsoleCursor.bVisible = 0;

ConsoleCursor.dwSize = 1;

SetConsoleCursorInfo(consoleHandle,&ConsoleCursor);

}

void titleDraw(){ //타이틀

puts(" ##### # # ##### # # ");

puts(" # # ## ## # # # # ");

puts(" # # # # # # # # # # ");

puts(" # # # # # # # ### ");

puts(" # # # # # # # # ");

puts(" # # # # # # # # ");

puts(" ##### # # ##### # # ");

}

int menuDraw(){ // 선택

int x=23;

int y=10;

xy(x-2,y);

printf("> 게임시작");

xy(x,y+2);

printf(" 종료 ");

while(1){

int n=key();

switch(n){

case UP: {

if(y> 10){

xy(x-2,y);

printf(" ");

xy(x-2,y-=2);

printf(">");

}

break;

}

case DOWN: {

if(y<12){

xy(x-2,y);

printf(" ");

xy(x-2,y+=2);

printf(">");

}

break;

}

case ENTER: {

return y-10;

}

}

}

}

void re(){ //다시하기 입력받는 함수

char a;

while(1){

cin >> a;

if(a=='Y'||a=='y'){

system("cls");

for(int i=0;i<16;i++){

for(int j=0;j<16;j++){

board[0][i][j]=0;

}

}

cnt=1;

puts("0 1 2 3 4 5 6 7 8 9 A B C D E F");

puts("1 - - - - - - - - - - - - - - -");

puts("2 - - - - - - - - - - - - - - -");

puts("3 - - - - - - - - - - - - - - -");

puts("4 - - - - - - - - - - - - - - -");

puts("5 - - - - - - - - - - - - - - -");

puts("6 - - - - - - - - - - - - - - -");

puts("7 - - - - - - - - - - - - - - -");

puts("8 - - - - - - - - - - - - - - -");

puts("9 - - - - - - - - - - - - - - -");

puts("A - - - - - - - - - - - - - - -");

puts("B - - - - - - - - - - - - - - -");

puts("C - - - - - - - - - - - - - - -");

puts("D - - - - - - - - - - - - - - -");

puts("E - - - - - - - - - - - - - - -");

puts("F - - - - - - - - - - - - - - -");

return ;

}

else if(a=='N'||a=='n'){

exit(0);

}

else cout << "잘못된 입력입니다. 다시 입력해주세요: ";

}

}

void input(){ // 플레이

system("mode con cols=200 lines=30"); // 크기 조절

system("cls");

int k=0,t=1,cnt1=0,x,y, temp;

puts("0 1 2 3 4 5 6 7 8 9 A B C D E F");

puts("1 - - - - - - - - - - - - - - -");

puts("2 - - - - - - - - - - - - - - -");

puts("3 - - - - - - - - - - - - - - -");

puts("4 - - - - - - - - - - - - - - -");

puts("5 - - - - - - - - - - - - - - -");

puts("6 - - - - - - - - - - - - - - -");

puts("7 - - - - - - - - - - - - - - -");

puts("8 - - - - - - - - - - - - - - -");

puts("9 - - - - - - - - - - - - - - -");

puts("A - - - - - - - - - - - - - - -");

puts("B - - - - - - - - - - - - - - -");

puts("C - - - - - - - - - - - - - - -");

puts("D - - - - - - - - - - - - - - -");

puts("E - - - - - - - - - - - - - - -");

puts("F - - - - - - - - - - - - - - -");

while(1){

if(cnt%2!=0) cout << "Player 1의 착수위치를 입력해주세요: ";

while(1){

cin >> x >> y;

if(x<0||x>15||y<0||y>15||board[0][x][y]!=0) cout << "잘못된 입력입니다. 다시 입력해주세요: ";

else break;

}

if(cnt%2!=0){

temp = -1;

board[0][x][y]=1; //player1 착수, board[0]은 루트노드

board[0][0][0] = x;

board[0][0][1] = y;

score[0] = sc(0);

if(score[0] < -50000000){

for(int i=0;i<16;i++){

for(int j=0;j<16;j++){

if(i==0){ //가로줄 번호 출력

printf("%X ", k);

k++;

continue;

}

if(j==0){ //세로줄 번호 출력

printf("%X ", t);

t++;

continue;

}

if(board[0][i][j]==0) cout << "- ";

else if(board[0][i][j]==1) cout << "O ";

else if(board[0][i][j]==2) cout << "X ";

}

puts("");

}

printf("player의 승리입니다!");

re();

}

for(int i = 1; i <= 120; i++){ //경우의 수 트리에 저장

copy(0, i);

score[i] = score[0];

if(x+dx[i] < 0 || x+dx[i] > 15 || y+dy[i] < 0 || y+dy[i] > 15){

score[i] = im; //나중에 점수 계산(미니맥스 알고리즘)을 사용할때 논외로 빼놓기 위함

for(int j = 1; j <= 120; j++){

score[i\*120+j] = im;

for(int k = 1; k <= 120; k++){

score[(i\*120+j)\*120+k] = im;

}

}

}

else if(board[i][x + dx[i]][y + dy[i]] != 0){ //이미 돌이 있는 경우

score[i] = im; //나중에 점수 계산(미니맥스 알고리즘)을 사용할때 논외로 빼놓기 위함

for(int j = 1; j <= 120; j++){

score[i\*120+j] = im;

for(int k = 1; k <= 120; k++){

score[(i\*120+j)\*120+k] = im;

}

}

}

else{

board[i][x+dx[i]][y+dy[i]] = 2;

board[i][0][0] = x+dx[i];

board[i][0][1] = y+dy[i];

score[i] += sc2(i);

for(int j = 1; j <= 120; j++){

copy(i,120\*i+j);

score[120\*i+j] = score[i];

if(x+dx[i]+dx[j] < 0 || x+dx[i]+dx[j] > 15 || y+dy[i]+dy[j] < 0 || y+dy[i]+dy[j] > 15){ //board 범위 초과

score[i\*120+j] = im; //나중에 점수 계산(미니맥스 알고리즘)을 사용할때 논외로 빼놓기 위함

for(int k = 1; k <= 120; k++){

score[(i\*120+j)\*120+k] = im;

}

}

else if(board[i\*120+j][x+dx[i]+dx[j]][y+dy[i]+dy[j]] != 0){ //이미 돌이 있는 경우

score[i\*120+j] = im; //나중에 점수 계산(미니맥스 알고리즘)을 사용할때 논외로 빼놓기 위함

for(int k = 1; k <= 120; k++){

score[(i\*120+j)\*120+k] = im;

}

}

else{

board[i\*120+j][x+dx[i]+dx[j]][y+dy[i]+dy[j]] = 1;

board[i\*120+j][0][0] = x+dx[i]+dx[j];

board[i\*120+j][0][1] = y+dy[i]+dy[j];

score[i\*120+j] += sc2(i\*120+j);

for(int k = 1; k <= 120; k++){

copy(i\*120+j, (i\*120+j)\*120+k);

score[(i\*120+j)\*120+k] = score[i\*120+j];

if(x+dx[i]+dx[j]+dx[k] > 15 || x+dx[i]+dx[j]+dx[k] < 0 || y+dy[i]+dy[j]+dy[k] > 15 || y+dy[i]+dy[j]+dy[k] < 0){

score[(i\*120+j)\*120+k] = im; //나중에 점수 계산(미니맥스 알고리즘)을 사용할때 논외로 빼놓기 위함

}

else if(board[(i\*120+j)\*120+k][x+dx[i]+dx[j]+dx[k]][y+dy[i]+dy[j]+dy[k]] != 0){ //이미 돌이 있는 경우

score[(i\*120+j)\*120+k] = im; //나중에 점수 계산(미니맥스 알고리즘)을 사용할때 논외로 빼놓기 위함

}

else{

board[(i\*120+j)\*120+k][x+dx[i]+dx[j]+dx[k]][y+dy[i]+dy[j]+dy[k]] = 2;

board[(i\*120+j)\*120+k][0][0] = x+dx[i]+dx[j]+dx[k];

board[(i\*120+j)\*120+k][0][1] = y+dy[i]+dy[j]+dy[k];

score[(i\*120+j)\*120+k] += sc2((i\*120+j)\*120+k);

}

}

}

}

}

}

}

for(int i = 1; i <= 120; i++){

if(score[i] > 50000000){

temp = i;

}

}

if(temp == -1){

temp = tree();

}

board[0][x+dx[temp]][y+dy[temp]] = 2;

score[0] = sc(0);

cnt++;

printf("temp : %d\n", temp);

for(int i=0;i<16;i++){

for(int j=0;j<16;j++){

if(i==0){ //가로줄 번호 출력

printf("%X ", k);

k++;

continue;

}

if(j==0){ //세로줄 번호 출력

printf("%X ", t);

t++;

continue;

}

if(board[0][i][j]==0) cout << "- ";

else if(board[0][i][j]==1) cout << "O ";

else if(board[0][i][j]==2) cout << "X ";

}

puts("");

}

t=1,k=0;

cnt ++;

if(score[0] > 50000000){

printf("AI의 승리입니다!");

cout << "다시 하시겠습니까? (Y/N) : ";

re();

}

}

}

int main(){

init();

while(1){

titleDraw();

int code= menuDraw();

if(code==0) input();

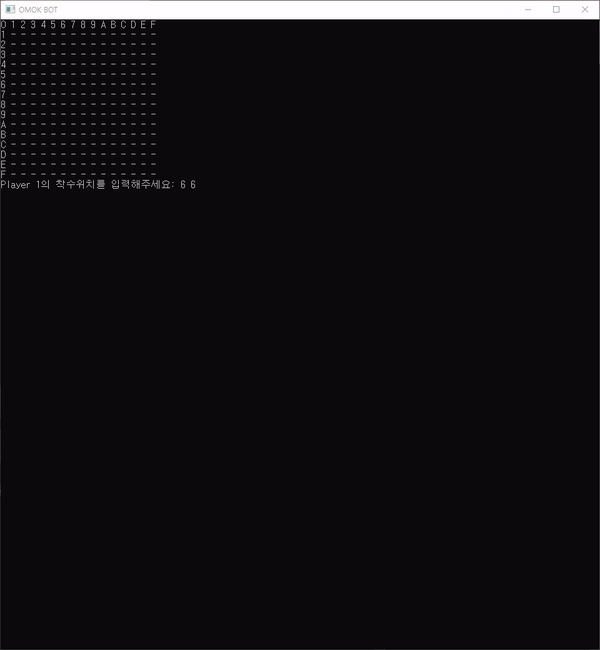
else if(code==1) return 0;

system("cls");

}

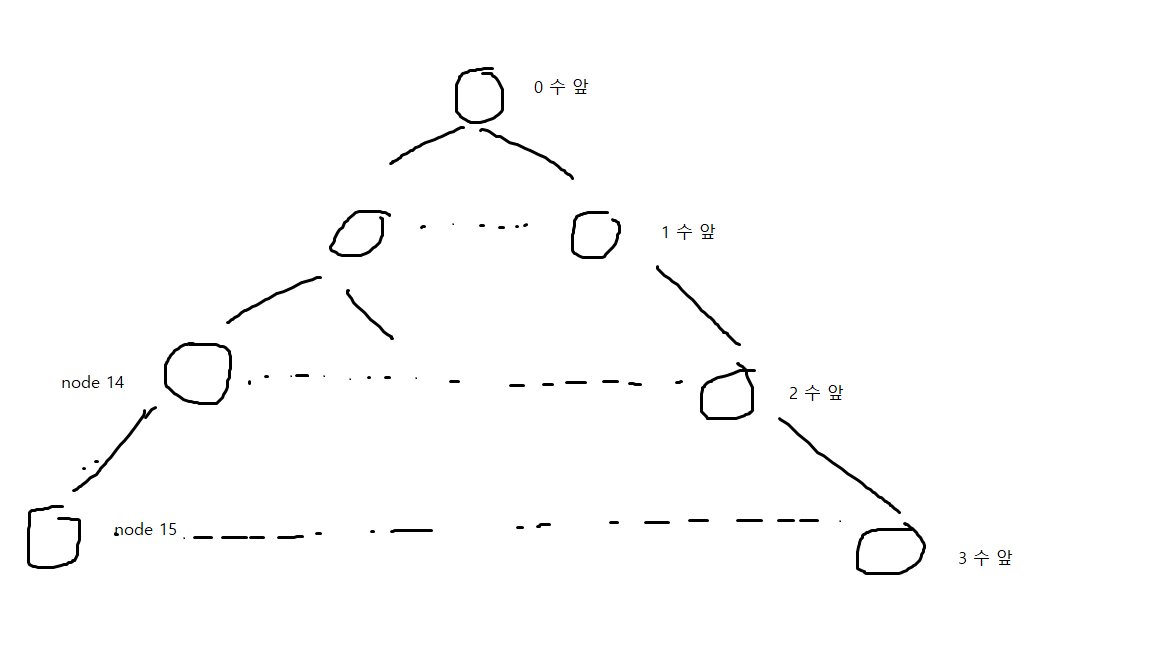
}

**P-1. 많은 탐색량으로 인한 시간 소요**

이전 모델

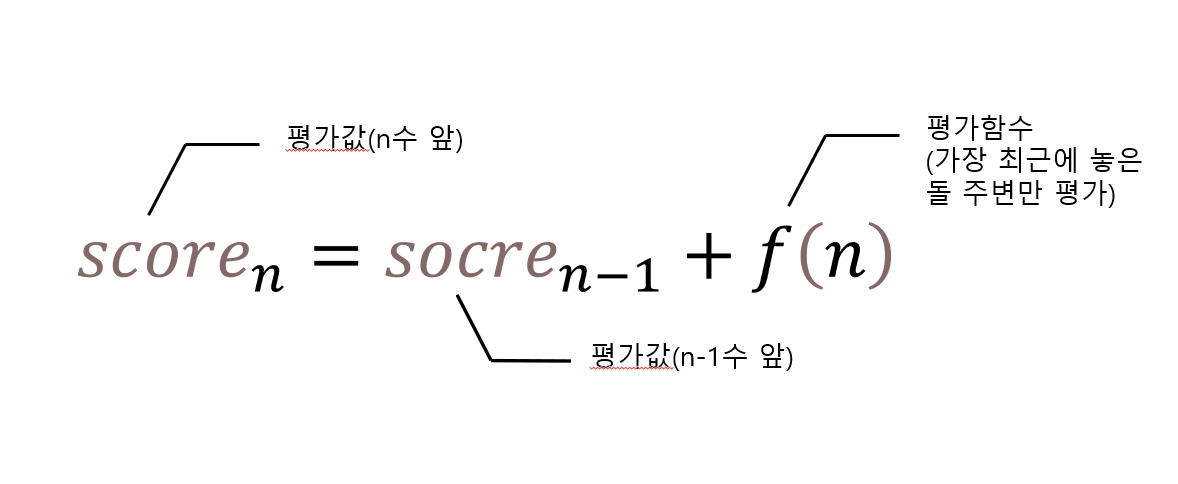
총 1742621개의 오목 판(15\*15)에 다 접근해야 했기에 많은 연산량이 필요했었다.

**S-1. 메모제이션을 활용하여 연산량 최적화**



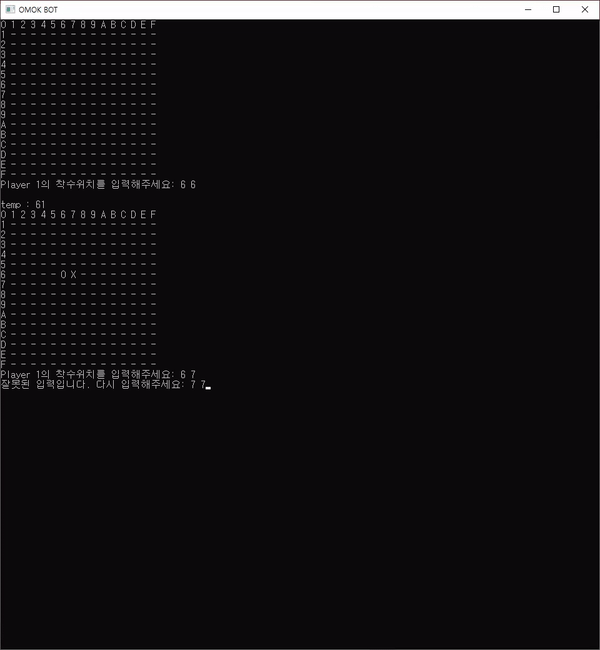
현재 내가 개발한 모델은 왼쪽과 같은 트리를 전개한다. 그런데 트리 모델을 보면 n수 앞의 돌 배치상태와 와 n-1수 앞의 돌 배치 상태가 크게 다르지 않다는 것을 발견했다.

그래서 나(Elizabeth jeong)는 메모제이션을 통해 n-1수 앞의 경우의 평가값과 가장 최근에 놓은 돌의 좌표를 n수 앞의 경우의 바둑판(board)에 저장하여 가장 최근에 둔 돌의 좌표와 거리가 5이하인 곳에서만 평가를 진행하여서



과 같은 방식으로 계산하도록 하였다.

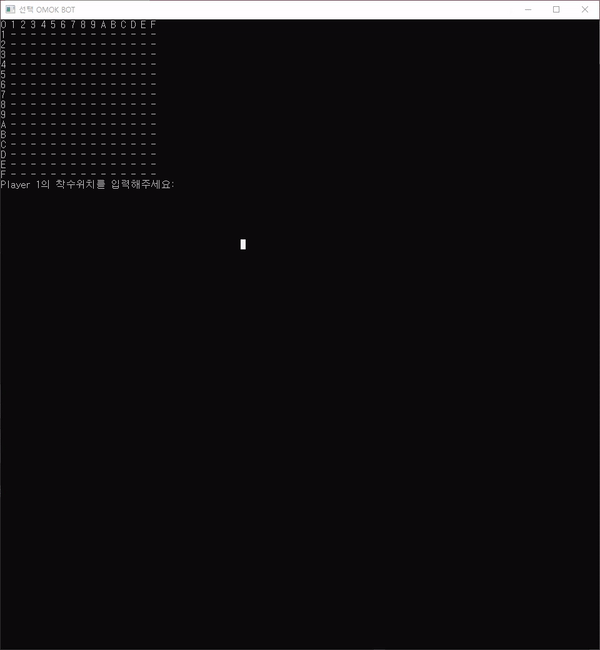
그 결과

****

이처럼 많은 연산량을 줄여 계산에 소요되는 시간을 크게 줄일 수 있었다.

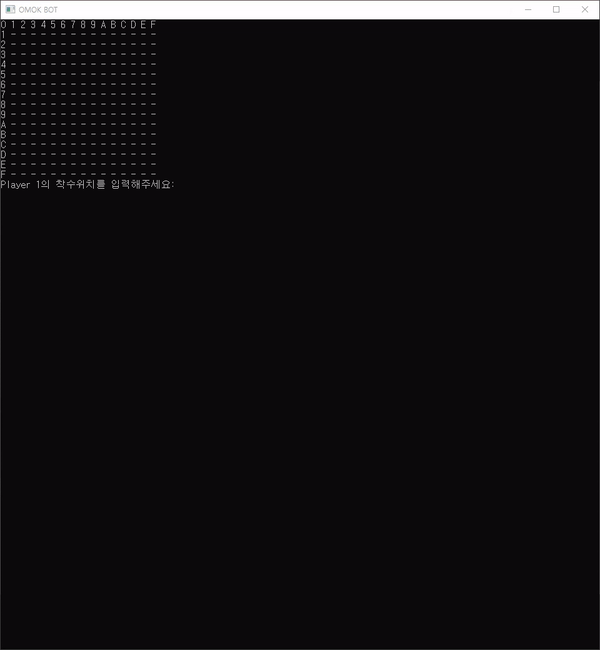
**P-2. 방어를 하지않는 무지성 플레이**

알파텍 2.0 개발을 끝내고 실제로 플레이를 하다보니, AI가 상당히 멍청하다는 것을 깨달았다.

****

이와 같이 방어를 하지않고 오로지 공격에만 치중하는 모습을 보인것이다. 그래서 내(IQ약 200)가 고민해보았을 때 문제점은 평가함수 였다. 알파택 2.0 버전에서의 평가함수는 *AI가 돌을 연속으로 이을 때 부여하는 점수와 플레이어가 돌을 연속으로 이을 때 부여하는 점수* 가 비슷했다. 그래서 플레이어가 돌을 연속으로 이었을 때의 점수가 AI가 돌을 연속으로 이었을 때의 점수보다 크도록 플레이어에게 가중치를 추가하고, AI가 플레이어의 돌 주변에 돌을 두었을 때와 플레이어가 3개 이상 연결한 돌을 막았을 때 추가점수를 주도록 하여서 AI가 조금 더 방어적으로 플레이할 수 있도록 평가함수를 바꾸었다.

그 결과



이처럼 AI가 플레이어의 공격을 방어하며 플레이하는 모습을 보였다.

평가함수를 수정하면서 느낀 점은 최대최소알고리즘에서 평가함수의 중요도가 매우 크다는 것이였다. 평가함수의 가중치가 조금만 바뀌어도 AI의 플레이 방향성이 크게 바뀌었다. 나도 가중치의 값을 여러번 바꾸며 실험해본 뒤에야 정상적으로 플레이하는 평가함수를 찾을 수 있었고, 이마저도 특정한 경우에는 이상적이지 않은 플레이 모습을 보여주었었다.

게다가 오목은 다른 게임에 비해서 간단한 게임임에도 불구하고 평가함수에서 많은 것들을 복잡하게 고려해야했으니, 다른 게임에서 최대최소 알고리즘을 사용하면 평가함수에서 고려해야 할 것들과 가중치 변화에 대한 민감도가 매우 커 적절한 평가함수를 찾는 데 많은 시행착오를 겪어야 할 것 같았다.

**4. 결론 및 제언**

4.1 결론

1. 최대최소알고리즘(MiniMaxAlgorithm)은 평가함수에 의존도가 매우 크다.
2. 경우의 수를 계산할때 n-1수 앞과 n 수 앞은 크게 다르지 않으므로 메모제이션을 활용하면 연산량을 크게 줄일 수 있다.

4.2 제언

없어용

**5. 참고 문헌**

기계는 어떻게 생각하는가? - 숀 게리시(이지스 퍼블리싱)