C++언어 기말 프로젝트

2017011 이준영

### 문제정의

# 오셀로 게임을 만들고 플레이 할 수 있게 하라.

* 이때, 오셀로 보드의 크기 N은 6~12사이의 동적으로 할당할 수 있게 한다.

# 중간 프로젝트 당시 사용하였던 코드를 확장하여 코드를 작성하라.

(해결방법 가. 장에 나열함. )

# 주어진 인터페이스를 사용하여 객체 지향적인 오셀로 코드를 작성하라.

(해결방법 나. 장에 나열함.)

1. BoardDraw : 말판 기본화면 구성클래스, 좌표에 해당하는 말 표현

2. UserPlay: 사용자가 좌표에 말을 두는 것을 처리, 사용자 입력 처리

3. ComPlay: 컴퓨터가 놓을 수 있는 자리 중 가중치가 가장 높은 자리 찾음

4. BoardCheck: 현재 보드의 점수를 계산

# 오셀로 게임의 기본 정의는 아래와 같다.

* 선 후공을 정한다.
* 일단, 기본 말을 배치한다. 동적으로 크기를 제공하게 되어 있으므로, 보드의 중간을 찾기 위해 보드 길이의 1/2 지점과 그 지점의 +1 지점에서 초기 돌을 배치한다. (4개 배치)
* 배치 이후, 다음과 같은 사이클을 진행한다.
* 1. 선공은 적군 말 앞에 배치한다. 이때, 배치했을 때 최소 1개 이상의 적군 말을 역전시킬 수 있어야 한다.
* 2. 직선, 각 대각선 방향으로 적군 말이 있는지 체크한다.
* 3. 만약, 적군말이 있다면, 그 방향으로 더 나아가서 아군말이 있는지 확인한다.
* 4. 만약 아군말이 그 방향으로 확인된다면, 그 방향으로 둘러싸인 말들을 모두 아군 말로 역전한다. // 그렇지 않다면, 후공으로 넘긴다.
* 5. 체크가 끝나면, 후공으로 넘긴다.
* 6. 후공도 이상의 과정을 반복한다.
* 7. 그런데 이때, 둘 중 한 명이 배치 할 수 없는 상황 (어디든 배치하여도 적군 말을 역전시킬 수 없을 때 등)이 온다면, 배치할 수 없는 측을 패스한다.
* 8. 그런데, 둘 다 둘 수 없다면, 게임을 끝낸다.
* 9. 만약, 보드가 다 차면, 게임을 끝낸다.
* 게임이 끝나면, 누가 가장 많은 말을 두고 있는지에 따라서 승부를 결정한다.

### 해결 방법

1. 중간 프로젝트와의 연관성 명시

|  |  |  |
| --- | --- | --- |
| 중간 | 기말 | 차이점 |
| Board :  Void print  setInput | Board :  PiecesCode\*\* getBoardInfo  setInput | * Print -> getBoardInfo :   중간 프로젝트 당시, 보드를 보여주는 model class인 board에 view역할인 print에서 역할을 추출하여 개선   * setInput은 최대한 유지함:   단, 두 게임의 규정 중 오셀로측이 역전 현상이 존재함을 감안하여 코드가 변경됨. 인터페이스는 동일. |
| Ware :  Bool computer | Pieces :  Bool computer | * Ware-> Pieces : 기물이라는 이름을 최대한 오셀로에 맞게 수정함. * 다만 의미론적으로 배치된 기물이 computer인지 아닌지를 확인해주는 클래스이므로, computer라는 변수는 수정하지 않음. * Board->Ware\*\* // Board-> Pieces\*\* 의 동일한 구조를 통하여 확장하였음. |

1. 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

   자동 생성된 설명주어진 인터페이스의 명시
2. 상세 데이터 객체의 설명과 구현 동작의 설명

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명

1. 기물 역전

* 기물 역전이란, 대상이 되는 좌표의 기물의 reverse()를 호출하여 소유권을 변경하는 행위를 의미한다.

1. `main`함수의 동작 구현
2. <반복 LOOP>
3. 컴퓨터의 선공인지 확인한다.
4. UserPlay를 생성(userPlay)하여 게임을 초기화한다.
5. 게임이 끝나지 않았다면, (userPlay의 getEnded()가 false인 상태)

<반복 LOOP>

1. userPlay를 통하여 턴을 진행한다.

<반복종료 END LOOP>

1. userPlay에서 추출한 게임 결과를 가져온다.
2. 게임 결과를 보여준다.
3. 게임의 승자를 판단한다.
4. 게임의 승자는 게임 결과의 컴퓨터 점수와 유저 점수를 통하여 판단한다.
5. 보드를 보여준다.
6. 이후 게임을 삭제하고, 게임을 다시 시작할 지 묻는다.
7. 만약, n 을 입력한다면 break를 통해 반복을 나간다.

<반복종료 END LOOP>

1. 별첨

* (사진 1)기물 역전의 판단 범위

이하의 그림과 같이, 배치 이후 대각선/직선으로 판단한다.

다만, 아래의 그림은 방향만 보여주는 것이며, x,y는 보드가 허락하는 한 I,j방향으로 증가한다고 가정한다.

텍스트, 사각형, 스크린샷, 직사각형이(가) 표시된 사진

자동 생성된 설명

* (별첨 2)기물 역전의 판단 알고리즘

기물역전은 (별첨 1)의 방향으로 아래 그림과 같은 알고리즘으로 판단한다.

텍스트, 라인, 도표, 평행이(가) 표시된 사진

자동 생성된 설명

### 소스코드

|  |
| --- |
| //Board.cpp  #include "Board.h"  #include "Pieces.h"  #include <iostream>  #include "../../enums/PlayerGameResults.h"  #include "../../enums/PiecesCode.h"  #include "../../enums/InputErrorCode.h"  #include <vector>  #include "../data/DataActions.h"  using namespace std;  /\*\*  \* @brief 보드를 정의함.  \*  \*/  Board::Board(int size)  {  this->size = size;  this->board = new Pieces\*\* [size];  for(int a = 0 ; a < size ; a ++)  {  this->board[a] = new Pieces\* [size];  for (int b=0;b<size;b++)  {  this->board[a][b] = NULL;  }  }  }  Board::~Board()  {  for (int a = 0 ; a < this->size ; a ++)  {  delete [] this->board[a];  }  delete [] board;  }  Board::Board(const Board& \_board)  {  // 이것은 깊은 복사문제로 인하여 발생함.  this->size = \_board.size;  this->board = new Pieces\*\* [this->size];  for(int a = 0 ; a < this->size ; a ++)  {  this->board[a] = new Pieces\* [this->size];  for (int b=0;b<this->size;b++)  {  if(\_board.board[a][b] != NULL) this->board[a][b] = new Pieces(\_board.board[a][b]); // 여기도 복사!  else this->board[a][b] = NULL;  }  }  }  PiecesCode\*\* Board::getBoardInfo()  {  PiecesCode\*\* result = new PiecesCode\* [this->size];  for(int a = 0 ; a < this->size ; a ++)  {  result[a] = new PiecesCode [this->size];  for (int b=0;b<this->size;b++)  {  result[a][b] = this->board[a][b] == NULL ? PiecesCode::EMPTY :  this->board[a][b]->getComputer() == true ? PiecesCode::COMPUTER : PiecesCode::USER;  }  }  return result;  }  /\*\*  \* @brief board에 유저 혹은 컴퓨터의 말을 설정함.  \*  \* @param \_isComputer  \* @param x  \* @param y  \* @return int  \*/  InputErrorCode Board::setInput(bool \_isComputer,int x,int y,bool isInit)  {  if(x < 0)  return InputErrorCode::INVALID\_INPUT\_X\_0;  else if(x > this->size)  return InputErrorCode::INVALID\_INPUT\_X\_MAX;  else if(y < 0)  return InputErrorCode::INVALID\_INPUT\_Y\_0;  else if(y > this->size)  return InputErrorCode::INVALID\_INPUT\_Y\_MAX;  int allocateX = x,allocateY = y;    if(this->board[allocateX][allocateY] != NULL)  return InputErrorCode::INVALID\_INPUT\_THERE\_IS\_ALREADY;  bool canSet = false;  // If enemy is computer, then "enemy" is user. so value is false.  // If enemy is user, then "enemy" is computer. so value is true.  if(isInit == false && this->searchCanSetInThisPoint(allocateX,allocateY,\_isComputer) == false) {  return InputErrorCode::INVALID\_INOUT\_THERE\_IS\_NO\_PIECE\_TO\_REVERSE;  }  this->board[allocateX][allocateY] = new Pieces(\_isComputer);  this->updateBoard(allocateX,allocateY,\_isComputer);  return InputErrorCode::VALID\_INPUT;  }  InputErrorCode Board::setInput(bool \_isComputer,int x,int y)  {  return this->setInput(\_isComputer,x,y,false);  }  void Board::updateBoard(int x,int y,bool isComputer)  {  for (int i=-1;i<=1;i++){  for (int j=-1;j<=1;j++)  {  if(i == 0 && j == 0) continue;  bool catchingEnemy = false;  int fx = x,fy = y;  while(true)  {  fx += i;  fy += j;  if(fx < 0 || fx > this->size-1 || fy < 0 || fy > this->size-1) break;  if(this->board[fx][fy] == NULL) break;  if(isComputer != this->board[fx][fy]->getComputer()) continue;  else  {  catchingEnemy = true;  break;  }  }  if(catchingEnemy == true){  while(true)  {  fx -= i;  fy -= j;  if(x==fx && y==fy) break;  if(fx < 0 || fx > this->size-1 || fy < 0 || fy > this->size-1) break;  if(this->board[fx][fy] == NULL) break;  if(isComputer != this->board[fx][fy]->getComputer()) {  this->board[fx][fy]->reverse();  }  else break;  }  }  }  }  }  PiecesCode Board::getPieceCode(int r,int c)  {  return this->board[r][c] == NULL ? PiecesCode::EMPTY :  this->board[r][c]->getComputer() == true ? PiecesCode::COMPUTER : PiecesCode::USER;  }  bool Board::searchCanSetInThisPoint(int x,int y,bool isComputer)  {  if(this->board[x][y] != NULL) return false;  for (int i=-1;i<=1;i++){  for (int j=-1;j<=1;j++)  {  if(i == 0 && j == 0) continue;  bool catchingEnemy = false;  int fx = x,fy = y;  while(true)  {  fx += i;  fy += j;  if(fx < 0 || fx > this->size-1 || fy < 0 || fy > this->size-1) break;  if(this->board[fx][fy] == NULL) break;  if(this->board[fx][fy]->getComputer() != isComputer) {  catchingEnemy = true;  continue;  } else if (this->board[fx][fy]->getComputer() == isComputer && catchingEnemy == true ) {  return true; // 최소 1회이상 조우하였으므로 둘 수 있음.  } else if (this->board[fx][fy]->getComputer() == isComputer) break; // 조우한 바 없으므로 둘 수 없음.    }  }  }  return false;  }  int Board::searchCanSetThisBoardCount(int x,int y,bool isComputer)  {  int result = 0;  for (int i=-1;i<=1;i++){  for (int j=-1;j<=1;j++)  {  if(i == 0 && j == 0) continue;  bool catchingEnemy = false;  int fx = x,fy = y;  while(true)  {  fx += i;  fy += j;  if(fx < 0 || fx > this->size-1 || fy < 0 || fy > this->size-1) break;  if(this->board[fx][fy] == NULL) break;  if(this->board[fx][fy]->getComputer() != isComputer) {  catchingEnemy = true;  continue;  } else if (this->board[fx][fy]->getComputer() == isComputer && catchingEnemy == true ) {  return true; // 최소 1회이상 조우하였으므로 둘 수 있음.  } else if (this->board[fx][fy]->getComputer() == isComputer) break; // 조우한 바 없으므로 둘 수 없음.  }  if(catchingEnemy == true){  while(true)  {  fx -= i;  fy -= j;  if(x==fx && y==fy) break;  if(fx < 0 || fx >= this->size-1 || fy < 0 || fy >= this->size-1) break;  if(this->board[fx][fy] == NULL) break;  if(this->board[fx][fy]->getComputer() != isComputer) {  result++;  } else if(this->board[fx][fy]->getComputer() == isComputer) {  break;  }  }  }  }  }  return result;  }  int Board::getSize()  {  return this->size;  } |

Board.h

|  |
| --- |
| #ifndef BOARD\_H  #define BOARD\_H  #include "./Pieces.h"  #include "../../enums/PlayerGameResults.h"  #include "../../enums/PiecesCode.h"  #include "../../enums/InputErrorCode.h"  using namespace std;  /\*\*  \* @brief 보드를 정의함.  \* (Board를 승계하여 N으로 확장할 수 있게 함.)  \*/  class Board  {  private:  Pieces \*\*\* board; // 삼중 포인터인 이유는 동적 크기 배열 할당을 위하여 사용하는 것임.  // 삼중...  int size; // 보드의 크기.  bool \_flag\_changed\_board\_before\_get\_code; // 보드가 변경되었는지 여부를 나타냄.  void updateBoard(int x,int y,bool isComputer); // 보드를 업데이트함.  public:  Board(int \_size); // 초기화임.  ~Board(); // 삭제자임.  Board(const Board& \_board); // 복사 생성자임.:: 내부값 복사용도임  PiecesCode\*\* getBoardInfo(); // 보드 정보를 가져옴(print 대체. 개방-폐쇄 원칙)  /\*\*  \* @brief board에 유저 혹은 컴퓨터의 기물을 설정함.  \*  \* @param \_isComputer 컴퓨터인지 여부  \* @param x x좌표  \* @param y y좌표  \* @return int error code : 1이면 정상임.  \*/  InputErrorCode setInput(bool \_isComputer,int x,int y,bool isInit); // 보드에 기물을 설정함.  InputErrorCode setInput(bool \_isComputer,int x,int y); // 보드에 기물을 설정함.  /\*\*  \* @brief 기물의 소유권을 반환합니다.  \* 1은 유저, -1은 컴퓨터, 0은 없음입니다.  \* @param x  \* @param y  \* @return PiecesCode  \*/  PiecesCode getPieceCode(int r,int c);  /\*\*  \* @brief 이 보드에 추가로 기물을 둘 수 있는지 확인합니다.  \*  \* @return int 추가로 배치 가능한 기물의 수입니다.  \*/  bool searchCanSetInThisPoint(int x,int y,bool isComputer);  int searchCanSetThisBoardCount(int x,int y,bool isComputer);  int getSize(); // 보드의 크기를 반환함.  };  #endif // BOARD\_H  // Path: ./Board.cpp |

DataActions.h

|  |
| --- |
| #ifndef DATAACTIONS\_H\_  #define DATAACTIONS\_H\_  /\*\*  \* @brief  \* AI가 어디에 선택하는것이 좋을지를 결정해주기 위해서, 준비해주는 구조체.  \*/  struct DataActions  {  int x;  int y;  double score;  DataActions(int \_x, int \_y, int \_score) : x(\_x), y(\_y), score(\_score) {}  };  #endif // DATAACTIONS\_H\_ |

GameResult.h

|  |
| --- |
| #ifndef GAMERESULT\_H\_  #define GAMERESULT\_H\_  struct GameResult  {  int userScore;  int computerScore;  bool isGameEnd;  GameResult(int \_userScore, int \_computerScore, bool \_isGameEnd) : userScore(\_userScore), computerScore(\_computerScore), isGameEnd(\_isGameEnd) {}  };  #endif // GAMERESULT\_H\_ |

Pieces.h

|  |
| --- |
| #ifndef PIECES\_H  #define PIECES\_H  class Pieces  {  // Ware를 승계하여,  // Piece는 보드위에 나타난 말이 컴퓨터인지 사용자인지를 나타내는 클래스이다.  private :  // 이 기물이 컴퓨터라면 True  bool computer;  public :  /\*\*  \* @brief Construct a new Pieces object  \*  \* @param \_isComputer  \*/  Pieces(bool \_isComputer);  Pieces(const Pieces& \_piece);  bool getComputer();  void reverse(); // 바꿈.  };  #endif // PIECES\_H  // Path: classes/data/Pieces.cpp |

Pieces.cpp

|  |
| --- |
| /\*\*  \* @brief 기물을 정의하는 class  \* (Ware가 너무 넓다는 문제에 따라서, 이름을 변경하는 refactor를 집행함.)  \*/  #include "Pieces.h"  Pieces::Pieces(bool \_isComputer){  this->computer = \_isComputer;  }  Pieces::Pieces(const Pieces& \_piece){  this->computer = \_piece.computer;  }  bool Pieces::getComputer(){  return this->computer;  }  void Pieces::reverse(){  this->computer = !this->computer;  }  // Path: classes/data/Pieces.h |

BoardCheck.h

|  |
| --- |
| #ifndef BOARDCHECK\_H  #define BOARDCHECK\_H  #include "../data/Board.h" // Add this line to include the "Board.h" file  #include "../../enums/PlayerGameResults.h"  #include "../../enums/GameState.h"  #include "../data/DataActions.h"  #include "ComPlay.h"  #include <vector>  using namespace std;  class BoardCheck  {  private:  Board\* board;  ComPlay\* complay;  public:  BoardCheck(Board\* \_board,ComPlay \*\_complay);  ~BoardCheck();  int getUserScore();  int getComputerScore();  PlayerGameResults checkGameResult();  GameState isGameEnd();    static vector<DataActions> getCanSetListOnBoard(Board\* \_bd,bool computer);  };  #endif // BOARDCHECK\_H  // Path: classes/schema/BoardCheck.h |

BoardCheck.cpp

|  |
| --- |
| #include "BoardCheck.h"  #include "../data/Board.h"  #include "../../enums/PlayerGameResults.h"  #include "../../enums/PiecesCode.h"  #include "../../enums/GameState.h"  #include <vector>  #include <map>  #include "ComPlay.h"  #include <iostream>  using namespace std;  BoardCheck::BoardCheck(Board\* bd,ComPlay \*mp){  this->board = bd;  this->complay = mp;  }  BoardCheck::~BoardCheck(){  }  /\*\*  \* @brief "게임 결과와는 상관하지 아니하고, " 사용자의 점수를 반환하는 함수  \*  \* @return int 현 시점에서의 사용자의 점수  \*/  int BoardCheck::getUserScore(){  PiecesCode\*\* bd = this->board->getBoardInfo();  int size = this->board->getSize();  int result = 0;  for(int a = 0 ; a < size ; a ++)  {  for (int b = 0;b < size; b ++)  {  if(bd[a][b] == PiecesCode::USER) result++; // 사용자의 기물이라면 점수를 증가시킴.  }  }  return result;  }  /\*\*  \* @brief "게임 결과와는 상관하지 아니하고, " 컴퓨터의 점수를 반환하는 함수  \*  \* @return int 현 시점에서의 컴퓨터의 점수  \*/  int BoardCheck::getComputerScore(){  PiecesCode\*\* bd = this->board->getBoardInfo();  int size = this->board->getSize();  int result = 0;  for(int a = 0 ; a < size ; a ++)  {  for (int b=0;b<size;b++)  {  if(bd[a][b] == PiecesCode::COMPUTER) result++; // 컴퓨터의 기물이라면 점수를 증가시킴.  }  }  return result;  }  /\*\*  \* @brief 게임 결과를 계산하는 함수  \*  \*/  PlayerGameResults BoardCheck::checkGameResult(){  /\*\*  \* @brief 오셀로에서, 게임의 승패는 하기와 같이 결정된다.  \* 1. 일단 모든 칸이 채워졌을 때, 더 많은 기물을 가진 플레이어가 승리한다.  \* 2. 만약, 모든 플레이어가 더이상의 수를 둘 수 없을 때, 더 많은 기물을 가진 플레이어가 승리한다.  \* 3. 만약, 두 플레이어가 같은 수의 기물을 가지고 있다면, 무승부가 된다.  \*  \*/  if(this->isGameEnd() == GameState::NONE){  int user\_score = this->getUserScore();  int computer\_score = this->getComputerScore();  if(user\_score > computer\_score) return PlayerGameResults::WIN;  else if(user\_score < computer\_score) return PlayerGameResults::LOSE;  else return PlayerGameResults::DRAW;  }  return PlayerGameResults::PROGRESS;  }  /\*\*  \* @brief 게임이 끝났는지 확인하는 함수  \*  \* @return bool 게임이 끝났는지  \*/  GameState BoardCheck::isGameEnd(){  // 1. 모든 칸이 채워졌는지 확인  PiecesCode\*\* bd = this->board->getBoardInfo();  int size = this->board->getSize();  bool is\_full = true;  cout << "size: " << size << endl;  for(int a = 0 ; a < size ; a ++){  for (int b=0;b<size;b++){  if(bd[a][b] == PiecesCode::EMPTY) is\_full = false;  }  }  if(is\_full == true) return GameState::NONE;  // 2. 더이상의 수를 둘 수 있는지 확인  vector<DataActions> can\_set\_list\_computer = getCanSetListOnBoard(this->board,true);  vector<DataActions> can\_set\_list\_user = getCanSetListOnBoard(this->board,false);  #pragma region Debug  cout << "can\_set\_list\_computer: " << can\_set\_list\_computer.size() << endl;  cout << "can\_set\_list\_user: " << can\_set\_list\_user.size() << endl;  #pragma endregion Debug  if(can\_set\_list\_computer.size() == 0 && can\_set\_list\_user.size() == 0) return GameState::NONE;  if(can\_set\_list\_computer.size() == 0) return GameState::PASS\_USER\_ONLY;  if(can\_set\_list\_user.size() == 0) return GameState::PASS\_COMPUTER\_ONLY;  return GameState::BOTH;  }  vector<DataActions> BoardCheck::getCanSetListOnBoard(Board\* \_bd,bool computer){  vector<DataActions> result;  PiecesCode\*\* bd = \_bd->getBoardInfo();  int size = \_bd->getSize();  PiecesCode enemy = computer == true ? PiecesCode::USER : PiecesCode::COMPUTER;  map<pair<int,int>,bool> checked = map<pair<int,int>,bool>();  for(int a = 0 ; a < size ; a ++){  for (int b=0;b<size;b++){  if(bd[a][b] == enemy){  // 상하좌우 대각선을 조사  for(int i = -1 ; i <= 1 ; i ++){  for(int j = -1 ; j <= 1 ; j ++){  if(i == 0 && j == 0) continue; //자기자신이면 생략  int x = a + i;  int y = b + j;  if(x < 0 || x > size-1 || y < 0 || y > size-1) continue; // 범위를 벗어나면 생략  if(bd[x][y] != PiecesCode::EMPTY) continue; // 이미 기물이 있으면 생략  if(checked.find(make\_pair(x,y)) != checked.end()) continue; // 이미 조사한 곳이면 생략  if(\_bd->searchCanSetInThisPoint(x,y,computer)==false) continue; // 둘 수 없는 곳이면 생략  checked[make\_pair(x,y)] = true;  result.push\_back(DataActions(x,y,0));  }  }  }  }  }  return result;  }  // Path: classes/schema/BoardCheck.cpp |

BoardDraw.h

|  |
| --- |
| #ifndef BOARDDRAW\_H  #define BOARDDRAW\_H  #include "../data/Board.h"  class BoardDraw  {  private:  Board\* board;  public:  BoardDraw(Board\* \_board);  ~BoardDraw();  void draw();  };  #endif // BOARDDRAW\_H  // Path: classes/schema/UserPlay.h |

BoardDraw.cpp

|  |
| --- |
| #include "BoardDraw.h"  #include "../data/Board.h"  #include <iostream>  #include <string>  using namespace std;  BoardDraw::BoardDraw(Board\* \_board)  {  this->board = \_board;  }  BoardDraw::~BoardDraw()  {  }  void BoardDraw::draw()  {  PiecesCode\*\* boardInfo = this->board->getBoardInfo();  int size = this->board->getSize();  for(int a = 0 ; a < size ; a ++)  {  if(a == 0) {  cout << "---|";  for(int b = 0 ; b < size ; b++)  {  if( b < 9 )  cout << " " << (b+1) << " |";  else  cout << " " << (b+1) << "|";  }  cout << endl;  }  for (int b=0;b<size;b++)  {  if(b == 0) {  if(a<9)  cout << " " << (a+1) << " |";  else  cout << " " <<(a+1) <<"|";  }    char show = boardInfo[a][b] != PiecesCode::EMPTY ?  (boardInfo[a][b] == PiecesCode::COMPUTER ? 'O' : 'X')  : ' ' ;  cout << " " << show << " |" ;  }  cout << endl;  cout << "---|";  for(int b = 0 ; b < size ; b++)  {  cout << "---|";  }  cout << endl;  }  } |

ComPlay.h

|  |
| --- |
| #ifndef COMPLAY\_H  #define COMPLAY\_H  // TODO : 예언자를 승계한다.  // 승계시, 멀티스레딩과 해싱을 사용하여 성능 향상을 추진한다.  #include "../data/DataActions.h"  #include "../data/Board.h"  class ComPlay  {  private:  Board\* board;  DataActions predictSingle(Board\* bd,int i,int j);// 스레딩용 단일 예측자  double\*\* predictWeight; // 예측 가중치  int size; // 예측 가중치 사이즈 (보드사이즈와 같다.)  public:  ComPlay(Board\* \_board);  ~ComPlay();  DataActions predict(); // 예측 실행후 반환  };  #endif // COMPLAY\_H  // Path: classes/schema/ComPlay.h |

ComPlay.cpp

|  |
| --- |
| #include "ComPlay.h"  #include <thread>  #include <mutex>  #include <vector>  #include <future>  #include "../data/DataActions.h"  #include "algorithm"  #include <iostream>  #include "BoardCheck.h"  ComPlay::ComPlay(Board\* \_board)  {  board = \_board;  size = board->getSize();  predictWeight = new double\*[size];  for (int i = 0; i < size; i++)  {  predictWeight[i] = new double[size];  for (int j = 0; j < size; j++)  {  if(j==0 || j==size-1 || i==0 || i==size-1){  if( (i==0 && j==0) || (i==0 && j==size-1) || (i==size-1 && j==0) || (i==size-1 && j==size-1) )  predictWeight[i][j] = 1000;  else if(i==1 || j == 1 || i == size-2 || j == size-2)  predictWeight[i][j] = 1;  else  predictWeight[i][j] = 10;  }  else  {  if((i == 1 && j == 1) || (i==size-2 && j== 1) || ( i == 1 && j == size - 2 ) || (i == size-2 && j == size-2))  predictWeight[i][j] = 1;  else  predictWeight[i][j] = 5;  }  }  }  }  ComPlay::~ComPlay()  {  for (int i = 0; i < size; i++)  {  delete[] predictWeight[i];  }  delete[] predictWeight;  }  /\*\*  \* @brief 확률값만 예언한다.  \*  \* @param bd  \* @param i  \* @param j  \* @return double  \*/  DataActions ComPlay::predictSingle(Board\* bd, int i,int j)  {  /\*\*  \* @brief 원래라면, 이 함수는 모든 경우의 수를 따져야 하나, 재귀함수 호출시 너무 많은 연산을 하게 될 예정일 것이고, 요구사항에서 Heuristic한 방안을 주문한 바, 그냥 1페이즈 시뮬레이션의 결과만 greedy하게 반영하게 한다. 끝.  \*/  if(i < 0 || i >= size-1 || j < 0 || j >= size-1)  return DataActions(i,j,-99);  if(bd->getBoardInfo()[i][j] != PiecesCode::EMPTY)  return DataActions(i,j,-99); // 거긴 못둠.  Board simulator = Board(\*bd); // 문제없게 복사.  // 일단 주어진 시나리오를 시뮬레이션 한다.  simulator.setInput(true,i,j);  int result = 0;  for(int x=0;x<size;x++)  {  for(int y=0;y<size;y++)  {  if(simulator.getPieceCode(x,y) == PiecesCode::COMPUTER)  {  simulator.setInput(false,x,y);  }  }  }  for(int x=0;x<size;x++)  {  for(int y=0;y<size;y++)  {  if(simulator.getPieceCode(x,y) == PiecesCode::COMPUTER)  {  result += predictWeight[x][y];  }  }  }  //delete simulator; // auto value이다. 지울필요없다.  // Weight + 점수값 (점수값 == 점수값 delta와 같은 효과이므로.)  return DataActions(i,j,predictWeight[i][j] + result);  }  DataActions ComPlay::predict()  {  vector<DataActions> canAction = BoardCheck::getCanSetListOnBoard(board,true);  vector<future<DataActions>> futures;  futures.reserve(canAction.size());  double max = 0;  int x = -1;  int y = -1;  for(int i = 0; i < canAction.size(); i++)  {  futures.emplace\_back(async(launch::async,&ComPlay::predictSingle,this,board,canAction[i].x,canAction[i].y));  }  for(auto& f : futures)  {  DataActions da = f.get();  if(da.score > max)  {  max = da.score;  x = da.x;  y = da.y;  }  }  return DataActions(x,y,max);  } |

UserPlay.h

|  |
| --- |
| #ifndef USERPLAY\_H  #define USERPLAY\_H  #include "../data/Board.h"  #include "ComPlay.h"  #include "../data/GameResult.h"  #include "BoardCheck.h"  #include "BoardDraw.h"  class UserPlay  {  private:  bool computerFirst;  Board\* board;  ComPlay\* complay;  BoardCheck\* boardCheck;  BoardDraw\* boardDraw;  public:  UserPlay(bool computer\_first);  ~UserPlay();  void setInput();  void play();  bool getEnded();  void showBoard();  GameResult getGameResult();  };  #endif // USERPLAY\_H  // Path: classes/schema/UserPlay.cpp |

UserPlay.cpp

|  |
| --- |
| #include "UserPlay.h"  #include <iostream>  #include "../data/Board.h"  #include "../data/Pieces.h"  #include "../data/GameResult.h"  #include "ComPlay.h"  #include "BoardCheck.h"  #include "BoardDraw.h"  #pragma region enums\_include  #include "../../enums/PlayerGameResults.h"  #include "../../enums/PiecesCode.h"  #include "../../enums/GameState.h"  #pragma endregion enums\_include  using namespace std;  UserPlay::UserPlay(bool computer\_first)  {  int size;  do{  cout << "보드의 크기를 입력하세요 : ";  cin >> size;  if(size < 6 || size > 12) cout << "6~12 사이의 값을 입력해주세요." << endl;  }while(size < 6 || size > 12); // 6~12 사이의 값만 받는다.    Board\* bod = new Board(size);  this->board = bod;  cout << "보드를 초기화합니다.";  ComPlay\* lComplay = new ComPlay(bod);  BoardCheck\* lBoardCheck = new BoardCheck(bod,lComplay);  this->boardDraw = new BoardDraw(bod);  this->complay = lComplay;  this->boardCheck = lBoardCheck;  cout << "Board size... : " << size << endl;  this->computerFirst = computer\_first;  // 이하 초기값 세팅  if(this->computerFirst)  {  this->board->setInput(true,this->board->getSize()/2,this->board->getSize()/2,true);  this->board->setInput(true,this->board->getSize()/2-1,this->board->getSize()/2-1,true);  this->board->setInput(false,this->board->getSize()/2-1,this->board->getSize()/2,true);  this->board->setInput(false,this->board->getSize()/2,this->board->getSize()/2-1,true);  }  else  {  this->board->setInput(false,this->board->getSize()/2,this->board->getSize()/2,true);  this->board->setInput(false,this->board->getSize()/2-1,this->board->getSize()/2-1,true);  this->board->setInput(true,this->board->getSize()/2-1,this->board->getSize()/2,true);  this->board->setInput(true,this->board->getSize()/2,this->board->getSize()/2-1,true);  }  cout << "보드 초기화 완료" << endl;  }  UserPlay::~UserPlay()  {  delete this->complay;  delete this->boardCheck;  delete this->board;  delete this->boardDraw;  }  void UserPlay::setInput()  {  bool end = false;  while (end == false){  this->boardDraw->draw();  cout << "좌표를 입력하세요. (x,y) :";  int x,y;  cin.ignore();  scanf("(%d,%d)",&x,&y);  // TODO : 유저가 선택가능한 곳인지 확인해야 함.  // setInput에서 확인하게 하였음.  InputErrorCode code = this->board->setInput(false,y-1,x-1);  switch(code)  {  case InputErrorCode::VALID\_INPUT:  cout << "유저의 입력이 완료되었습니다." << endl;  end = true; // 입력이 완료되었으므로 루프를 빠져나간다.  break;  case InputErrorCode::INVALID\_INPUT\_X\_0:  cout << "x좌표가 0보다 작습니다." << endl;  break;  case InputErrorCode::INVALID\_INPUT\_X\_MAX:  cout << "x좌표가 최대값을 초과합니다." << endl;  break;  case InputErrorCode::INVALID\_INPUT\_Y\_0:  cout << "y좌표가 0보다 작습니다." << endl;  break;  case InputErrorCode::INVALID\_INPUT\_Y\_MAX:  cout << "y좌표가 최대값을 초과합니다." << endl;  break;  case InputErrorCode::INVALID\_INPUT\_THERE\_IS\_ALREADY:  cout << "이미 기물이 있습니다." << endl;  break;  case InputErrorCode::INVALID\_INOUT\_THERE\_IS\_NO\_PIECE\_TO\_REVERSE:  cout << "뒤집을 기물이 없습니다." << endl;  break;  }  if(end == false)  {  cout << "다시 입력해주세요.(x,y)" << endl;  }  }  }  void UserPlay::play()  {  // TODO...  if(this->computerFirst)  {  DataActions computerAction = this->complay->predict();  this->board->setInput(true,computerAction.x,computerAction.y);  // 컴퓨터가 둠으로서 게임이 끝났는가?  if(this->boardCheck->isGameEnd() == GameState::NONE)  {  return; // 끝났다고 판단되면 조기리턴한다.  }  else if(this->boardCheck->isGameEnd() == GameState::PASS\_COMPUTER\_ONLY)  {  // 컴퓨터가 둔 상태에서, 컴퓨터만 둘 수 있다면 유저의 턴은 생략한다. (그러나 게임은 끝나지 않았으므로 다시 돌아온다.)  cout << "유저가 더이상 둘 수 있는 곳이 없습니다." << endl;  return;  }  this->setInput();  }  else  {  this->setInput();  // 유저가 둠으로서 게임이 끝났는가?  if(this->boardCheck->isGameEnd() == GameState::NONE)  {  return;  }  else if(this->boardCheck->isGameEnd() == GameState::PASS\_USER\_ONLY)  {  // 유저가 둔 상태에서, 유저만 둘 수 있다면 컴퓨터의 턴은 생략한다. (그러나 게임은 끝나지 않았으므로 다시 돌아온다.)  cout << "컴퓨터가 더이상 둘 수 있는 곳이 없습니다." << endl;  return;  }  this->boardDraw->draw();  DataActions computerAction = this->complay->predict();  this->board->setInput(true,computerAction.x,computerAction.y);  }  }  bool UserPlay::getEnded()  {  return this->boardCheck->isGameEnd() == GameState::NONE;  }  GameResult UserPlay::getGameResult()  {  bool isGameEnd = this->boardCheck->isGameEnd() == GameState::NONE;  if(isGameEnd == false)  {  return GameResult(0,0,false); // 게임이 끝나지 않았다면 빈 결과를 반환한다.  }  PiecesCode\*\* bd = this->board->getBoardInfo();  int size = this->board->getSize();  int userScore = 0;  int computerScore = 0;  for (int a = 0; a < size ; a++)  {  for (int b = 0; b < size ; b++)  {  if(bd[a][b] == PiecesCode::USER) userScore++;  else if(bd[a][b] == PiecesCode::COMPUTER) computerScore++;  else if(bd[a][b] == PiecesCode::EMPTY) continue;  }  }  return GameResult(userScore,computerScore,isGameEnd);  }  void UserPlay::showBoard()  {  this->boardDraw->draw();  }  // Path: classes/schema/UserPlay.h |

|  |  |
| --- | --- |
| GameState.h | InputErrorCode.h |
| #ifndef GAMESTATE\_H  #define GAMESTATE\_H  enum class GameState // 한 게임의 상태를 의미한다.  {  BOTH=1, // 둘 다 둘 수 있다.  PASS\_USER\_ONLY=2, // 사용자만 둘 수 있다.  PASS\_COMPUTER\_ONLY=3, // 컴퓨터만 둘 수 있다.  NONE=0 // 승패가 정해짐.  };  #endif // GAMESTATE\_H  // Path: src/enums/GameState.h | #ifndef INPUTERRORCODE\_H  #define INPUTERRORCODE\_H  enum InputErrorCode  {  INVALID\_INPUT\_X\_0 = -1,  INVALID\_INPUT\_X\_MAX = -2,  INVALID\_INPUT\_Y\_0 = -3,  INVALID\_INPUT\_Y\_MAX = -4,  INVALID\_INPUT\_THERE\_IS\_ALREADY = -5,  INVALID\_INOUT\_THERE\_IS\_NO\_PIECE\_TO\_REVERSE = -6,  VALID\_INPUT = 1  };  #endif // INPUTERRORCODE\_H |

|  |  |
| --- | --- |
| PiecesCode.h | PlayerGameResults.h |
| #ifndef PIECESCODE\_H  #define PIECESCODE\_H  enum PiecesCode  {  EMPTY = 0,  USER = 1,  COMPUTER = -1  };  #endif // PIECESCODE\_H | #ifndef GAME\_RESULTS\_H  #define GAME\_RESULTS\_H  enum PlayerGameResults {  WIN = 1,  LOSE = -1,  PROGRESS = 0,  DRAW = -2  };  #endif // GAME\_RESULTS\_H |

Main.cpp

|  |
| --- |
| /\*\*  \* @file main.cpp  \* @author 이준영 (presan100@gmail.com)  \* @brief KNUT C++ Programming Final Project Othello Game  \* @version 0.1  \* @date 2024-05-29  \*  \* @copyright Copyright (c) 2024  \*  \*/  #pragma region Include HEADERS  #include "src/classes/schema/BoardCheck.h"  #include "src/classes/schema/ComPlay.h"  #include "src/classes/schema/UserPlay.h"  #include "src/classes/schema/BoardDraw.h"  #include "src/classes/data/Board.h"  #pragma endregion Include HEADERS  #include <string>  #include <iostream>  int main()  {  cout <<"2017011 이준영 C++ Final Project Othello Game" << endl;  while(true)  {  cout << "컴퓨터의 선공이면 o를 입력하세요 : ";  string c;  getline(cin,c);  UserPlay\* userPlay = new UserPlay(c=="o");  // 초기값 셋팅을 완수하였다. 그렇다면, 진행한다.  // 게임이 끝날때까지 턴을 돌린다.  while (userPlay->getEnded() == false)  {  userPlay->play();  }  GameResult result = userPlay->getGameResult();  cout << "게임이 끝났습니다. " << endl;  cout << "게임 결과 : " << endl;  cout << "사용자의 점수 : " << result.userScore << endl;  cout << "컴퓨터의 점수 : " << result.computerScore << endl;  cout << "게임의 승자 : ";  if(result.userScore > result.computerScore)  {  cout << "사용자" << endl;  }  else if(result.userScore < result.computerScore)  {  cout << "컴퓨터" << endl;  }  else  {  cout << "무승부" << endl;  }  userPlay->showBoard();  delete userPlay; // 게임이 끝나면, userPlay를 삭제한다.  cout << "게임을 다시 시작하시겠습니까? (y/n) : ";  cin.ignore();  string s;  getline(cin,s);  if(s=="n") break;  }  return 0;  } |

### 실행결과