[ 문제 분석 ]

|  |
| --- |
| 주어진 문제는 오목 프로그램을 구현하는 문제인데, 사용자와 AI가 번갈아 가면서 수를 놓는 오목 프로그램이다. 사용자가 흑돌, AI가 백돌이 되어 사용자가 먼저 흑돌을 두면, 기존의 바둑판의 상태와 사용자가 둔 흑돌의 위치를 종합하여 판단하여 AI가 백돌을 두게 된다. 이렇게 누군가가 오목을 완성하여 승리할 때까지 사용자와 AI는 계속 번갈아가며 수를 둔다. |

[ 프로그램 설계 / 알고리즘 ]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **<사용 자료구조>**   * **Location 클래스** : 특정 좌표에 대한 클래스이다. 행, 열, 상태, 가중치에 대한 정보를 가지고 있다. 좌표의 상태가 비어있으면 condition=-1, 흑돌이 있으면 condition=0, 백돌이 있으면 condition=1이다. 그리고 setWeight 함수는 현재의 가중치에 입력받은 가중치를 더해주는 함수이다. * **GameBoard 클래스**: 바둑판에 대한 상황 판단을 할 때 사용될 함수들이 멤버로 있는 클래스이다.   **< 프로그램 설계 원리 / 알고리즘 >**   * **가중치 설정** : 특정 위치에 있는 돌이 흑돌이면, 그 돌 주변의 가중치가 +2되고, 백돌이면 그 돌 주변의 가중치가 -2된다. 하나의 상황에 대해 예를 들면 다음과 같다.  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 0 | 0 | -2 | -2 | -2 | 0 | | +2 | +2 | 0 | ○ | 0 | 0 | | +2 | ● | ● | ● | -2 | -2 | | +2 | 0 | ○ | -2 | ○ | -2 | | 0 | -2 | -2 | -2 | -2 | -2 |  * **가중치 설정의 경우의 수** :  1. 가장자리가 아닐 때 - 그 돌을 둘러싸는 좌표에 대해 가중치 설정 (8군데) 2. 가장자리 (변) 일 때 - 5군데에 대해 가중치 설정 3. 가장자리 (꼭짓점) 일 때 - 3군데에 대해 가중치 설정      * **연속된 돌들에 대한 조건을 판단할 때**  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | BoardSize | BoardSize | BoardSize | BoardSize | BoardSize | BoardSize | BoardSize | | BoardSize | … | … | … | … | … | BoardSize | | BoardSize | … | (i-1 , j-1) | (i-1 , j) | (i-1 , j+1) | … | BoardSize | | BoardSize | … | (i , j-1) | (i , j) | (i , j+1) | … | BoardSize | | BoardSize | … | (i+1 , j-1) | (i+1 , j) | (i+1 , j+1) | … | BoardSize | | BoardSize | … | … | … | … | … | BoardSize | | BoardSize | BoardSize | BoardSize | BoardSize | BoardSize | BoardSize | BoardSize |   예를 들어, 가로로 5개의 검은 돌이 연속으로 있다는 것을 판단하려면,   |  | | --- | | for (int i = 0; i < BOARD\_SIZE; i++)  for (int j = 0; j < BOARD\_SIZE; j++)  if (point[i][j].getCondition() == 0 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0 && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == 0) |   와 같이 된다.   * **닫힌 4목에 대한 경우의 수:** ( x는 빈자리, ●는 흑돌, ○는 백돌)  1. x●●●●x꼴 : 가로, 세로, 대각선, 반대각선 2. ○●●●●x꼴: 가로, 세로, 대각선, 반대각선 3. x●●●●○꼴: 가로, 세로, 대각선, 반대각선 4. ●●●●x꼴:  |  |  |  |  | | --- | --- | --- | --- | | 왼쪽에서 오른쪽 | 오른쪽에서 왼쪽 | 위에서 아래 | 아래에서 위 | | 왼쪽 위 대각선에서 출발 | 오른쪽 위 대각선에서 출발 | 왼쪽 아래 대각선에서 출발 | 오른쪽 아래 대각선에서 출발 |  * **띈 4목에 대한 경우의 수**  1. ●●x●●꼴: 가로, 세로, 대각선, 반대각선 2. ●●●x●꼴: 가로, 세로, 대각선, 반대각선 3. ●x●●●꼴: 가로, 세로, 대각선, 반대각선  * **띈 3목에 대한 경우의 수**  1. ●x●●꼴: 가로, 세로, 대각선, 반대각선 2. ●●x●꼴: 가로, 세로, 대각선, 반대각선  * **열린 3목에 대한 경우의 수**  1. x●●●x꼴: 가로, 세로, 대각선, 반대각선 2. ○●●●x꼴: 가로, 세로, 대각선, 반대각선 3. x●●●○꼴: 가로, 세로, 대각선, 반대각선 4. ●●●x꼴  |  |  |  |  | | --- | --- | --- | --- | | 왼쪽에서 오른쪽 | 오른쪽에서 왼쪽 | 위에서 아래 | 아래에서 위 | | 왼쪽 위 대각선에서 출발 | 오른쪽 위 대각선에서 출발 | 왼쪽 아래 대각선에서 출발 | 오른쪽 아래 대각선에서 출발 |  * **승리조건**  1. 사용자 승리 : ●●●●● 연속 : 가로, 세로, 대각선, 반대각선 2. AI 승리 : ○○○○○ 연속 : 가로, 세로, 대각선, 반대각선  * **main함수에서의 진행**   - 누군가가 승리할 때까지 무한 루프를 돌면서 사용자, AI가 번갈아가면서 수를 놓음.  - AI가 수를 놓기까지의 상황 판단 : 흑돌의 상황이 닫힌 4목, 띈 4목, 띈 3목, 열린 3목 중에 해당되지 않는다면, 바둑판의 모든 좌표에 대하여 계산한 가중치 중 가장 가중치가 큰 곳에 수를 놓는다. (모든 좌표에 대해 가중치가 가장 큰 좌표인 max와 비교하여 max보다 가중치가 더 크다면 그 좌표가 max가 되고, 그때의 좌표값을 따로 저장하여 착수함수에 파라미터로 넣는다. ) |

[ 소스코드/주석 ]

header.h

|  |
| --- |
| /\* 2018112075 이가현 \*/  #ifndef \_\_header\_h  #define \_\_header\_h  #include <iostream>  using namespace std;  #define BOARD\_SIZE 9  //extern bool stone; //true:흑돌(사용자), false:백돌(AI)  /\*바둑판 상의 위치를 행과 열로 받기 위해 정의한 클래스\*/  class Location {  private:  int row = 0; //행 : i  int col = 0; //열 : j  int condition = 0; //상태 : 비어있음(-1), 흑돌(0), 백돌(1)  int weight = 0; //해당 위치의 가중치, 디폴트 값은 0  public:  Location() { this->row = 0; this->col = 0; }  Location(int col, int row) { this->col = col; this->row = row; }  void set(int col, int row) { this->col = col; this->row = row; }  void setCondition(int condition) { this->condition = condition; }  int getCondition() { return condition; }  int getRow() { return row; }  int getCol() { return col; }  int getWeight() { return weight; }  void setWeight(int weight) { this->weight += weight; } //현재의 가중치에 입력받은 가중치만큼 더해줌  };  /\*바둑판을 관리하기 위해 정의한 클래스\*/  class GameBoard {  public:  //Location USER\_location;  //Location AI\_location;  int check\_closed\_4(); //닫힌 4목 확인 : 1. x●●●●x꼴, 2. ○●●●●x꼴, 3. x●●●●○꼴 ....  int check\_blanked\_4(); //띈 4목 확인 : 1. ●●x●●꼴, 2. ●●●x●꼴, 3. ●x●●●꼴 ....  int check\_blanked\_3(); //띈 3목 확인 : 1. ●x●●꼴, 2. ●●x●꼴 ....  int check\_opened\_3(); //열린 3목 확인 : 1. x●●●x꼴, 2. ○●●●x꼴, 3. x●●●○꼴 ....  bool check\_5(); //승리조건 확인 & 승리 여부 출력(사용자가 승리했는지 AI가 승리했는지 출력)  int getGBdata(Location p); //해당 위치의 가중치 또는 착수 여부 불러오기  };  #endif // !\_\_header\_h |

weight.h

|  |
| --- |
| /\* 2018112075 이가현 \*/  #ifndef \_\_weight\_h  #define \_\_weight\_h  #include "header.h"  Location point[BOARD\_SIZE][BOARD\_SIZE];  /\*  setGBData : 가중치를 set함  getGBData : 가중치를 리턴함  chaksu : 돌을 착수시키고 이를 반영한 오목판을 새로 출력  \*/  void setGBdata() { //모든 위치에 대해 가중치 설정, 갱신  // 1. 가장자리가 아닐 때  for (int i = 1; i < BOARD\_SIZE; i++) { //for문을 돌리면서 : 바둑판 내의 모든 위치 확인  for (int j = 1; j < BOARD\_SIZE; j++) {  //흑돌이면(0) 주변 가중치 +2  if (point[i][j].getCondition() == 0) {  point[i - 1][j - 1].setWeight(2); point[i][j - 1].setWeight(2); point[i + 1][j - 1].setWeight(2);  point[i - 1][j].setWeight(2); point[i + 1][j].setWeight(2);  point[i - 1][j + 1].setWeight(2); point[i][j + 1].setWeight(2); point[i + 1][j + 1].setWeight(2);  }  //백돌이면(1) 주변 가중치 -2  if (point[i][j].getCondition() == 1) {  point[i - 1][j - 1].setWeight(-2); point[i][j - 1].setWeight(-2); point[i + 1][j - 1].setWeight(-2);  point[i - 1][j].setWeight(-2); point[i + 1][j].setWeight(-2);  point[i - 1][j + 1].setWeight(-2); point[i][j + 1].setWeight(-2); point[i + 1][j + 1].setWeight(-2);  }  }  }  // 2. 가장자리(변)일 때  //왼쪽 변  for (int i = 1; i < BOARD\_SIZE; i++) {  if (point[i][0].getCondition() == 0) { //흑  point[i - 1][0].setWeight(2); point[i + 1][0].setWeight(2);  point[i - 1][1].setWeight(2); point[i][1].setWeight(2); point[i + 1][1].setWeight(2);  }  if (point[i][0].getCondition() == 1) { //백  point[i - 1][0].setWeight(-2); point[i + 1][0].setWeight(-2);  point[i - 1][1].setWeight(-2); point[i][1].setWeight(-2); point[i + 1][1].setWeight(-2);  }  }  //오른쪽 변  for (int i = 1; i < BOARD\_SIZE; i++) {  if (point[i][BOARD\_SIZE - 1].getCondition() == 0) { //흑  point[i - 1][BOARD\_SIZE - 1].setWeight(2); point[i + 1][BOARD\_SIZE - 1].setWeight(2);  point[i - 1][BOARD\_SIZE - 2].setWeight(2); point[i][BOARD\_SIZE - 2].setWeight(2); point[i + 1][BOARD\_SIZE - 2].setWeight(2);  }  if (point[i][BOARD\_SIZE - 1].getCondition() == 1) { //백  point[i - 1][BOARD\_SIZE - 1].setWeight(-2); point[i + 1][BOARD\_SIZE - 1].setWeight(-2);  point[i - 1][BOARD\_SIZE - 2].setWeight(-2); point[i][BOARD\_SIZE - 2].setWeight(-2); point[i + 1][BOARD\_SIZE - 2].setWeight(-2);  }  }  //위쪽 변  for (int j = 1; j < BOARD\_SIZE; j++) {  if (point[0][j].getCondition() == 0) { //흑  point[0][j - 1].setWeight(2); point[0][j + 1].setWeight(2);  point[1][j - 1].setWeight(2); point[1][j].setWeight(2); point[1][j + 1].setWeight(2);  }  if (point[0][j].getCondition() == 1) { //백  point[0][j - 1].setWeight(-2); point[0][j + 1].setWeight(-2);  point[1][j - 1].setWeight(-2); point[1][j].setWeight(-2); point[1][j + 1].setWeight(-2);  }  }  //아래쪽 변  for (int j = 1; j < BOARD\_SIZE; j++) {  if (point[BOARD\_SIZE - 1][j].getCondition() == 0) { //흑  point[BOARD\_SIZE - 1][j - 1].setWeight(2); point[BOARD\_SIZE - 1][j + 1].setWeight(2);  point[BOARD\_SIZE - 2][j - 1].setWeight(2); point[BOARD\_SIZE - 2][j].setWeight(2); point[BOARD\_SIZE - 2][j + 1].setWeight(2);  }  if (point[BOARD\_SIZE - 1][j].getCondition() == 1) { //백  point[BOARD\_SIZE - 1][j - 1].setWeight(-2); point[BOARD\_SIZE - 1][j + 1].setWeight(-2);  point[BOARD\_SIZE - 2][j - 1].setWeight(-2); point[BOARD\_SIZE - 2][j].setWeight(-2); point[BOARD\_SIZE - 2][j + 1].setWeight(-2);  }  }  // 3. 가장자리(꼭짓점)일 때  //왼쪽 위  if (point[0][0].getCondition() == 0) { point[1][0].setWeight(2); point[0][1].setWeight(2); point[1][1].setWeight(2); } //흑돌  if (point[0][0].getCondition() == 1) { point[1][0].setWeight(-2); point[0][1].setWeight(-2); point[1][1].setWeight(-2); } //백돌  //오른쪽 위  if (point[0][BOARD\_SIZE - 1].getCondition() == 0) { point[0][BOARD\_SIZE - 2].setWeight(2); point[1][BOARD\_SIZE - 2].setWeight(2); point[1][BOARD\_SIZE - 1].setWeight(2); } //흑돌  if (point[0][BOARD\_SIZE - 1].getCondition() == 1) { point[0][BOARD\_SIZE - 2].setWeight(-2); point[1][BOARD\_SIZE - 2].setWeight(-2); point[1][BOARD\_SIZE - 1].setWeight(-2); }//백돌  //왼쪽 아래  if (point[BOARD\_SIZE - 1][0].getCondition() == 0) { point[BOARD\_SIZE - 2][0].setWeight(2); point[BOARD\_SIZE - 2][1].setWeight(2); point[BOARD\_SIZE - 1][1].setWeight(2); } //흑돌  if (point[BOARD\_SIZE - 1][0].getCondition() == 1) { point[BOARD\_SIZE - 2][0].setWeight(-2); point[BOARD\_SIZE - 2][1].setWeight(-2); point[BOARD\_SIZE - 1][1].setWeight(-2); } //백돌  //오른쪽 아래  if (point[BOARD\_SIZE - 1][BOARD\_SIZE - 1].getCondition() == 0) { point[BOARD\_SIZE - 1][BOARD\_SIZE - 2].setWeight(2); point[BOARD\_SIZE - 2][BOARD\_SIZE - 2].setWeight(2); point[BOARD\_SIZE - 2][BOARD\_SIZE - 1].setWeight(2); } //흑돌  if (point[BOARD\_SIZE - 1][BOARD\_SIZE - 1].getCondition() == 1) { point[BOARD\_SIZE - 1][BOARD\_SIZE - 2].setWeight(-2); point[BOARD\_SIZE - 2][BOARD\_SIZE - 2].setWeight(-2); point[BOARD\_SIZE - 2][BOARD\_SIZE - 1].setWeight(-2); } //백돌  }  int GameBoard::getGBdata(Location p) {  //리턴값 : 비어 있을 경우 해당 위치의 가중치, 돌이 있을 경우 -500을 리턴  int row = p.getRow(); int col = p.getCol();  if (point[row][col].getCondition() == -1) { //해당 위치가 비어 있는 경우 가중치를 리턴  return p.getWeight();  }  else { //해당 위치에 돌이 있으면 -500(매우 작은 수)을 리턴  return -500;  }  }  void chaksu(int x, int y, bool stone) {  //(x,y) 좌표에 착수, stone으로 흑백 판별, 새로운 오목판 출력  if (stone == true) { //흑돌(사용자) 차례이면  point[x][y].set(x, y); //좌표 저장  point[x][y].setCondition(0); //그 좌표 0으로 초기화  }  else { //백돌(AI) 차례이면  point[x][y].set(x, y);//좌표 저장  point[x][y].setCondition(1); //그 좌표 1로 초기화  }  cout << " ";  for (int k = 0; k < BOARD\_SIZE; k++) {  cout << " " << k << " ";  }  cout << endl;  for (int i = 0; i < BOARD\_SIZE; i++) {  cout << i;  for (int j = 0; j < BOARD\_SIZE; j++) {  if (point[i][j].getCondition() == -1) cout << " + "; //격자를 그림  else if (point[i][j].getCondition() == 0) cout << " ●"; //검은돌(사용자) 착수 시 흑돌 그림  else if (point[i][j].getCondition() == 1) cout << " ○"; //흰돌(AI) 착수 시 흰돌 그림  }  cout << endl;  }  setGBdata(); // 모든 위치에 대해 가중치 갱신  }  #endif // ! \_\_weight\_h |

gameboard.h

|  |
| --- |
| /\* 2018112075 이가현 \*/  #ifndef \_\_gameboard\_h  #define \_\_gameboard\_h  #include "header.h"  #include "weight.h"  /\*GameBoard 클래스 구현\*/  /\*검은돌(사용자)의 형태가 어떤지를 기반으로 AI가 취할 행동을 제시 : 판단 & 착수까지 구현\*/  int GameBoard::check\_closed\_4() {//닫힌 4목이 있는지 확인: 세 가지 경우의 수에 대해 다른 숫자 리턴하여 어떤 형태인지 판별  // 1. x●●●●x꼴 -> x자리에 ○넣어야 됨 (앞의 x자리에 넣도록 하자)  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로로 4개 연속인 경우  if (point[i][j].getCondition() == -1 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0  && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == 0 && point[i][j + 5].getCondition() == -1) {  chaksu(i, j, false); return 1;  }  // 2. 세로로 4개 연속인 경우  else if (point[j][i].getCondition() == -1 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == 0  && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == 0 && point[j + 5][i].getCondition() == -1) {  chaksu(j, i, false); return 1;  }  // 3. 대각선으로 4개 연속인 경우  else if (point[i][j].getCondition() == -1 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == 0  && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == 0 && point[i + 5][j + 5].getCondition() == -1) {  chaksu(i, j, false); return 1;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 5; j < BOARD\_SIZE; j++) {  // 4. 반대각선으로 4개 연속인 경우  if (point[i][j].getCondition() == -1 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == 0  && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == 0 && point[i + 5][j - 5].getCondition() == -1) {  chaksu(i, j, false); return 1;  }  }  }  // 2. ○●●●●x꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로로 4개 연속인 경우  if (point[i][j].getCondition() == 1 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0  && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == 0 && point[i][j + 5].getCondition() == -1) {  chaksu(i, j, false); return 2;  }  // 2. 세로로 4개 연속인 경우  else if (point[j][i].getCondition() == 1 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == 0  && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == 0 && point[j + 5][i].getCondition() == -1) {  chaksu(j + 5, i, false); return 2;  }  // 3. 대각선으로 4개 연속인 경우  else if (point[i][j].getCondition() == 1 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == 0  && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == 0 && point[i + 5][j + 5].getCondition() == -1) {  chaksu(i + 5, j + 5, false); return 2;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 5; j < BOARD\_SIZE; j++) {  // 4. 반대각선으로 4개 연속인 경우  if (point[i][j].getCondition() == 1 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == 0  && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == 0 && point[i + 5][j - 5].getCondition() == -1) {  chaksu(i + 5, j - 5, false); return 2;  }  }  }  // 3. x●●●●○꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로로 4개 연속인 경우  if (point[i][j].getCondition() == -1 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0  && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == 0 && point[i][j + 5].getCondition() == 1) {  chaksu(i, j, false); return 3;  }  // 2. 세로로 4개 연속인 경우  else if (point[j][i].getCondition() == -1 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == 0  && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == 0 && point[j + 5][i].getCondition() == 1) {  chaksu(j, i, false); return 3;  }  // 3. 대각선으로 4개 연속인 경우  else if (point[i][j].getCondition() == -1 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == 0  && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == 0 && point[i + 5][j + 5].getCondition() == 1) {  chaksu(i, j, false); return 3;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 5; j < BOARD\_SIZE; j++) {  // 4. 반대각선으로 4개 연속인 경우  if (point[i][j].getCondition() == -1 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == 0  && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == 0 && point[i + 5][j - 5].getCondition() == 1) {  chaksu(i, j, false); return 3;  }  }  }  // 4. ●●●●x꼴 (바둑판의 가장자리에서 시작하는 경우) -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  // 1. 왼쪽에서 오른쪽  if (point[i][0].getCondition() == 0 && point[i][1].getCondition() == 0 && point[i][2].getCondition() == 0  && point[i][3].getCondition() == 0 && point[i][4].getCondition() == -1) {  chaksu(i, 4, false); return 4;  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  // 2. 오른쪽에서 왼쪽  if (point[i][BOARD\_SIZE - 1].getCondition() == 0 && point[i][BOARD\_SIZE - 2].getCondition() == 0 && point[i][BOARD\_SIZE - 3].getCondition() == 0  && point[i][BOARD\_SIZE - 4].getCondition() == 0 && point[i][BOARD\_SIZE - 5].getCondition() == -1) {  chaksu(i, BOARD\_SIZE - 5, false); return 4;  }  }  for (int j = 0; j < BOARD\_SIZE; j++) {  // 3. 위에서 아래  if (point[0][j].getCondition() == 0 && point[1][j].getCondition() == 0 && point[2][j].getCondition() == 0  && point[3][j].getCondition() == 0 && point[4][j].getCondition() == -1) {  chaksu(4, j, false); return 4;  }  }  for (int j = 0; j < BOARD\_SIZE; j++) {  // 4. 아래에서 위  if (point[BOARD\_SIZE - 1][j].getCondition() == 0 && point[BOARD\_SIZE - 2][j].getCondition() == 0  && point[BOARD\_SIZE - 3][j].getCondition() == 0 && point[BOARD\_SIZE - 4][j].getCondition() == 0 && point[BOARD\_SIZE - 5][j].getCondition() == -1) {  chaksu(BOARD\_SIZE - 5, j, false); return 4;  }  }  // 5. 왼쪽 위 대각선에서 출발  if (point[0][0].getCondition() == 0 && point[1][1].getCondition() == 0 && point[2][2].getCondition() == 0  && point[3][3].getCondition() == 0 && point[4][4].getCondition() == -1) {  chaksu(4, 4, false); return 4;  }  // 6. 오른쪽 위 대각선에서 출발  if (point[0][BOARD\_SIZE - 1].getCondition() == 0 && point[1][BOARD\_SIZE - 2].getCondition() == 0  && point[2][BOARD\_SIZE - 3].getCondition() == 0 && point[3][BOARD\_SIZE - 4].getCondition() == 0 && point[4][BOARD\_SIZE - 5].getCondition() == -1) {  chaksu(4, BOARD\_SIZE - 5, false); return 4;  }  // 7. 왼쪽 아래 대각선에서 출발  if (point[BOARD\_SIZE - 1][0].getCondition() == 0 && point[BOARD\_SIZE - 2][1].getCondition() == 0  && point[BOARD\_SIZE - 3][2].getCondition() == 0 && point[BOARD\_SIZE - 4][3].getCondition() == 0 && point[BOARD\_SIZE - 5][4].getCondition() == -1) {  chaksu(BOARD\_SIZE - 5, 4, false); return 4;  }  // 8. 오른쪽 아래 대각선에서 출발  if (point[BOARD\_SIZE - 1][BOARD\_SIZE - 1].getCondition() == 0 && point[BOARD\_SIZE - 2][BOARD\_SIZE - 2].getCondition() == 0  && point[BOARD\_SIZE - 3][BOARD\_SIZE - 3].getCondition() == 0 && point[BOARD\_SIZE - 4][BOARD\_SIZE - 4].getCondition() == 0 && point[BOARD\_SIZE - 5][BOARD\_SIZE - 5].getCondition() == -1) {  chaksu(BOARD\_SIZE - 5, BOARD\_SIZE - 5, false); return 4;  }  return 0;//아무 형태에도 해당되지 않으면 0 리턴  }  int GameBoard::check\_blanked\_4() {//띈 4목이 있는지 : 세 가지 경우의 수에 대해 다른 숫자 리턴하여 어떤 형태인지 판별  /\*검은돌(사용자)의 형태가 어떤지?\*/ //:: x는 비어 있는 것  // 1. ●●x●●꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로  if (point[i][j].getCondition() == 0 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == -1 && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == 0) {  chaksu(i, j + 2, false); return 1;  }  // 2. 세로  else if (point[j][i].getCondition() == 0 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == -1 && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == 0) {  chaksu(j + 2, i, false); return 1;  }  // 3. 대각선  else if (point[i][j].getCondition() == 0 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == -1 && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == 0) {  chaksu(i + 2, j + 2, false); return 1;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 4; j < BOARD\_SIZE; j++) {  // 4. 반대각선  if (point[i][j].getCondition() == 0 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == -1 && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == 0) {  chaksu(i + 2, j - 2, false); return 1;  }  }  }  // 2. ●●●x●꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로  if (point[i][j].getCondition() == 0 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0 && point[i][j + 3].getCondition() == -1 && point[i][j + 4].getCondition() == 0) {  chaksu(i, j + 3, false); return 2;  }  // 2. 세로  if (point[j][i].getCondition() == 0 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == 0 && point[j + 3][i].getCondition() == -1 && point[j + 4][i].getCondition() == 0) {  chaksu(j + 3, i, false); return 2;  }  // 3. 대각선  if (point[i][j].getCondition() == 0 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == 0 && point[i + 3][j + 3].getCondition() == -1 && point[i + 4][j + 4].getCondition() == 0) {  chaksu(i + 3, j + 3, false); return 2;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 4; j < BOARD\_SIZE; j++) {  // 4. 반대각선  if (point[i][j].getCondition() == 0 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == 0 && point[i + 3][j - 3].getCondition() == -1 && point[i + 4][j - 4].getCondition() == 0) {  chaksu(i + 3, j - 3, false); return 2;  }  }  }  // 3. ●x●●●꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로  if (point[i][j].getCondition() == 0 && point[i][j + 1].getCondition() == -1 && point[i][j + 2].getCondition() == 0 && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == 0) {  chaksu(i, j + 1, false); return 3;  }  // 2. 세로  if (point[j][i].getCondition() == 0 && point[j + 1][i].getCondition() == -1 && point[j + 2][i].getCondition() == 0 && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == 0) {  chaksu(j + 1, i, false); return 3;  }  // 3. 대각선  if (point[i][j].getCondition() == 0 && point[i + 1][j + 1].getCondition() == -1 && point[i + 2][j + 2].getCondition() == 0 && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == 0) {  chaksu(i + 1, j + 1, false); return 3;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 4; j < BOARD\_SIZE; j++) {  // 4. 반대각선  if (point[i][j].getCondition() == 0 && point[i + 1][j - 1].getCondition() == -1 && point[i + 2][j - 2].getCondition() == 0 && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == 0) {  chaksu(i + 1, j - 1, false); return 3;  }  }  }  return 0; //아무 형태에도 해당되지 않으면 0 리턴  }  int GameBoard::check\_blanked\_3() {//띈 3목이 있는지 확인 : 두 가지 경우의 수에 대해 다른 숫자 리턴하여 어떤 형태인지 판별  // 1. ●x●●꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로  if (point[i][j].getCondition() == 0 && point[i][j + 1].getCondition() == -1 && point[i][j + 2].getCondition() == 0 && point[i][j + 3].getCondition() == 0) {  chaksu(i, j + 1, false); return 1;  }  // 2. 세로  if (point[j][i].getCondition() == 0 && point[j + 1][i].getCondition() == -1 && point[j + 2][i].getCondition() == 0 && point[j + 3][i].getCondition() == 0) {  chaksu(j + 1, i, false); return 1;  }  // 3. 대각선  if (point[i][j].getCondition() == 0 && point[i + 1][j + 1].getCondition() == -1 && point[i + 2][j + 2].getCondition() == 0 && point[i + 3][j + 3].getCondition() == 0) {  chaksu(i + 1, j + 1, false); return 1;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 3; j < BOARD\_SIZE; j++) {  // 4. 반대각선  if (point[i][j].getCondition() == 0 && point[i + 1][j - 1].getCondition() == -1 && point[i + 2][j - 2].getCondition() == 0 && point[i + 3][j - 3].getCondition() == 0) {  chaksu(i + 1, j - 1, false); return 1;  }  }  }  // 2. ●●x●꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로  if (point[i][j].getCondition() == 0 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == -1 && point[i][j + 3].getCondition() == 0) {  chaksu(i, j + 2, false); return 2;  }  // 2. 세로  else if (point[j][i].getCondition() == 0 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == -1 && point[j + 3][i].getCondition() == 0) {  chaksu(j + 2, i, false); return 2;  }  // 3. 대각선  else if (point[i][j].getCondition() == 0 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == -1 && point[i + 3][j + 3].getCondition() == 0) {  chaksu(i + 2, j + 2, false); return 2;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 3; j < BOARD\_SIZE; j++) {  // 4. 반대각선  if (point[i][j].getCondition() == 0 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == -1 && point[i + 3][j - 3].getCondition() == 0) {  chaksu(i + 2, j - 2, false); return 2;  }  else return 0; //아무 형태에도 해당되지 않으면 0 리턴  }  }  return 0;  }  int GameBoard::check\_opened\_3() {//열린 3목이 있는지 확인  /\*검은돌(사용자)의 형태가 어떤지?\*/  // 1. x●●●x꼴 -> x자리에 ○넣어야 됨 (앞의 x자리에 넣도록 하자)  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로로 4개 연속인 경우  if (point[i][j].getCondition() == -1 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0 && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == -1) {  chaksu(i, j, false); return 1;  }  // 2. 세로로 4개 연속인 경우  else if (point[j][i].getCondition() == -1 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == 0 && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == -1) {  chaksu(j, i, false); return 1;  }  // 3. 대각선으로 4개 연속인 경우  else if (point[i][j].getCondition() == -1 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == 0 && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == -1) {  chaksu(i, j, false); return 1;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 4; j < BOARD\_SIZE; j++) {  // 4. 반대각선으로 4개 연속인 경우  if (point[i][j].getCondition() == -1 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == 0 && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == -1) {  chaksu(i, j, false); return 1;  }  }  }  // 2. ○●●●x꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로로 4개 연속인 경우  if (point[i][j].getCondition() == 1 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0 && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == -1) {  chaksu(i, j + 4, false); return 2;  }  // 2. 세로로 4개 연속인 경우  else if (point[j][i].getCondition() == 1 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == 0 && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == -1) {  chaksu(j + 4, i, false); return 2;  }  // 3. 대각선으로 4개 연속인 경우  else if (point[i][j].getCondition() == 1 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == 0 && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == -1) {  chaksu(i + 4, j + 4, false); return 2;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 4; j < BOARD\_SIZE; j++) {  // 4. 반대각선으로 4개 연속인 경우  if (point[i][j].getCondition() == 1 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == 0 && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == -1) {  chaksu(i + 4, j - 4, false); return 2;  }  }  }  // 3. x●●●○꼴 -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  // 1. 가로로 4개 연속인 경우  if (point[i][j].getCondition() == -1 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0 && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == 1) {  chaksu(i, j, false); return 3;  }  // 2. 세로로 4개 연속인 경우  else if (point[j][i].getCondition() == -1 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == 0 && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == 1) {  chaksu(j, i, false); return 3;  }  // 3. 대각선으로 4개 연속인 경우  else if (point[i][j].getCondition() == -1 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == 0 && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == 1) {  chaksu(i, j, false); return 3;  }  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 4; j < BOARD\_SIZE; j++) {  // 4. 반대각선으로 4개 연속인 경우  if (point[i][j].getCondition() == -1 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == 0 && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == 1) {  chaksu(i, j, false); return 3;  }  }  }  // 4. ●●●x꼴 (바둑판의 가장자리에서 시작하는 경우) -> x자리에 ○넣어야 됨  for (int i = 0; i < BOARD\_SIZE; i++) {  // 1. 왼쪽에서 오른쪽  if (point[i][0].getCondition() == 0 && point[i][1].getCondition() == 0 && point[i][2].getCondition() == 0 && point[i][3].getCondition() == -1) {  chaksu(i, 3, false); return 4;  }  }  for (int i = 0; i < BOARD\_SIZE; i++) {  // 2. 오른쪽에서 왼쪽  if (point[i][BOARD\_SIZE - 1].getCondition() == 0 && point[i][BOARD\_SIZE - 2].getCondition() == 0 && point[i][BOARD\_SIZE - 3].getCondition() == 0 && point[i][BOARD\_SIZE - 4].getCondition() == -1) {  chaksu(i, BOARD\_SIZE - 4, false); return 4;  }  }  for (int j = 0; j < BOARD\_SIZE; j++) {  // 3. 위에서 아래  if (point[0][j].getCondition() == 0 && point[1][j].getCondition() == 0 && point[2][j].getCondition() == 0 && point[3][j].getCondition() == -1) {  chaksu(3, j, false); return 4;  }  }  for (int j = 0; j < BOARD\_SIZE; j++) {  // 4. 아래에서 위  if (point[BOARD\_SIZE - 1][j].getCondition() == 0 && point[BOARD\_SIZE - 2][j].getCondition() == 0 && point[BOARD\_SIZE - 3][j].getCondition() == 0 && point[BOARD\_SIZE - 4][j].getCondition() == -1) {  chaksu(BOARD\_SIZE - 4, j, false); return 4;  }  }  // 5. 왼쪽 위 대각선에서 출발  if (point[0][0].getCondition() == 0 && point[1][1].getCondition() == 0 && point[2][2].getCondition() == 0 && point[3][3].getCondition() == -1) {  chaksu(3, 4, false); return 4;  }  // 6. 오른쪽 위 대각선에서 출발  if (point[0][BOARD\_SIZE - 1].getCondition() == 0 && point[1][BOARD\_SIZE - 2].getCondition() == 0 && point[2][BOARD\_SIZE - 3].getCondition() == 0 && point[3][BOARD\_SIZE - 4].getCondition() == -1) {  chaksu(3, BOARD\_SIZE - 4, false); return 4;  }  // 7. 왼쪽 아래 대각선에서 출발  if (point[BOARD\_SIZE - 1][0].getCondition() == 0 && point[BOARD\_SIZE - 2][1].getCondition() == 0 && point[BOARD\_SIZE - 3][2].getCondition() == 0 && point[BOARD\_SIZE - 4][3].getCondition() == -1) {  chaksu(BOARD\_SIZE - 4, 3, false); return 4;  }  // 8. 오른쪽 아래 대각선에서 출발  if (point[BOARD\_SIZE - 1][BOARD\_SIZE - 1].getCondition() == 0 && point[BOARD\_SIZE - 2][BOARD\_SIZE - 2].getCondition() == 0 && point[BOARD\_SIZE - 3][BOARD\_SIZE - 3].getCondition() == 0 && point[BOARD\_SIZE - 4][BOARD\_SIZE - 4].getCondition() == -1) {  chaksu(BOARD\_SIZE - 4, BOARD\_SIZE - 4, false); return 4;  }  return 0;//아무 형태에도 해당되지 않으면 0 리턴  }  bool GameBoard::check\_5() {//승리조건 확인 & 누가 승리했는지 출력 (사용자가 승리했는지 AI가 승리했는지 출력)  //true : 누군가가 승리함 , false : 승패가 결정나지 않음  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  /\*검은돌(사용자)이 승리할 경우\*/  // 1. 가로로 5개 연속인 경우  if (point[i][j].getCondition() == 0 && point[i][j + 1].getCondition() == 0 && point[i][j + 2].getCondition() == 0  && point[i][j + 3].getCondition() == 0 && point[i][j + 4].getCondition() == 0) {  cout << "PLAYER WINS" << endl; return true;  }  // 2. 세로로 5개 연속인 경우  else if (point[j][i].getCondition() == 0 && point[j + 1][i].getCondition() == 0 && point[j + 2][i].getCondition() == 0  && point[j + 3][i].getCondition() == 0 && point[j + 4][i].getCondition() == 0) {  cout << "PLAYER WINS" << endl; return true;  }  // 3. 대각선으로 5개 연속인 경우  else if (point[i][j].getCondition() == 0 && point[i + 1][j + 1].getCondition() == 0 && point[i + 2][j + 2].getCondition() == 0  && point[i + 3][j + 3].getCondition() == 0 && point[i + 4][j + 4].getCondition() == 0) {  cout << "PLAYER WINS" << endl; return true;  }  /\*AI가 승리할 경우\*/  // 1. 가로로 5개 연속인 경우  else if (point[i][j].getCondition() == 1 && point[i][j + 1].getCondition() == 1 && point[i][j + 2].getCondition() == 1  && point[i][j + 3].getCondition() == 1 && point[i][j + 4].getCondition() == 1) {  cout << "AI WINS" << endl; return true;  }  // 2. 세로로 5개 연속인 경우  else if (point[j][i].getCondition() == 1 && point[j + 1][i].getCondition() == 1 && point[j + 2][i].getCondition() == 1  && point[j + 3][i].getCondition() == 1 && point[j + 4][i].getCondition() == 1) {  cout << "AI WINS" << endl; return true;  }  // 3. 대각선으로 5개 연속인 경우  else if (point[i][j].getCondition() == 0 && point[i + 1][j + 1].getCondition() == 1 && point[i + 2][j + 2].getCondition() == 1  && point[i + 3][j + 3].getCondition() == 1 && point[i + 4][j + 4].getCondition() == 1) {  cout << "AI WINS" << endl; return true;  }  }  }  // 4. 반대각선으로 5개 연속인 경우  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 4; j < BOARD\_SIZE; j++) {  /\*검은돌(사용자)이 승리할 경우\*/  if (point[i][j].getCondition() == 0 && point[i + 1][j - 1].getCondition() == 0 && point[i + 2][j - 2].getCondition() == 0  && point[i + 3][j - 3].getCondition() == 0 && point[i + 4][j - 4].getCondition() == 0) {  cout << "PLAYER WINS" << endl; return true;  }  /\*AI가 승리할 경우\*/  else if (point[i][j].getCondition() == 0 && point[i + 1][j - 1].getCondition() == 1 && point[i + 2][j - 2].getCondition() == 1  && point[i + 3][j - 3].getCondition() == 1 && point[i + 4][j - 4].getCondition() == 1) {  cout << "AI WINS" << endl; return true;  }  }  }  return false;  }  #endif // !\_\_gameboard\_h |

main.cpp

|  |
| --- |
| /\* 2018112075 이가현 \*/  #include "header.h"  #include "weight.h"  #include "gameboard.h"  int main() {  GameBoard USER; // 사용자 객체  GameBoard AI; // AI 객체  int x, y; // x,y좌표  //기본 바둑판을 그리고 시작 : 바둑판 초기화  cout << " ";  for (int k = 0; k < BOARD\_SIZE; k++) {  cout << " " <<k << " ";  }  cout << endl;  for (int i = 0; i < BOARD\_SIZE; i++) {  cout << i;  for (int j = 0; j < BOARD\_SIZE; j++) {  cout << " + ";  }  cout << endl;  }  // 모든 좌표 상태를 - 1로 초기화 (-1 : 비어있음)  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  point[i][j].set(i, j); //x,y좌표 저장  point[i][j].setCondition(-1); //그 좌표 상태 초기화  point[i][j].setWeight(0); //좌표 가중치 초기화  }  }  //누군가가 이길 때까지 계속 반복  while (1) {  if (USER.check\_5() == true || AI.check\_5() == true) {//누군가가 이기면 루프를 빠져나오고 게임은 끝남  break; cout<< endl << "GAME OVER" << endl;  }  // 1. 사용자로부터 흑돌의 위치를 입력받음  bool stone = true;  cout << "사용자(흑돌) : 착수할 위치를 입력하세요>>";  cin >> x >> y; //착수할 x,y좌표를 입력받음  chaksu(x, y, true);  // 2. AI가 위치를 판단하여 흰돌을 놓음  // => 어떤 경우에 해당이 된다면 그 함수 내에서 착수까지 진행될 것임. 그리고 0이 아닌 수를 리턴  // => 아무 경우의 수에도 해당 되지 않을 경우 가중치를 불러와서 가중치가 가장 큰 위치에 착수  stone = false;  if (AI.check\_closed\_4() == 0 && AI.check\_blanked\_4() == 0 && AI.check\_blanked\_3() == 0 && AI.check\_opened\_3() == 0) {//아무 경우의 수에도 해당 되지 않을 경우  int max = AI.getGBdata(point[0][0]); //가중치 최댓값  int weight; //가중치  int x, y; //가중치 최대가 되는 점의 x,y좌표  // 모든 위치의 가중치를 불러와서 비교함  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  weight = AI.getGBdata(point[i][j]);  if (weight > max) {  max = weight;  x = i; y = j;  }  }  }  //그리고 가중치가 가장 큰 위치 선택, 거기에 착수  chaksu(x, y, false);  }  else {  int s;  //s = AI.check\_closed\_4(); s = AI.check\_blanked\_4(); s = AI.check\_blanked\_3(); s = AI.check\_opened\_3();  }  cout << endl;  //이걸 반복  }  return 0;  } |

[ 결과 / 결과분석 ]

|  |
| --- |
| 들어가기 전에 : 위아래로 두개의 바둑판이 있는데, 위쪽 바둑판은 Player이 수를 놓은 후 나타나는 바둑판이고, 아래의 바둑판은 위쪽 바둑판을 인식하여 AI가 수를 놓은 바둑판의 모습이다.  사용자가 수를 놓으면 AI는 가중치를 계산하여 랜덤하게 수를 놓는다 (아래의 ~에 해당하지 않는 이상)    색칠되어 있는 돌이 흑돌(사용자), 속이 비어 있는 돌이 백돌(AI)이다. 좌표에 대해 설명하자면, 현재 흑돌의 좌표는 (2,6)이고, 백돌의 좌표는 (1,5)이다.  닫힌 4목이 있다면 막기    사용자의가 흑돌을 (5,3)에 놓았을 때, 흑돌이 닫힌 4목을 형성하고 AI는 이를 인식하여 5개가 연속되지 않도록 막는다.  띈 4목이 있다면 막기 (case 1. ●●x●●꼴)    원래 ●●xx●인 상태에서 사용자가 흑돌을 (2,4)에 놓아 ●●x●●이 되어, 흑돌이 띈 4목을 형성하고 AI는 이를 인식하여 5개가 연속되지 않도록 막는다.  띈 4목이 있다면 막기 (case 2. ●●●x●꼴)    사용자의가 흑돌을 (3,6)에 놓았을 때, 흑돌이 닫힌 4목을 형성하고 AI는 이를 인식하여 5개가 연속되지 않도록 막는다.  띈 3목이 있다면 막기    사용자의가 흑돌을 (5,6)에 놓았을 때, 흑돌이 대각선으로 띈 3목을 형성하고 AI는 이를 인식하여 4개가 연속되지 않도록 막는다. 이 경우, 흑돌 사이의 안쪽을 막도록 설계했다.  열린 3목이 있다면 막기    사용자의가 흑돌을 (1,3)에 놓았을 때, 열린 3목을 형성하고 AI는 이를 인식하여 4개가 연속되지 않도록 막는다.  5목이 형성되면 누가 이겼는지 출력하고 프로그램을 종료한다.    흑돌이 대각선으로 5목을 완성하여 이겼다. 프로그램은 PLAYER WINS를 출력하고 종료된다. |

[ 소감 ]

|  |
| --- |
| 클래스를 활용하여 게임을 만들 수 있다는 것이 흥미로웠다. gameboard.h를 구현할 때 비슷비슷한 코드가 계속 반복되는 형태가 되어 코드가 길어졌는데, 이 부분을 개선하지 못한 점이 조금 아쉽다. 프로그램 구현이 끝났을 때, 계속 링커 에러가 발생하여 한참 애를 먹었다. 오류창에는 아무런 오류가 안 뜨는데 프로그램 실행은 되지 않아서 심리적으로 힘들었다. 파일 간의 연결 문제인것 같아서 이를 해결하기 위해 원래 gameboard.cpp, weight.cpp 였던 두 파일을 cpp 파일에서 헤더 파일로 바꾼 뒤 서로 include 해주는 방식으로 진행하였다. 그리고 전역 객체인 Location point[BOARD\_SIZE][BOARD\_SIZE];가 문제가 된다고 생각하여 원래 header.h에 있던 이 객체를 weight.h로 옮겼다. 이를 지역 객체로 만들려는 시도도 했지만, 아무래도 전역 객체로 놔두는 것이 사용하기에 더 편할 것 같아서 이 부분은 그대로 두었다. |