작성자/소속	승인자	문서관리자	검토	문서종류
으라차차 / 충북대학교 소프트웨어학과	엄남경	으라차차	으라차차	기술문서

제 목	Title
오목 트레이닝 프로그램	오목 트레이닝 프로그램

최종보고서

(과제명 : 오목 트레이닝 프로그램)

문서 정보

구 분	소속	성명	날짜	서명
작성자	소프트웨어학과	최승진	2017.06.08	**최승진**
	소프트웨어학과	오상헌	2017.06.08	**오상헌**
	소프트웨어학과	양회욱	2017.06.08	**양회욱**
검토자	소프트웨어학과	엄남경		
승인자	소프트웨어학과	엄남경		
버 전	V1.1			
발행일	2017.06.08			
상 태	**완료**			

개정 이력

버전	개정일자	개정 내역	작성자	확인자
1.0	2017.06.05	최종보고서 초안 작성	으라차차	엄남경
1.1	2017.06.08	최종보고서 최종 발행	으라차차	엄남경

목 차

1. 개요

1.1 목적

이 문서는 오목 트레이닝 프로그램(사용자가 처음 오목을 접했을 때 규칙을 이해하고 원활하게 게임 플레이를 하기 위한 기술 개발)의 사용자 요구사항과 기능설계와 구현이 포함된 문서이다.

본 문서는 “오목 트레이닝 프로그램”의 3 가지를 기술하고 있다. 첫번째는 사용자의 요구사항을 요약하여 기술하고 있다. 두번째는 프로그램의 5 가지 기능(사용자 계정 관리 기능, 게임 플레이 공통 기능, 1 인용 게임 플레이 기능, 2 인용 게임 플레이 기능, 게임 설명 기능)의 핵심 기술을 서술하고 있다. 세번째는 사용자 요구사항과 프로그램 기능설계를 바탕으로 구현한 프로그램을 자세하게 기술하고 있다.

프로그램의 전반적인 이해를 돕고 사용자의 요구사항을 적용하여 기능을 설계하고 구현한 과정 및 결과를 파악할 수 있도록 하는 것이 이 문서의 목적이다.

1.2 참고문헌

[1] 응용 프로그램 개발을 위한 명품 C언어 프로젝트
[2] 재미있는 필승 오목교실
[3] 한국오목협회 오목클럽매거진
[4] Monte Carlo Tree Search [http://www.cameronius.com/research/mcts/index.html]

1.3 용어 및 약어

1) Gomoku 룰

: 오목 국제 대회에서 사용하는 룰은 두 가지가 있는데 그 중 하나가 Gomoku 룰이다. Gomoku 룰은 항상 흑이 먼저 두기 시작하고 흑과 백 모두 놓지 못하는 위치가 없으며, 먼저 자신의 오목 알이 차례대로 5 개가 위치하면 승리하는 룰이다.

2. 사용자 요구사항

2.1 요구사항 요약

로그인한 사용자가 게임을 진행하고, 게임 전적을 사용자에게 보여줌으로써 오목 실력을 향상시킬 수 있도록 하는 게임 기술을 개발한다. 순수한 게임 실력 향상을 위해 가장 공정한 게임 규칙을 적용한다.

즉, 필요로 하는 기능은 다음과 같이 크게 5 가지로 분류할 수 있다.

- 사용자 계정 관리 기능
- 게임 플레이 공통기능
- 1 인용 게임 플레이 기능
- 2 인용 게임 플레이 기능
- 게임 설명 기능

2.2 주요 요구사항

2.2.1 사용자 계정 관리 기능

Req. ID	상세 내용	M/O	개발자
1UFR-001	사용자가 시스템을 처음 시작할 때 아이디, 비밀번호, 이름, 나이, email 을 입력하고 오목알의 모양을 설정하여 회원가입을 할 수 있다.	M	최승진
1UFR-002	사용자가 아이디와 비밀번호를 이용하여 시스템에 로그인할 수 있다.	M	최승진
1UFR-003	로그인을 한 사용자가 개인정보 및 전적을 볼 수 있다.	M	최승진
1UFR-004	로그인을 한 사용자가 게임 플레이를 마친 후에 로그아웃을 할 수 있다	M	최승진

2.2.2 게임 플레이 공통 기능

Req. ID	상세 내용	M/O	개발자
2UFR-001	오목알은 사용자가 사용자 계정 관리 기능을 통해 설정한 오목 알 모양이 놓여야 한다.	M	양회욱
2UFR-002	사용자가 자신의 턴일 때 일정 시간(30 초)이 지나면 기권패가 되게 한다	M	양회욱
2UFR-003	오목 게임은 Gomoku 규칙에 근거하여 승패가	M	양회욱

	결정되어야 한다.		
2UFR-004	사용자가 오목 알을 둘 때, 방향키를 이용해서 원하는 위치에 둘 수 있게 한다.	M	양회욱

2.2.3 1인용 게임 플레이 기능

Req. ID	상세 내용	M/O	개발자
3UFR-001	사용자가 컴퓨터와 대결할 수 있다.	M	오상헌
3UFR-002	게임의 순서는 사용자를 우선으로 한다.	M	오상헌
3UFR-003	사용자는 자신의 순서에 오목 알을 둔다.	M	오상헌

2.2.4 2인용 게임 플레이 기능

Req. ID	상세 내용	M/O	개발자
4UFR-001	게임 시작 전, 함께 플레이 할 사용자가 로그인을 해야한다.	M	최승진
4UFR-002	게임의 순서는 임의로 결정된다.	M	양회욱
4UFR-003	두 사용자는 자신의 순서에 오목 알을 둔다.	M	양회욱

2.2.5 게임 설명 기능

Req. ID	상세 내용	M/O	개발자
5UFR-001	사용자가 튜토리얼 메뉴를 선택했을 때, 직접 컴퓨터의 설명에 따라 연습게임 플레이를 해보면서 게임 규칙을 이해한다.	M	양회욱

3. 기능설계

"오목 트레이닝 프로그램"의 5 가지 기능에 대하여 기술하고 설계한다. 각 기능은 사용자 계정 관리 기능, 게임 플레이 공통 기능, 1 인용 게임 플레이 기능, 2 인용 게임 플레이 기능, 게임 설명 기능이다.

3.1 기능 설명

"오목 트레이닝 프로그램(OTP)" 기능은 사용자가 처음 오목을 접할 때 규칙을 이해하고 원활하게 게임을 할 수 있도록 하는 기능을 정의한다.'

3.1.1 (OTP-FUNS2017-1) 사용자 계정 관리 기능

사용자가 프로그램에 로그인을 하여 사용자 정보 조회 및 수정 또는 게임 플레이시 필요한 설정을 진행하는 기능이다.

3.1.2 (OTP-FUNS2017-2) 게임 플레이 공통 기능

사용자와 사용자 또는 사용자와 컴퓨터가 오목 게임을 플레이할 때 필요한 규칙과 제한을 규정하는 기능이다.

3.1.3 (OTP-FUNS2017-3) 1인용 게임 플레이 기능

사용자가 컴퓨터와 오목 게임 대결을 할 수 있도록 하는 기능이다.

3.1.4 (OTP-FUNS2017-4) 2인용 게임 플레이 기능

두 명의 사용자가 오목 게임 대결을 할 수 있도록 하는 기능이다.

3.1.5 (OTP-FUNS2017-5) 게임 설명 기능

사용자가 직접 연습 플레이를 하며 게임 규칙을 이해하도록 하고, 프로그램 사용 방법을 볼 수 있게 하는 기능이다.

3.2 데이터 정의

3.2.1 오목판 (Board) 포맷

이름	Type	설명
OM1	UINT64	오목판의 첫번째 부분
OM2	UINT64	오목판의 두번째 부분
OM3	UINT64	오목판의 세번째 부분
OM4	UINT64	오목판의 네번째 부분

```
typedef  struct  {
    UINT64          OM1;
    UINT64          OM2;
    UINT64          OM3;
    UINT64          OM4;
    } Board;
```

3.2.2 사용자 (User) 포맷

이름	Type	설명
Id	UINT32	사용자를 식별하기 위한 정보.
Name	String	사용자의 이름 정보.
Age	UINT16	사용자의 나이 정보.
Email	String	사용자의 이메일 정보.
Nickname	String	사용자가 로그인할 때 필요한 닉네임 정보.
Stone_Shape1	String	사용자가 게임 플레이시 두는 오목알(백)의 모양 정보.
Stone_Shape2	String	사용자가 게임 플레이시 두는 오목알(흑)의 모양 정보.

```
typedef  struct  {
    UINT16          Id;
    String          Name;
    UINT16          Age;
    String          Email;
    String          Nickname;
```

```
    String          Password;
    String          Stone_Shape1;
    String          Stone_Shape2;

} User;
```

3.2.3 트리 노드 (Node) 포맷

이름	Type	설명
boardWhite	Board (Struct)	게임 진행중인 오목판의 흰돌이 놓여있는 정보.
boardBlack	Board (Struct)	게임 진행중인 오목판의 검은돌이 놓여있는 정보.
boundWhite	Bound (Struct)	흰돌이 놓여있는 범위를 나타내는 정보.
boundBlack	Bound (Struct)	검은돌이 놓여있는 범위를 나타내는 정보.
Win	UINT32	해당 상태의 승리 횟수 정보.
N	UINT32	해당 상태의 게임 시뮬레이션 정보.
Turn	Turn (Enum)	현재 상태에 돌을 두어야하는 차례 정보.
nodes	NodeArray (Struct)	트리 자식 리스트.

```
typedef   struct   {
    Board           boardWhite;
    Board           boardBlack;
    Bound           boundWhite;
    Bound           boundBlack;
    UINT32          Win;
    UINT32          N;
    Turn            turn;
    NodeArray               nodes;
} Node;
```

3.2.4 트리 노드 배열 (NodeArray) 포맷

이름	Type	설명
array	Node*(Struct)	Node 의 값이 배열로 저장되어 있는 포인터.
size	Size_t	배열에 들어가 있는 Node 의 개수.
allocated	Size_t	실제로 배열에 할당된 Node 의 개수의 메모리.

```
typedef   struct   {
    Node*          array;
    Size_t         size;
    Size_t         allocated;
} NodeArray;
```

3.2.5 오목알 위치 범위(Bound) 포맷

이름	Type	설명
Max_x	UINT32	오목판의 돌이 두어진 행의 최대 범위를 저장하는 정보
Min_x	UINT32	오목판의 돌이 두어진 행의 최소 범위를 저장하는 정보
Max_y	UINT32	오목판의 돌이 두어진 열의 최대 범위를 저장하는 정보
Min_y	UINT32	오목판의 돌이 두어진 열의 최소 범위를 저장하는 정보

```
typedef   struct   {
    UINT32         max_x;
    UINT32         min_x;
    UINT32         max_y;
    UINT32         min_y;
} Bound;
```

3.2.6 게임 순서(Turn) 포맷

이름	Type	설명
Black	UINT32	검은돌이 두어야하는 차례를 나타내는 정보.
White	UINT32	흰돌이 두어야하는 차례를 나타내는 정보.

Typedef enum { Black, White } Turn;

3.2.7 참/거짓 여부(bool) 포맷

이름	Type	설명
false	UINT32	거짓을 나타내는 정보.
true	UINT32	참을 나타내는 정보.

Typedef enum { false, true } bool;

3.3 함수 정의

3.3.1 최승진

식별번호	함수		입력 데이터		출력 데이터
	타입	이름	타입	이름	
1OPT-C-1	void	Input_data	_User	User	void
1OPT-C-2	int	login	MYSQL	conn	Int(0,1)
1OPT-C-3	void	Print_Information	Char*	Nickname password	Void
1OPT-C-4	int	Calculater_rate	int	Win lose	Int(rate)
1OPT-C-5	int	login2	MYSQL	conn	Int(0,1)
1OPT-C-6	int	insert	MYSQL * _User user	conn user	int

3.3.2 오상헌

식별번호	이름	입력 데이터		출력 데이터 타입
		타입	이름	
3OTP-O-001	makeBoard	UINT64	Om1	Board*
		UINT64	Om2	
		UINT64	Om3	
		UINT64	Om4	
3OTP-O-002	board_mask	Board*	board	Board*
3OTP-O-003	board_shift_right	Board*	board	Board*
		INT	shift	
3OTP-O-004	board_shift_left	Board*	board	Board*
		INT	shift	
3OTP-O-005	board_and	Board*	baseBoard	Board*

		Board*	layerBoard	
3OTP-O-006	board_or	Board*	userId	Board*
		Board*	layerBoard	
3OTP-O-007	board_xor	Board*	userId	Board*
		Board*	layerBoard	
3OTP-O-008	board_set	UINT32	x	Board*
		UINT32	y	
		Board*	board	
3OTP-O-009	board_isSet	UINT32	x	UINT32
		UINT32	y	
		Board*	board	
3OTP-O-010	board_copy	Board*	userId	Board*
		Board*	layerBoard	
3OTP-O-011	board_isEmpty	Board*	Board	Board*
3OTP-O-012	board_isEqual	Board*	boardA	Board*
		Board*	boardB	
3OTP-O-013	board_find_diff_one	Board*	Bigger	Board*
		Board*	Smaller	
		Int*	X	
		Int*	y	

3OTP-O-014	narray_init	-	-	NodeArray*
3OTP-O-015	narray_insert	NodeArray*	a	void
		Node	element	

3OTP-O-016	narray_copy	NodeArray*	target	NodeArray*
3OTP-O-017	narray_free	NodeArray*	a	void
3OTP-O-018	MCTS	Node*	root	void
3OTP-O-019	simulate	Node*	node	int
3OTP-O-020	selectHeuristic	Node*	root	Node*
3OTP-O-021	node_init	Node*	node	Node*
3OTP-O-022	node_copy	Node*	target	Node*
		bool	liveChild	
3OTP-O-023	setBound	Node*	node	void
		int	x	
		int	y	
		Turn	turn	
3OTP-O-024	ready	-	-	void
3OTP-O-025	findNext	Node*	node	Node*
3OTP-O-026	findChildSameBoard	Node*	node	Node*
		Board*	blackBoard	
		Board*	whiteBoard	

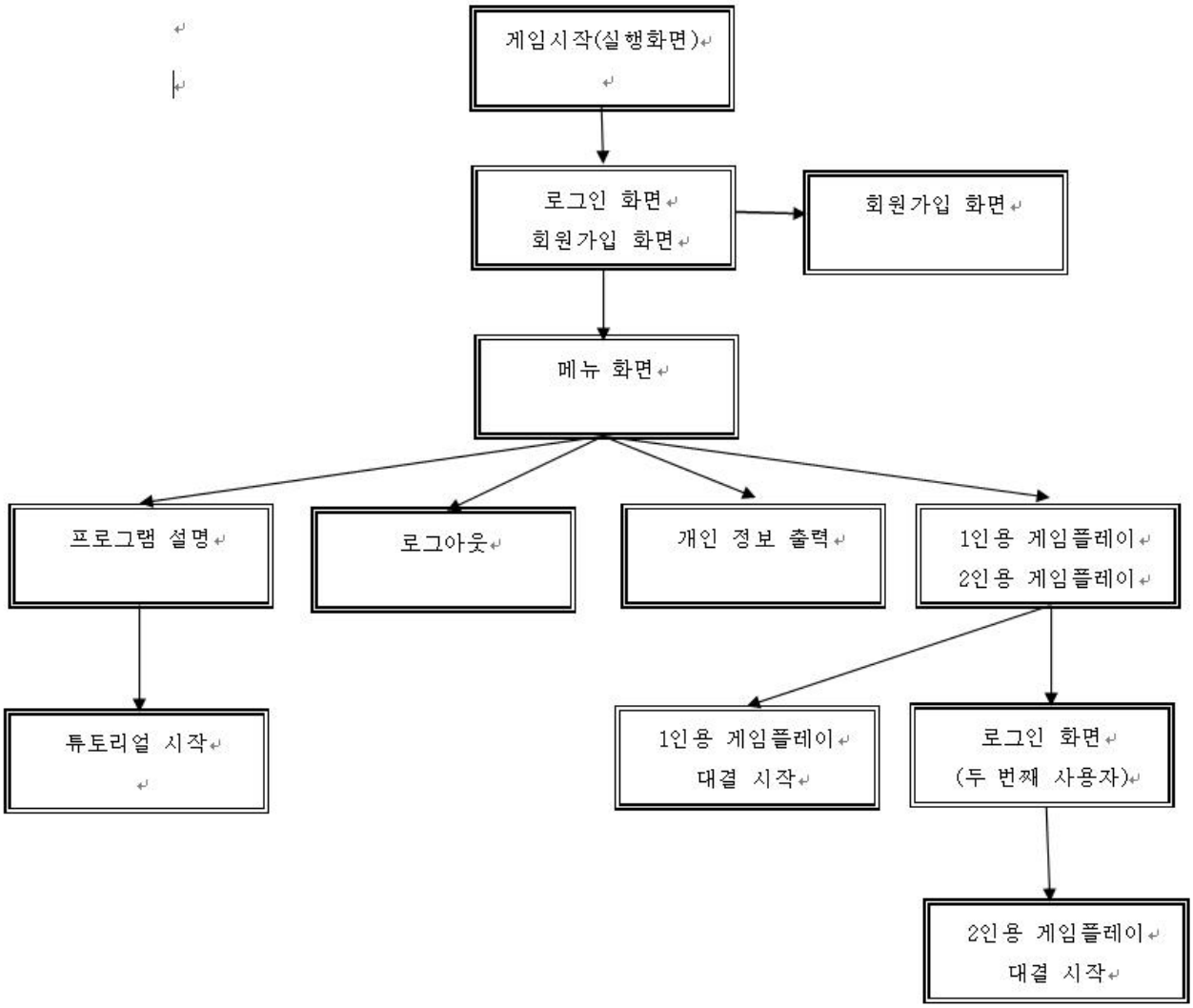
3.3.3 양회욱

식별번호	이름	입력데이터		출력데이터타입
		타입	이름	
2OTP-Y-001	check_board_complete	Board*	board	bool
2OTP-Y-002	check_board_mask	Board* Board*	Board Mask	bool
2OTP-Y-003	Game_table	-	-	void
2OTP-Y-004	draw_rectangle	int int	r c	void
2OTP-Y-005	Gotoxy	int	x,y	void
2OTP-Y-006	Move_control	Board* Board*	Board_A Board_B	void
4OTP-Y-007	draw_omok	void	-	void
4OTP-Y-008	print_board	Board* Board* int	Board_1 Board_2 n	void
4OTP-Y-009	Turn_two_players	int	n	int
5OTP-Y-010	Game_tutorial	void	-	void
5OTP-Y-011	Move_point	void	-	void

5OTP-Y-012	ThreadFunc	void *	data	DWORD WINAPI

*함수별 절차는 5.구현코드 참고

3.4 프로그램 흐름도



4. 구현 상세 내용

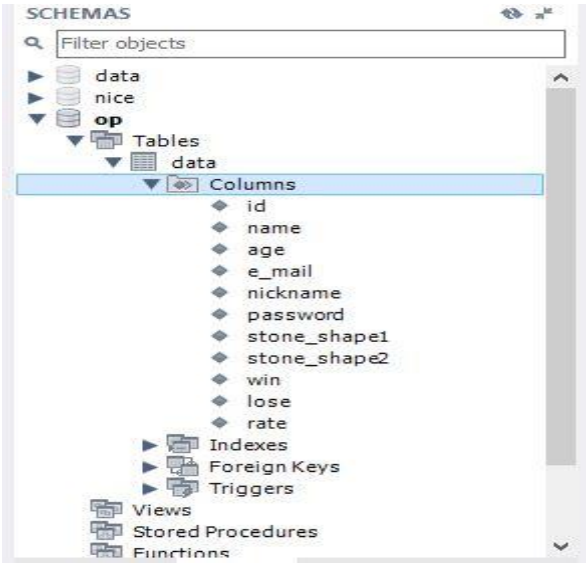
요구사항을 바탕으로 구현한 프로그램의 각각의 기능을 구현코드와 함께 상세하게 기술한다. '오목 트레이닝 프로그램'은 사용자 계정 관리 기능, 게임 플레이 공통 기능, 1 인용 게임 플레이 기능, 2 인용 게임 플레이 기능, 게임 설명 기능으로 크게 5 가지 기능으로 나누어진다.

4.1 사용자 계정 관리 기능

사용자 계정 관리 기능은 사용자가 오목 트레이닝 프로그램을 하기 위한 계정을 생성 및 관리하도록 도와준다. 이 기능은 데이터베이스를 사용하였으며 데이터베이스를 통해 만들었던 계정을 데이터베이스에 데이터를 저장을 하고 계정을 사용하고 싶을 때에는 데이터베이스에서 데이터를 가져와서 사용할 수 있게 되어있다.

4.1.1 DB구조체

계정을 만들기 위해서는 DB 구조체와 본 C 프로그램에서 사용하고 있는 데이터 구조가 같아야 한다. 시작하기 전에 User 구조체를 만들어서 DB 구조체를 먼저 만들어 한다.



DB 구조체를 만들고 DB 를 사용하기 위한 MYSQL 쿼리문을 알아야 한다. 사용할 MYSQL 쿼리문은 UPDATE, INSERT, SELECT 문이다. 회원가입에는 INSERT 문이 필요하다. USER 구조체에 대한 데이터를 입력을 받고, 입력받은 데이터를 DB 에 INSERT 를 하는 것이다.

4.1.2 INSERT

```
int insert(MYSQL *conn, _User user)
{
    char buff[255];
    sprintf(buff, "insert into data (name, age, e_mail, nickname, password, stone_shape1,ston
    printf("%s : %p: %d₩n", buff, conn, mysql_query(conn, buff));
    fprintf(stderr, "Mysql query error : %s ₩n", mysql_error(&conn));
}
```

MYSQL INSERT 문은 데이터를 삽입하는 데에 사용이 된다. C 프로그램에서 데이터를 입력을 받아서 DB 데이터에 삽입을 하는 것이다. 데이터를 입력을 받는 과정은 이러하다. 데이터는 사용자의 이름, 나이, 이메일, 아이디, 패스워드, 오목알의 모양까지 입력을 받습니다. 이렇게 입력받은 데이터는 DB 의 데이터에 저장이 된다.

4.1.3 SELECT

```
printf("아 이 디를   입력하시오 >>  ");
scanf("%s", nickname2);
gotoxy(35, 7);
printf("비밀번호를   입력하시오 >>  ");
scanf("%s", password2); //암호화실패

sprintf(buff2, "SELECT * FROM data WHERE nickname='%s'AND password ='%s'", nickname2, password2);
```

MYSQL SELECT 문은 로그인 하는 데에 사용하는 것이다. SELECT FROM nickname AND password 를 통해 아이디와 비밀번호가 일치하는 데이터를 가져오는 것이다. 아이디와 비밀번호가 일치한다면 로그인을 할 수 있다. 로그인을 하게 되면 기존에 있는 데이터에 DB 에 있는 데이터로 초기화 시켜서 이용 가능하게 된다.

4.1.4 UPDATE

MYSQL UPDATE 는 데이터가 저장이 되게 다시 DB 로 데이터를 저장하는 거다. 이것은 전적확인을 할 때 필요한 것이다. 전적은 게임 종료후에도 저장되어야 하기 때문에 DB 에 게임을 이겼을 때와 졌을 때 데이터를 그 때 그 때 DB 에 넣어줘야 한다.

```
if (timeover == 1)
{
    print_board(board_1, board_2, first_n);
    gotoxy(80, 13);
    printf("player1이 승리하셨습니다₩n");
    timeover = 0;
    user.win++;
    user2.lose++;
    sprintf(buff1, "update data set win = win + 1 where name = '%s'", user.name);
    mysql_query(conn, buff1);
    sprintf(buff1, "update data set lose = lose + 1 where name='%s'", user2.name);
    mysql_query(conn, buff1);
    break;
}
```

4.2 게임 플레이 공통 기능

4.2.1 비트보드 (Bitboard)

비트보드는 보드게임을 구현할 때 많이 쓰이는 자료구조중에 하나로 비트 하나하나가 의미를 가지도록 설계하는 것이 주 목적이다. '오목 트레이닝 프로그램'을 구현하면서 사용한 부분은 오목판에 오목알을 어디에 두었는지를 나타내기위해 사용하였다. 2 차원 배열을 이용하여 오목판을 구현하면 메모리 공간도 많이 사용할 뿐만아니라 인공지능을 개발할 때 각각의 트리의 노드가 오목판의 상태를 저장하고 있어야하기 때문에 효율적으로 저장하고 연산하기 위해서 비트보드 방식을 이용하였다.

```
#define ll unsigned long long

struct Board {
    ll OM1;
    ll OM2;
    ll OM3;
    ll OM4;
};
```

비트의 나열을 오목판이라고 가정하고 각각의 비트가 해당 오목판의 위치에 돌을 두었는지 두지 않았는지 결정한다. 값이 1 이면 돌이 있는 곳이고, 0 이면 아직 두지 않은 곳이다. 오목판의 크기가 15*15 로 64bit 변수가 저장하기에 부족하기 때문에 총 4 개의 변수를 이용하였고 각각을 편의상 OM1, OM2, OM3, OM4 라고 이름지어 오목판을 표현하였다. Board 라는 구조체로 나타내고 2 차원 배열처럼 익숙한 방법으로 사용하기 위해서 비트의 기본연산(or, and, xor, not 등)과 배열처럼 이용하기 위한 Random Access 함수(board_isSet, board_set 등)을 구현하여 이용하였다.

4.2.2 Gomoku 게임 규칙에 의한 승패결정

Gomoku 게임 규칙은 가장 간단한 오목 게임 규칙으로써 어떤 방법을 이용해서라도 5 개의 일렬을 만들기만 하면 된다. 다른 규칙들과 다르게 3x3 제한이나 6 목 제한과 같은 어떠한 제한도 존재하지 않는다. 복

잡한 규칙이 존재하지 않는다면 이는 비트보드를 이용하여 구현하기에 좋다.

```
bool check_board_complete(Board* board) {
    int i, j;
    bool isComplete = false;

    Board* boardHorizontal = makeBoard((ll)0, (ll)2181431069507584, (ll)0, (ll)0);  //가로 직선
    Board* boardReverseSlash = makeBoard((ll)0, (ll)281479271743488, (ll)65537, (ll)0);  //역슬래시
    Board* boardVertical = makeBoard((ll)0, (ll)9223653520421683200, (ll)16384, (ll)0); //세로 직선
    Board* boardSlash = makeBoard((ll)0, (ll)4611967510585016320, (ll)4096, (ll)0); //슬래시

    isComplete = check_board_mask(board, boardHorizontal);
    if (isComplete) return true;

    isComplete = check_board_mask(board, boardReverseSlash);
    if (isComplete) return true;

    isComplete = check_board_mask(board, boardVertical);
    if (isComplete) return true;

    isComplete = check_board_mask(board, boardSlash);
    if (isComplete) return true;

    return isComplete;
}
```

오목이 되는 경우는 크게 4 가지 경우가 있는데 이는 각각 가로로 직선이 이루어지는 경우, 세로로 직선이 이루어지는 경우, 슬래시와 같은 방향으로 이루어지는 경우, 역슬래시와 같은 방향으로 이루어지는 경우가 있다. 오목판의 정가운데를 중심으로 각각의 경우를 표현한 Board 변수 4 개를 위의 사진에서 볼 수 있다. Check_board_complete 함수로 들어온 Board 변수가 게임이 끝났는지 아닌지 판단하기 위해서 총 4 가지의 경우를 모든 오목판에 대해 적용을 해보는데 이는 각 변수를 좌로 또는 우로 shift 연산을 하는 것으로 모든 경우가 표현된다. 모든 경우를 확인해 보면서 하나라도 매칭되는 경우가 생긴다면 이는 오목이 만들어졌다는 의미이다.

4.2.3 결정시간(30초) 제한

사용자의 입력을 기다리면서 동시에 오목알을 두기까지의 제한시간을 계산하여 화면에 보여주기도 해야한다. 이를 실질적으로 구현하기 위해서는 쓰레드(Thread)라는 개념이 필요하다. 하나의 프로세스에서 동시에 여러 개여 작업이 필요한 경우 쓰레드를 생성하여 작업하면 되는데 c 언어에서는 함수단위로 실행할 수 있다. Time 함수를 이용하여 1 초가 지났을 때 화면에 출력하게 되는데 사용자의 입력을 방해하면 안되기 때문에 사용자가 입력하고 있는 터미널의 x, y 좌표를 기억하고 있다가 시간을 출력하고 바로 다시 돌아간다. 사용자가 입력을 수행하기 전에 쓰레드를 생성하고 입력을 종료했을 때 해당 쓰레드를 종료한다.

```
DWORD WINAPI ThreadFunc(void *data)
{
    count_time = 30;
    time_t startTime, endTime;

    startTime = time(NULL);
    while (1)
    {
        endTime = time(NULL);

        if (endTime - startTime >= 1) {
            startTime = endTime;
            count_time--;
            gotoxy(92, 11);

            printf("%d        ", count_time);
            gotoxy(x_c, y_c);

            if (count_time < 0) {
                gotoxy(92, 11);
                printf("시간초과!!!₩n");

                timeover++;

                return 0;
            }
        }
    }

    return 0;
}
```

4.2.4 방향키 이용

플레이어가 방향키를 이용하여 원하는 위치에 자신의 오목알을 둘 수 있도록 하였다.

```
case 72:
    y_c--;
    if (y_c < 1)
        y_c = 1;
    break;
case 75:
    x_c -= 2;
    if (x_c < 1)
        x_c = 1;
    break;
case 77:
    x_c += 2;
    break;
case 80:
    y_c++;
    break;
```

위와 같이 화살표 방향에 따라 x, y 좌표값을 수정하도록 하였다. 우리의 프로그램은 게임을 진행할 때,

오목판은 black_board, white_board 로 나눠져 있고 출력할 때는 두 board 를 합쳐 출력하는 형태로 프로그래밍 되어있다. 스페이스를 이용하면 오목알을 둘 수 있는데, 이때 고려해야할 부분은 오목알이 놓인 위치에 다시 오목알이 놓일 수 없다는 것이다.

```
case 32:
    if (board_isSet(board_A, (x_c - 22) / 2, y_c - 7) || board_isSet(board_B, (x_c - 22) / 2, y_c - 7))
        continue;
    *board_A = *(board_set(board_A, (x_c - 22) / 2, y_c - 7)); /*수를 놓았을때 표현*/
    put = 1;
    break;
```

위와 같이 스페이스를 눌렀을 때, 현재 놓으려는 위치에 black_board 와 white_board 둘 다 오목알이 놓여있지 않은 상태라면 board_set 함수를 통해 오목알을 놓아주고 while 문 탈출을 위해 put 에 1 을 넣어준다. 하지만 만약 두 board 중 한군데라도 오목알이 놓여있다고 나오면 아무 변화 없이 continue 를 해주도록 하였다. 최종적으로 플레이어가 턴을 넘겨받는 순간부터 제한시간(30 초)가 카운트 되기 위해서 thread 가 생성되는데 사용자가 스페이스를 눌러 오목알을 두면 다음과 같은 if 문 조건에 해당되어 thread 를 종료하게 된다.

```
if (key == 32)
{
    TerminateThread(thread, 0);
}
```

4.3 1인용 게임 플레이 기능

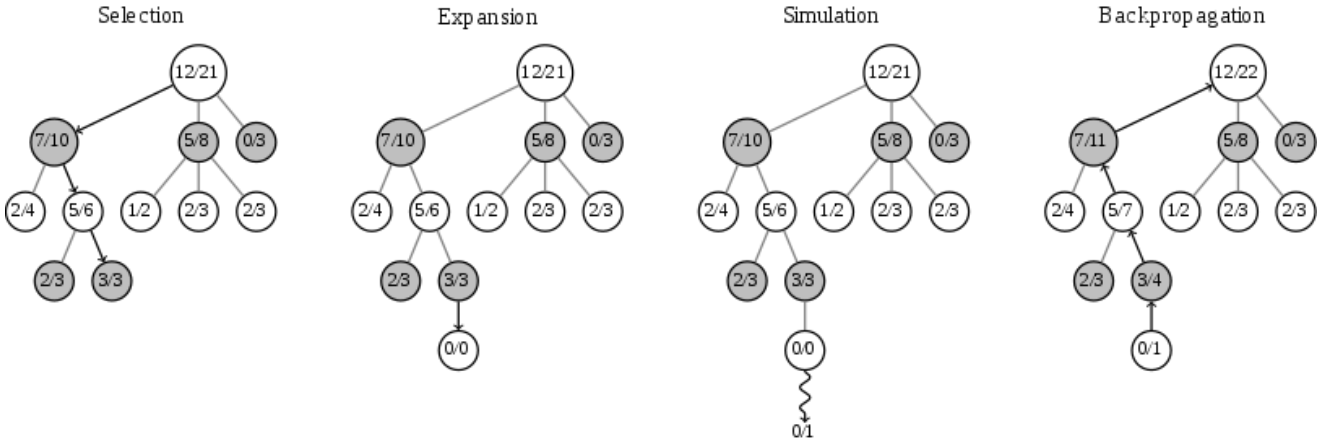
1 인용 게임 플레이 기능은 사용자가 타인이 없이 혼자 오목 트레이닝을 진행할 수 있도록 하기 위해서 학습하는 인공지능 컴퓨터와 대결할 수 있도록 도와준다. 이 기능은 Monte Carlo Simulation 이라는 기법이 이용되었으며 이 기법을 이용하여 MCTS(Monte Carlo Tree Search)로 컴퓨터의 의사결정을 진행한다. MCTS 과정 중에서 탐색할 공간을 줄이기 위해 Bound 라는 구조체를 만들어 범위를 설정해 주었다.

4.3.1 Monte Carlo Simulation

몬테카를로 방법은 난수를 이용하여 함수의 값을 확률적으로 계산하는 알고리즘을 말한다. 이 프로젝트에서 이 방법은 사용자가 둔 오목알의 배치를 기반으로 컴퓨터가 어디에 다음수를 두어야 할지 의사결정을 할 때 난수를 이용하여 결정한다. 오목판 전체를 난수의 대상으로 잡을 경우, 한가지의 경우를 고르기 위해 x 축과 y 축을 모두 선택해야 하기 때문에 15*15=225 가지의 경우의 수가 존재한다. 하지만 코드에 보이는 것처럼 범위를 조절하여 난수를 설정하였기 때문에 효율적으로 난수를 생성하고 이는 나쁘지 않는 휴리스틱 기법이다.

4.3.2 MCTS (Monte Carlo Tree Search)

몬테카를로 탐색 트리는 AlphaGo 에서 사용되었다고 알려져 화제가 되었던 알고리즘인데 몬테카를로 방법을 기반으로 트리를 만들어 경험적 탐색을 가능하게 한다. 이 트리는 총 4 가지의 단계를 거치면서 의사결정 트리를 만들어낸다. 아래 그림에서처럼 Selection, Expansion, Simulation, Backpropagation 으로 나누어진다.



Selection 단계에서는 root 에서 시작하여 어떤 노드를 탐색해야 할지 결정하는 역할을 한다. 여기에는 2 가지 경우가 있는데 새로운 경우를 탐색해야 하는 경우와 기존에 탐색했던 노드를 다시 탐색하는 경우로 나누어진다. 전자의 경우 새로운 노드를 만들어 주는 과정에 Expansion 과정이다. 적절한 경우를 선택한 후에 게임이 끝날때까지 시뮬레이션을 진행하는데 오목 게임에서는 오목이 만들어질때까지를 의미한다. 이 과정이 Simulation 과정이다. 승패가 결정되고 나서 그 결과를 선택된 노드의 조상노드들의 승리횟수와 게임 진행횟수를 증가시켜주는데 이과정을 Backpropagation 이라고 부른다.

```
int simulate(Node *node) {
    Node *randomMove = NULL;
    int result;

    if (check_board_complete(node->boardBlack)) {
        return 0;
    }
    else if (check_board_complete(node->boardWhite)) {
        return 1;
    }

    randomMove = selectHeuristic(node);
    if (randomMove == (Node*)NULL) {
        return 0;
    }
    else {
        result = simulate(randomMove);
        result = ((result == 0) ? 1 : 0);
        node->win = node->win + result;
        node->N = node->N + 1;
        return result;
    }
}
```

simulate 라는 함수로 구현된 MCTS 방식은 해당 함수에서 4 가지 기능을 모두 볼 수 있다. Select Heuristic 이라는 함수가 Select 와 Expansion 을 모두 수행한다. 랜덤하게 정해진 x, y 를 두는데 기존에 존재하는 경우이면 해당 자식노드를 찾아서 리턴해주고, 해당 자식노드를 찾지 못했으면 Expansion 과정을 통해 새로운 자식노드를 추가해준다. simulate 함수가 재귀적으로 호출되면서 게임이 끝날때까지 진행되고, 게임이 끝났을 때, 검은돌이 이겼으면 0 을, 흰돌이 이겼을때는 1 을 리턴해줌으로써 backpropagation 과정이 마무리된다.

4.3.3 Bound

```
struct Bound {
    int max_x;
    int min_x;
    int max_y;
    int min_y;
};
```

```
struct Node {
    Board *boardWhite;
    Board *boardBlack;
    Bound boundWhite;
    Bound boundBlack;
    int win;
    int N;
    Turn turn;
    NodeArray *nodes;
};
```

Bound 라는 구조체는 컴퓨터 인공지능이 몬테카를로 방법을 이용하여 새로운 수를 만들어낼 때 그 범위를 제한해주는 역할을 한다. 이는 경우의 수를 줄여주는 역할도 하지만 오목을 둘 때 휴리스틱한 방법으로 최적의 해를 찾을 수 있도록 도와준다. 여기서 하나의 가정이 필요한데 오목에서 다음 수를 생각할 때 이전까지 상대가 둔 수들에서 가까운 범위에만 그 결과가 존재한다는 가정이다. 이 프로젝트에서는 상대가 둔 수들의 범위에서 상하좌우로 1 칸씩 증가한 만큼을 Bound 구조체로 만들어 저장한다. 그리고 이를 활용하여 난수를 생성한 후에 MCTS 를 만들고 의사결정을 진행한다.

```
void setBound(Node* node, int x, int y, Turn turn) {
    if (turn == Black) {
        node->boundBlack.max_x = min(max(node->boundBlack.max_x, x + 1), 14);
        node->boundBlack.min_x = max(min(node->boundBlack.min_x, x - 1), 0);
        node->boundBlack.max_y = min(max(node->boundBlack.max_y, y + 1), 14);
        node->boundBlack.min_y = max(min(node->boundBlack.min_y, y - 1), 0);
    }
    else {
        node->boundWhite.max_x = min(max(node->boundWhite.max_x, x + 1), 14);
        node->boundWhite.min_x = max(min(node->boundWhite.min_x, x - 1), 0);
        node->boundWhite.max_y = min(max(node->boundWhite.max_y, y + 1), 14);
        node->boundWhite.min_y = max(min(node->boundWhite.min_y, y - 1), 0);
    }

}
```

이를 구현하기 위해 Node 구조체마다 Bound 구조체를 2 개씩(흰돌의 Bound, 검은돌의 Bound) 저장하였고, 매 수를 둘때마다 그 경계를 업데이트하는 것을 볼 수 있다.

4.4 2인용 게임 플레이 기능

2 인용 게임 플레이 기능은 새로운 사용자의 로그인 정보를 입력 받은 후, 두 플레이어가 대결을 할 수 있도록 한다. 게임을 시작하기 전 랜덤으로 두 사용자 중 먼저 플레이어 할 사용자를 정하고 먼저 두는 플레이어의 오목알을 흑, 나중에 두는 플레이어의 오목알을 백으로 설정하였다. 게임은 gomoku 룰에 의해 승패 여부가 결정되며, 승리한 플레이어는 전적에 승리 횟수가 1 증가하고 패배한 플레이어는 전적에 패배 횟수가 1 증가하도록 하였다.

4.4.1 Turn

게임이 시작되면 가장 먼저 순서를 정해야 한다.

```
srand(time(NULL));
n = rand() % 2;
first_n = n;
```

임의의 수를 변수 n 에 받아서 짝수인지 홀수인지 판별하여 플레이어의 순서를 정한다. 플레이어의 오목알 색깔은 stoneshape1 과 stoneshape2 에 저장되어 있는데 백이 1, 흑이 2 로 되어있다. 처음 나온 수가 홀수인지 짝수인지에 따라 오목알 색깔이 결정되기 때문에 첫 숫자를 first_n 이라는 변수에 저장해준다. 턴은

```
int turn_two_players(int n)
{
    if (n % 2 == 1) // 홀수
        return 1;

    else          // 짝수
        return 0;
}
```

함수를 통해서 변화를 주는데

```
gotoxy(80, 10);
printf("player1의 차례입니다. ");
gotoxy(80, 11);
printf("제한시간 : ");
move_control(board_1, board_2);
n++;
```

플레이어가 오목알을 두면 n++가 되면서 n 이 지속적으로 짝수에서 홀수, 홀수에서 짝수로 변화한다. 그럼 이 n 값을 turn_two_players 함수에 넣어주고 return 값에 따라서 플레이어의 순서가 결정되게 된다.

4.4.2 게임결과

플레이어가 게임에서 gomouk 룰에 의해서 이길 수 있는 방법은 두가지다. 첫번째는 오목알을 5 개 놓는 것, 두번째는 상대방이 제한시간(30 초)내에 오목알을 놓지 못했을 경우 승리하게 된다. 두 플레이어가 게임을 하는 동안 지속적으로 룰을 체크하여 게임을 계속 진행할 지, 승패가 결정되었는지 알 수 있도록 한다. 먼저 플레이어가 오목알을 둘 때 마다 체크하도록 다음과 같이 하였다.

```
if (check_board_complete(board_1) == true)
{
    print_board(board_1, board_2, first_n);
    gotoxy(80, 13);
    printf("player1이 승리하셨습니다\n");
    user.win++;
    user2.lose++;
    sprintf(buff1, "update data set win = win + 1 where name = '%s'", user.name);
    mysql_query(conn, buff1);
    sprintf(buff1, "update data set lose = lose + 1 where name='%s'", user2.name);
    mysql_query(conn, buff1);
    break;
}

if (check_board_complete(board_2) == true)
{
    print_board(board_1, board_2, first_n);
    gotoxy(80, 13);
    printf("player2이 승리하셨습니다\n");
    user2.win++;
    user.lose++;
    sprintf(buff1, "update data set lose = lose + 1 where name = '%s'", user.name);
    mysql_query(conn, buff1);
    sprintf(buff1, "update data set win = win + 1 where name='%s'", user2.name);
    mysql_query(conn, buff1);
    break;
}
```

그림에서 보면 플레이어가 오목알을 놓은 후, check_board_complete 함수를 통해 오목알이 5 개가 놓인 위치가 있는지 판별해주고 있다. 만약 두 플레이어 중 오목알이 5 개가 놓인 플레이어가 존재할 경우, “승리하였습니다”를 출력하고 플레이어들의 전적의 승패를 수정하고 게임을 종료하여 준다. 다음은 상대 플레이어가 제한시간 내에 오목알을 놓지 못했을 경우를 보여준다.

```
if (timeover == 1)
{
    print_board(board_1, board_2, first_n);
    gotoxy(80, 13);
    printf("player1이 승리하셨습니다\n");
    timeover = 0;
    user.win++;
    user2.lose++;
    sprintf(buff1, "update data set win = win + 1 where name = '%s'", user.name);
    mysql_query(conn, buff1);
    sprintf(buff1, "update data set lose = lose + 1 where name='%s'", user2.name);
    mysql_query(conn, buff1);
    break;
}
```

시간을 관리하는 thread 함수에서 30 초가 카운트 되면 전역변수로 지정되어 있던 timeover 함수에 1 을 넣어준다. 게임 중 30 초가 다 카운트 되는 경우, timeover 에 1 이 들어가면서 if 문에 들어가게 되고, “승리하였습니다”를 출력 후 timeover 함수를 다시 0 으로 초기화 시켜주고 두 플레이어의 전적을 수정해준다. 그리고 게임을 종료해준다.

4.5 게임 설명 기능

사용자가 게임 플레이를 하기 전, 프로그램의 간단한 설명을 듣고 직접 연습 플레이를 해볼 수 있도록 하였다. 실제 게임 플레이를 할 때 사용되는 오목판과 상황판을 재현해 놓았으며 간단한 설명이 진행 된 후, 스페이스를 이용하여 오목알 두기, 게임에서 승리하기 위해 오목알을 5 줄로 만들어 보기 순으로 튜토리얼이 진행되도록 하였다. 튜토리얼을 마치면 사용자가 게임을 하는데 큰 어려움이 없을 것이다.

4.5.1 튜토리얼

튜토리얼은 getch()를 이용하여 사용자의 입력을 받아 다음 상황으로 넘어가도록 하였다.

```
void Game_Tutorial(Board* board)
{
    int num, i, j;

    printf("안녕하세요 튜토리얼을 시작하겠습니다.\n");
    getch();
    system("cls");
    printf("먼저 게임기능에 대해 설명드리겠습니다.\n\n");

    game_table_tutorial(board);

    gotoxy(53, 13);     //gotoxy문을 사용해서 화살표로 오목판을 가리킨다
    printf("↖");

    gotoxy(53, 14);
    printf("이것은 오목판 입니다\n");
    gotoxy(53, 15);
    printf("여기에서 오목게임이 진행이 되어집니다.\n");

    getch();
    system("cls");

    printf("\n\n");

    game_table_tutorial(board);
```

gotoxy 문을 이용하여 적절한 위치에서 설명하는 텍스트가 출력될 수 있도록 하였고, 프로그램에 대한 설명이 차례대로 이루어지고 난 뒤, 직접 오목판에 놓는 연습을 해보도록 하였다. 아래의 그림에서는 플레이어가 오목알을 두도록 구현한 것이다.

```
do
{
    move_control_tuto(board);

    draw_omok();

    for (i = 0; i < MAX_LINE; i++)
    {
        for (j = 0; j < MAX_LINE; j++)
        {
            gotoxy(i * 2 + 22, j + 7);

            if (board_isSet(board, i, j))
                printf("●");

        }
    }
} while (check_board_complete(board) != true);
```

플레이어의 오목알이 연속적으로 5 개가 놓일 때까지 오목알을 두도록 하였다.

```
    gotoxy(80, 18);
    printf("잘하셨습니다.\n");
    gotoxy(80, 19);
    printf("이렇게 다섯알을 놓으면");
    gotoxy(80, 20);
    printf("게임에서 승리하게 됩니다.\n");

    getch();
    system("cls");
    gotoxy(30, 8);
    printf("게임에 대한 이해가 조금 되셨나요?\n");
    gotoxy(30, 10);
    printf("수고하셨습니다^^\n");

    gotoxy(1, 20);

}
```

오목알을 5 개 놓으면 while 문에서 나오게 되고, 다음과 같은 화면이 뜨고 난 뒤, 튜토리얼은 종료된다.

5. 구현코드

5.1 Types.h

```
#pragma once

#define MAX_LINE 15
#define MAX_BIT_CNT 64
#define NODE_ARRAY_INIT_SIZE 1
#define MAX_SEARCH_TIME_IN_SEC 5
#define ll unsigned long long

typedef enum { false, true } bool;
typedef enum { Black, White } Turn;
typedef struct Board Board;
typedef struct Node Node;
typedef struct NodeArray NodeArray;
typedef struct Bound Bound;

struct Board {
    ll OM1;
    ll OM2;
    ll OM3;
    ll OM4;
};

struct NodeArray {
    Node* array;
    size_t size;
    size_t allocated;
};

struct Bound {
    int max_x;
    int min_x;
    int max_y;
    int min_y;
};

struct Node {
```

```
    Board *boardWhite;
    Board *boardBlack;
    Bound boundWhite;
    Bound boundBlack;
    int win;
    int N;
    Turn turn;
    NodeArray *nodes;
};

typedef struct user {
    unsigned int id; //32   //사용자를 식별하기 위한 정보
    char name[20];              //사용자의 이름
    unsigned int age; //16 //사용자의 나이
    char e_mail[30];           //사용자의 이메일
    char nickname[20];         //사용자의 아이디
    char password[20];         //사용자의 패스워드
    char stone_shape1[10]; //사용자의 오목알백모양
    char stone_shape2[10]; //사용자의 오목알흑모양
    unsigned int win;          //사용자의 전적 승
    unsigned int lose;         //사용자의 전적 패
    unsigned int rate;                 //사용자의 승률
}_User;
```

5.2 Omok.c

```
#include <mysql.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <time.h>
#include "Types.h"

#define DB_HOST "localhost"
#define DB_USER "root"
#define DB_PASS "tmdwls7127"
```

```
#define DB_NAME "op"
#define CHOP(x) x[strlen(x) - 1] = ' '

_User user;
_User user2;
int timeover;

int login2(MYSQL *conn); //두번째 고객 로그인
int Calculate_rate(int a, int b); //승률 계산 함수
void Print_Information(); //개인 정보 출력
void end_game_screen(); //게임 종료 스크린
void second_menu(); //두번째 메뉴 스크린
void input_data(_User *User); //회원가입
int insert(MYSQL *, _User User);
int login(MYSQL *); //로그인함수
int print(MYSQL *); //전체 고객 출력
int delete(MYSQL *); //회원탈퇴
void gotoxy(int x, int y); //커서 이동
void first_screen(); //첫번째 스크린
void second_screen(); //두번째 스크린
void program_screen(); //프로그램 설명 스크린
void game_screen(); //게임 설명 스크린
void logout_fail_screen(); //로그인 실패 스크린
void logout_screen();//로그아웃 스크린
void zero_screen(); //제로 스크린

Board* makeEmptyBoard();
Board* makeBoard(ll A, ll B, ll C, ll D);
Board* board_mask(Board* board);
Board* board_shift_right(Board* board, int shift);
Board* board_shift_left(Board* board, int shift);
Board* board_and(Board* baseBoard, Board* layerBoard);
Board* board_or(Board* baseBoard, Board* layerBoard);
Board* board_xor(Board* baseBoard, Board* layerBoard);
Board* board_not(Board* board);
Board* board_copy(Board* board);
Board* board_set(Board* board, int i, int j);
bool board_isSet(Board* board, int i, int j);
bool board_isEmpty(Board* board);
```

```
bool board_isEqual(Board* boardA, Board* boardB);
void board_find_diff_one(Board* bigger, Board* smaller, int *x, int *y);
bool check_board_complete(Board* board);
bool check_board_mask(Board* board, Board* mask);

NodeArray* narray_init();
void narray_insert(NodeArray *a, Node element);
NodeArray* narray_copy(NodeArray *target);
void narray_free(NodeArray *a);

void MCTS(Node *root);
int simulate(Node *node);
Node* selectHeuristic(Node *root);

Node* node_init(Node* node);
Node* node_copy(Node* target, bool liveChild);
void setBound(Node* node, int x, int y, Turn turn);

void ready();
Node* findNext(Node* node);
Node* findChildSameBoard(Node* node, Board* black, Board* white);
Node *realRoot = NULL, *tmpRoot, *toFind;

void game_table(MYSQL *conn);
void game_table_com();
void draw_rectangle(int r, int c);
void print_board_com(Board* boardBlack, Board* boardWhite);
void print_board(Board* board_1, Board* board_2, int n);
int turn_two_players(int n);
void Game_Tutorial(Board* board);
void game_table_tutorial(Board *board);
void move_point();
void move_control_tuto(Board* board);
void draw_omok();
void move_control(Board* board_A, Board* board_B);
////////////
int x_c, y_c;
int count_time;
```

```
DWORD WINAPI ThreadFunc(void *data);
HANDLE thread;

char password1[20];
char nickname1[20];

char password2[20];
char nickname2[20];

int main(void)
{
	Board* board = makeBoard((ll)0, (ll)0, (ll)0, (ll)0);

	int logout_count = 0;
	int logout_key; //로그아웃 키
	int input_menu3; //네번째 메뉴 입력
	int input_menu2; //세번째 메뉴 입력
	int input_menu1; //두번째 메뉴 입력
	int first_menu; //첫번째 메뉴 입력
	int i, n;

	MYSQL		*connection = NULL, conn;
	MYSQL_RES	*sql_result;
	MYSQL_ROW	row;
	int field;
	int query_stat;
	mysql_init(&conn);
	int query[255];

	connection = mysql_real_connect(&conn, DB_HOST, DB_USER, DB_PASS, DB_NAME, 3306, (char *)NULL,
0);
	if (connection == NULL) {
		printf("connect error!\n");
		exit(1);
	}
	zero_screen();
	first_screen(); //첫번째 화면 -> 게임시작
	while (1) {
		second_screen();//두번째 화면 -> 메뉴
```

```
gotoxy(35, 21);
printf("메뉴를 입력해주십시오 >>     ");
scanf("%d", &first_menu);
switch (first_menu) {
case 1: //회원가입
        system("cls");
        input_data(&user);
        printf("%p\n", connection);
        insert(connection, user);
        system("cls");
        break;
case 2: //로 그 인
        system("cls");
        if (login(connection) != 0) {
                system("cls");
                break;
        }
        system("cls");
        while (1) {
                system("cls");
                second_menu();
                gotoxy(35, 23);
                printf("메뉴를 입력해주십시오 >>     ");
                scanf("%d", &input_menu1);
                switch (input_menu1) {
                case 1: //프로그램 설명
                        system("cls");
                        program_screen();
                        gotoxy(35, 21);
                        printf("메뉴를 입력해주십시오 >>     ");
                        scanf("%d", &input_menu2);
                        switch (input_menu2) {
                        case 1: //튜토리얼 시작
                                system("cls");
                                Game_Tutorial(board);
                                //printf("튜토리얼 시작\n");
                                getch();
                                break;
                        case 2: //뒤로 돌아가기
```

```
                        break;
                default:
                        break;
                } // end switch
                break;
        case 2: //개인 정보 출력
                system("cls");
                Print_Information();
                printf("\n\n");
                getch();
                system("cls");
                break;
        case 3: //1 인용 / 2 인용 게임플레이
                system("cls");
                game_screen();
                gotoxy(35, 21);
                printf("메뉴를 입력해주십시오 >>     ");
                scanf("%d", &input_menu2);
                switch (input_menu2) {
                case 1: //1 인용 게임플레이
                        system("cls");
                        system("cls");
                        game_table_com();
                        //printf("1 인용 게임플레이\n");
                        getch();
                        break;
                case 2: //2 인용 게임플레이
                        system("cls");
                        if (login2(connection) != 0) {
                                system("cls");
                                break;
                        }
                        system("cls");
                        game_table(connection);
                        user.rate = Calculate_rate(user.win, user.lose);
                        getch();
                        break;
                case 3: //뒤로 돌아가기
                        break;
```

```
                        default:
                                break;
                        }
                        break;

                case 4: //로그아웃창
                        logout_count = 1;
                        break;
                default:
                        break;
                }
                if (logout_count == 1) {
                        system("cls");
                        logout_count = 0;
                        gotoxy(35, 5);
                        printf("오 목 프 로 그
램");
                        gotoxy(100, 9);
                        printf("by 으라차차");
                        gotoxy(35, 15);
                        printf("로그아웃을 하시겠습니까?(로그아웃 : 1 번) >> ");
                        scanf("%d", &logout_key);
                        if (logout_key == 1) {
                                logout_screen();
                                break;
                        }
                }
        }
        system("cls");
        break;
case 3: //종 료
        system("cls");
        end_game_screen();
        //system("cls");
        //zero_screen();
        return 1;
        break;
default:
        system("cls");
```

```
                        break;
                }
        }
        return 0;
}
void logout_screen() {//로그아웃 스크린
        system("cls");
        gotoxy(35, 5);
        printf("오        목        프        로        그        램");
        gotoxy(100, 9);
        printf("by 으라차차");
        gotoxy(39, 15);
        printf("로 그 아 웃 되 었 습 니 다");
        gotoxy(39, 18);
        printf("회원가입 로그인 창으로 이동하겠습니다.");
        printf("\n\n");
        getch();
}
void Print_Information() {
        gotoxy(35, 5);
        printf("오        목        프        로        그        램");
        gotoxy(100, 9);
        printf("by 으라차차");
        gotoxy(30, 15);
        printf("고객님의 이름 : %s\n", user.name);
        gotoxy(30, 17);
        printf("고객님의 나이 : %d\n", user.age);
        gotoxy(30, 19);
        printf("고객님의 아이디 : %s\n", user.nickname);
        gotoxy(30, 21);
        printf("고객님의 이메일 : %s\n", user.e_mail);
        gotoxy(30, 23);
        printf("고객님의 오목알 모양 : %s / %s\n", user.stone_shape1, user.stone_shape2);
        gotoxy(30, 25);
        printf("고객님의 승 : %d / 패 : %d / 승률 : %d\n", user.win, user.lose, user.rate);
}
void logout_fail_screen() {
        system("cls");
        gotoxy(35, 5);
```

```
		printf("오		목		프		로		그		램");
		gotoxy(100, 9);
		printf("by 으라차차");
		gotoxy(35, 15);
		printf("아이디 또는 비밀번호를 다시 확인하세요  \n");
		gotoxy(35, 18);
		printf("기존에 등록되지 않은 아이디이거나, 아이디 또는 비밀번호를 잘못 입력하셨습니다  \n");
		gotoxy(35, 21);
		printf("다시 입력해주십시오  ");
		printf("\n\n");
		getch();
		system("cls");
}
void end_game_screen() {
		gotoxy(35, 5);
		printf("오		목		프		로		그		램");
		gotoxy(39, 15);
		printf("이 용 해 주 셔 서    감 사 합 니 다 ^^");
		gotoxy(90, 20);
		printf("팀 명 : 으  라  차  차");
		gotoxy(90, 22);
		printf("팀 원 : 최 승 진");
		gotoxy(90, 24);
		printf("		양 회 욱");
		gotoxy(90, 26);
		printf("		오 상 헌");
		printf("\n\n");
}
void game_screen() {
		int menu;
		gotoxy(35, 5);
		printf("오		목		프		로		그		램");
		gotoxy(100, 9);
		printf("by 으라차차");
		gotoxy(35, 15);
		printf("1 번 째 메 뉴		>>		1 인용 게임플레이");
		gotoxy(35, 17);
		printf("2 번 째 메 뉴		>>		2 인용 게임플레이");
}
```

```
void program_screen() {
	int menu;
	gotoxy(35, 5);
	printf("오	목	프	로	그	램");
	gotoxy(100, 9);
	printf("by 으라차차");
	gotoxy(35, 15);
	printf("1 번 째 메 뉴	>>	튜토리얼 시작");
	gotoxy(35, 17);
	printf("2 번 째 메 뉴	>>	뒤로 돌아가기");
}
void second_menu() {
	int menu;
	gotoxy(35, 5);
	printf("오	목	프	로	그	램");
	gotoxy(100, 9);
	printf("by 으라차차");
	gotoxy(35, 15);
	printf("1 번 째 메 뉴	>>	프로그램 설명");
	gotoxy(35, 17);
	printf("2 번 째 메 뉴	>>	개인 정보 출력");
	gotoxy(35, 19);
	printf("3 번 째 메 뉴	>>	1 인용 / 2 인용 게임플레이");
	gotoxy(35, 21);
	printf("4 번 째 메 뉴	>>	로 그 아 웃");
}
void first_screen() {
	gotoxy(35, 5);
	printf("오	목	프	로	그	램");
	gotoxy(90, 20);
	printf("팀 명 : 으 라 차 차");
	gotoxy(90, 22);
	printf("팀 원 : 최 승 진");
	gotoxy(90, 24);
	printf("	양 회 욱");
	gotoxy(90, 26);
	printf("	오 상 헌");
	printf("\n\n");
	getch();
```

```
	system("cls");
}
void second_screen() {
	int menu;
	gotoxy(35, 5);
	printf("오 목 프 로 그 램");
	gotoxy(100, 9);
	printf("by 으라차차");
	gotoxy(35, 15);
	printf("1 번 째 메 뉴 >> 회 원 가 입");
	gotoxy(35, 17);
	printf("2 번 째 메 뉴 >> 로 그 인");
	gotoxy(35, 19);
	printf("3 번 째 메 뉴 >> 종 료");
	printf("\n\n");
}
void gotoxy(int x, int y) {
	COORD Pos = { x - 1,y - 1 };
	SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Pos);
}
void input_data(_User *user) // 회원가입
{
	int shape_number;
	gotoxy(35, 2);
	printf("사용자의 이름을 입력하시오 : ");
	scanf("%s", user->name);
	gotoxy(35, 4);
	printf("사용자의 나이을 입력하시오 : ");
	scanf("%d", &user->age);
	gotoxy(35, 6);
	printf("사용자의 이메일을 입력하시오 : ");
	scanf("%s", user->e_mail);
	gotoxy(35, 8);
	printf("사용자의 아이디을 입력하시오 : ");
	scanf("%s", &user->nickname);
	gotoxy(35, 10);
	printf("사용자의 패스워드을 입력하시오 : ");
	scanf("%s", &user->password);
	gotoxy(35, 12);
```

```
		printf("1. ○  2. ☆  3. ◇  4. ▽\n");
		gotoxy(35, 14);
		printf("사용자의 오목알모양에 해당되는 번호를 입력해주세요 : ");
		scanf("%d", &shape_number);
		switch (shape_number) {
		case 1:
			strcpy(user->stone_shape1, "○");
			strcpy(user->stone_shape2, "●");
			break;
		case 2:
			strcpy(user->stone_shape1, "☆");
			strcpy(user->stone_shape2, "★");
			break;
		case 3:
			strcpy(user->stone_shape1, "◇");
			strcpy(user->stone_shape2, "◆");
			break;
		case 4:
			strcpy(user->stone_shape1, "▽");
			strcpy(user->stone_shape2, "▼");
			break;
		default:
			break;

		}
}
int insert(MYSQL *conn, _User user)
{
		char buff[255];
		sprintf(buff, "insert into data (name, age, e_mail, nickname, password, stone_shape1,stone_shape2, win, lose, rate) values ('%s', '%d', '%s', '%s','%s','%s','%s', 0, 0, 0);", user.name, user.age, user.e_mail, user.nickname, user.password, user.stone_shape1, user.stone_shape2);
		printf("%s : %p: %d\n", buff, conn, mysql_query(conn, buff));
		fprintf(stderr, "Mysql query error : %s \n", mysql_error(&conn));
}
int login(MYSQL *conn)
{
		int offset = 0;
		int j;
```

```
char buff2[255];
int state;

MYSQL_RES    *sql_result;
MYSQL_ROW    row;
int field;
int query_stat;
int id_number;
gotoxy(35, 5);
printf("아 이 디를    입력하시오 >>  ");
scanf("%s", nickname1);
gotoxy(35, 7);
printf("비밀번호를    입력하시오 >>  ");
scanf("%s", password1); //암호화실패

sprintf(buff2, "SELECT * FROM data WHERE nickname='%s'AND password ='%s'", nickname1, password1);
if (mysql_query(conn, "USE op")) {
        printf("%s\n", mysql_error(conn));
        return 1;
}

if (mysql_query(conn, buff2)) {
        printf("%s\n", mysql_error(conn));
        return 1;
}
sql_result = mysql_store_result(conn);

if (mysql_affected_rows(conn) != 1) { // 로그인 처리
        logout_fail_screen();
        return 1;
}
field = mysql_num_fields(sql_result);

while ((row = mysql_fetch_row(sql_result))) {
        for (j = 0; j < field; j++) {
                switch (j) {
                case 0:
                        user.id = row[0];
                        break;
```

```
            case 1:
                    strcpy(user.name, row[1]);
                    break;
            case 2:
                    user.age = atoi(row[2]);
                    break;
            case 3:
                    strcpy(user.e_mail, row[3]);
                    break;
            case 4:
                    strcpy(user.nickname, row[4]);
                    break;
            case 5:
                    strcpy(user.password, row[5]);
                    break;
            case 6:
                    strcpy(user.stone_shape1, row[6]);
                    break;
            case 7:
                    strcpy(user.stone_shape2, row[7]);
                    break;
            case 8:
                    user.win = atoi(row[8]);
                    break;
            case 9:
                    user.lose = atoi(row[9]);
                    break;
            case 10:
                    user.rate = atoi(row[10]);
                    break;
            default:
                    break;

            }
    }
    user.rate = Calculate_rate(user.win, user.lose);
    //printf("%12s",row[j]);

    //printf("\n");
```

```
        }
        //mysql_free_result(sql_result);
        return 0;
}
int login2(MYSQL *conn)
{
        int offset = 0;
        int j;
        char buff2[255];
        int state;

        MYSQL_RES     *sql_result;
        MYSQL_ROW     row;
        int field;
        int query_stat;
        int id_number;
        gotoxy(35, 5);
        printf("아 이 디를     입력하시오 >>   ");
        scanf("%s", nickname2);
        gotoxy(35, 7);
        printf("비밀번호를     입력하시오 >>   ");
        scanf("%s", password2); //암호화실패

        sprintf(buff2, "SELECT * FROM data WHERE nickname='%s'AND password ='%s'", nickname2, password2);
        if (mysql_query(conn, "USE op")) {
                printf("%s\n", mysql_error(conn));
                return 1;
        }

        if (mysql_query(conn, buff2)) {
                printf("%s\n", mysql_error(conn));
                return 1;
        }
        sql_result = mysql_store_result(conn);

        if (mysql_affected_rows(conn) != 1) { //  로그인 처리
                logout_fail_screen();
                return 1;
        }
```

```
field = mysql_num_fields(sql_result);

while ((row = mysql_fetch_row(sql_result))) {
        for (j = 0; j < field; j++) {
                switch (j) {
                case 0:
                        user2.id = row[0];
                        break;
                case 1:
                        strcpy(user2.name, row[1]);
                        break;
                case 2:
                        user2.age = atoi(row[2]);
                        break;
                case 3:
                        strcpy(user2.e_mail, row[3]);
                        break;
                case 4:
                        strcpy(user2.nickname, row[4]);
                        break;
                case 5:
                        strcpy(user2.password, row[5]);
                        break;
                case 6:
                        strcpy(user2.stone_shape1, row[6]);
                        break;
                case 7:
                        strcpy(user2.stone_shape2, row[7]);
                        break;
                case 8:
                        user2.win = atoi(row[8]);
                        break;
                case 9:
                        user2.lose = atoi(row[9]);
                        break;
                case 10:
                        user2.rate = Calculate_rate(user2.win, user2.lose);
                        break;
                default:
```

```
                                        break;

                                }
                        }

                        //printf("%12s",row[j]);

                        //printf("\n");
                }
                //mysql_free_result(sql_result);
                return 0;
}
int Calculate_rate(int a, int b) {//a = 승, b= 패
                int result = 0;
                if (a == 0 && b == 0)
                        return 0;
                else if (a == 0 && b>0)
                        return 0;
                else if (a > 0 && b == 0)
                        return 100;
                else {
                        result = (a * 100) / (a + b);
                        return result;
                }
}
int delete(MYSQL *conn) {
                char buff[255];
                char nickname[30];
                printf("삭제할 아이디를     입력하시오 >>   ");
                scanf("%s", nickname);
                sprintf(buff, "DELETE from data where nickname = '%s'", nickname);
                printf("%s : %p: %d\n", buff, conn, mysql_query(conn, buff));
                //printf("%s\n", mysql_error(conn));
}
int print(MYSQL *conn)
{
                int k;
                MYSQL_RES *sql_result;
                MYSQL_ROW row;
```

```
	int field;
	mysql_query(conn, "select * from data");
	sql_result = mysql_store_result(conn);
	field = mysql_num_fields(sql_result);
	printf("%12s%12s%12s%12s%12s%12s%12s%12s%12s%12s%12s\n", "id", "name", "age", "e_mail",
"nickname", "password", "shape1", "shape2", "win", "lose", "rate");
	while ((row = mysql_fetch_row(sql_result))) {
		for (k = 0; k<field; k++)
			printf("%12s", row[k]);
		printf("\n");
	}
}
void zero_screen() {

	gotoxy(34, 5);
	printf("■■■■■■■"); printf("      ■■■■■■■     "); printf("    ■■■■■■■\n");
	gotoxy(34, 6);
	printf("■          ■"); printf("          ■          ");     printf("    ■          ■\n");
	gotoxy(34, 7);
	printf("■          ■"); printf("          ■          ");       printf("    ■          ■\n");
	gotoxy(34, 8);
	printf("■          ■"); printf("          ■          ");     printf("    ■■■■■■■\n");
	gotoxy(34, 9);
	printf("■          ■"); printf("          ■          ");     printf("    ■\n");
	gotoxy(34, 10);
	printf("■          ■"); printf("          ■          ");     printf("    ■\n");
	gotoxy(34, 11);
	printf("■          ■"); printf("          ■          ");     printf("    ■\n");
	gotoxy(34, 12);
	printf("■■■■■■■"); printf("          ■          ");     printf("    ■\n");


	gotoxy(1, 25);
	printf("                                                                          ●●
");    printf("  ●●\n");
	gotoxy(1, 26);
	printf("●●●●
"); printf("●      ●\n");
	gotoxy(1, 27);
```

```
            printf("●●●●
"); printf("●       ●\n");
            gotoxy(1, 28);
            printf("                                                                    ●●
");     printf("  ●●\n");
            gotoxy(1, 29);
            getch();
            system("cls");
}


void draw_rectangle(int r, int c)
{
            int i, j;
            unsigned char a = 0xa6;
            unsigned char b[7];

            for (i = 1; i < 7; i++)
                        b[i] = 0xa0 + i;

            printf("%c%c", a, b[3]);

            for (i = 0; i < c; i++)
                        printf("%c%c", a, b[1]);

            printf("%c%c", a, b[4]);
            printf("\n");

            for (i = 0; i < r; i++)
            {
                        printf("%c%c", a, b[2]);

                        for (j = 0; j < c; j++)
                                    printf("    ");

                        printf("%c%c", a, b[2]);
                        printf("\n");
            }
```

```
		printf("%c%c", a, b[6]);

		for (i = 0; i<c; i++)
			printf("%c%c", a, b[1]);

		printf("%c%c", a, b[5]);
		printf("\n");
}
int turn_two_players(int n)
{
		if (n % 2 == 1) //  홀수
			return 1;

		else		//  짝수
			return 0;
}


void game_table_tutorial(Board *board)
{
		srand(time(NULL));
		int n;
		char stone[3];
		int i, j;

		draw_rectangle(25, 35);

		draw_omok();

		for (i = 0; i < MAX_LINE; i++)
		{
			for (j = 0; j < MAX_LINE; j++)
			{
				gotoxy(i * 2 + 22, j + 7);

				if (board_isSet(board, i, j))
					printf("●");
			}
		}
```

```
	gotoxy(80, 7);
	printf("player1 : 알파고 (●)\n");
	gotoxy(80, 8);
	printf("player2 : 이세돌 (○)\n");
	gotoxy(80, 9);
	printf("==============================\n");
	gotoxy(80, 10);
	printf("player1 의 차례입니다. ");
	gotoxy(80, 11);
	printf("제한시간 : 30");
}

void Game_Tutorial(Board* board)
{
	int num, i, j;

	printf("안녕하세요 튜토리얼을 시작하겠습니다.\n");
	getch();
	system("cls");
	printf("먼저 게임기능에 대해 설명드리겠습니다.\n\n");

	game_table_tutorial(board);

	gotoxy(53, 13);		//gotoxy 문을 사용해서 화살표로 오목판을 가리킨다
	printf("↖");

	gotoxy(53, 14);
	printf("이것은 오목판 입니다\n");
	gotoxy(53, 15);
	printf("여기에서 오목게임이 진행이 되어집니다.\n");

	getch();
	system("cls");

	printf("\n\n");

	game_table_tutorial(board);
```

```
move_point();                //화살표를 이동해서 상황판으로 간다
gotoxy(78, 14);
printf("이것은 상황판 입니다.\n");
gotoxy(78, 15);
printf("게임플레이어가 누구인지, 누구의 차례인지\n");
gotoxy(78, 16);
printf("제한시간이 얼마나 남았는지를 알려줍니다.\n");
getch();

system("cls");

game_table_tutorial(board);

printf("\n");
gotoxy(80, 14);
printf("그럼 이제 오목알을 놓아보겠습니다.");
gotoxy(80, 15);
printf("화살표를 이용하여 이동하고.");
gotoxy(80, 17);
printf("스페이스바를 이용하여");
gotoxy(80, 18);
printf("오목알을 놓을 지점을 정합니다.\n");

x_c = 36, y_c = 13;
// 스페이스를 입력받으면 다음 화면으로 넘어간다
gotoxy(x_c, y_c);
move_control_tuto(board);

//if(스페이스바를 눌렀을때 다음 명령문 출력)
system("cls");
game_table_tutorial(board);

gotoxy(80, 14);
printf("그럼 ");
gotoxy(80, 15);
printf("오목알을 가로 혹은 세로 혹은 대각선으로");
gotoxy(80, 16);
printf("다섯알을 놓아보겠습니다.\n");
```

```
	do
	{
		move_control_tuto(board);

		draw_omok();

		for (i = 0; i < MAX_LINE; i++)
		{
			for (j = 0; j < MAX_LINE; j++)
			{
				gotoxy(i * 2 + 22, j + 7);

				if (board_isSet(board, i, j))
					printf("●");

			}
		}
	} while (check_board_complete(board) != true);

	gotoxy(80, 18);
	printf("잘하셨습니다.\n");
	gotoxy(80, 19);
	printf("이렇게 다섯알을 놓으면");
	gotoxy(80, 20);
	printf("게임에서 승리하게 됩니다.\n");

	getch();
	system("cls");
	gotoxy(30, 8);
	printf("게임에 대한 이해가 조금 되셨나요?\n");
	gotoxy(30, 10);
	printf("수고하셨습니다^^\n");

	gotoxy(1, 20);

}

void game_table_com() {
	Turn turn = Black;
```

```
Node *node = NULL;

node = node_init(node);

srand(time(NULL));

x_c = 36, y_c = 13;

draw_rectangle(25, 35);
ready();

while (1) {
	print_board_com(node->boardBlack, node->boardWhite);

	gotoxy(80, 7);
	printf("player1 : %s (%s)\n", user.name, user.stone_shape2);
	gotoxy(80, 8);
	printf("player2 : %s (%s)\n", "COM", "○");
	gotoxy(80, 9);
	printf("============================\n");

	thread = CreateThread(NULL, 0, ThreadFunc, NULL, 0, NULL);

	if (turn == Black) {
		// user

		if (timeover == 1)
		{
			print_board_com(node->boardBlack, node->boardWhite);
			gotoxy(80, 13);
			printf("player1 이 승리하셨습니다");
			timeover = 0;
			break;
		}

		gotoxy(80, 10);
		printf("player1 의 차례입니다.          ");
		gotoxy(80, 11);
		printf("제한시간 : ");
```

```
		move_control(node->boardBlack, node->boardWhite);
		turn = White;

	}
	else {
		// com

		if (timeover == 1)
		{
			print_board_com(node->boardBlack, node->boardWhite);
			gotoxy(80, 13);
			printf("player2 이 승리하셨습니다.");
			timeover = 0;
			break;
		}
		gotoxy(80, 10);
		printf("COM 의 차례입니다.                ");
		gotoxy(80, 11);
		printf("제한시간 :");

		node = findNext(node);
		turn = Black;
		TerminateThread(thread, 0);
	}

	if (check_board_complete(node->boardBlack) == true)
	{
		print_board_com(node->boardBlack, node->boardWhite);
		gotoxy(80, 13);
		printf("player1 이 승리하셨습니다.");
		break;
	}

	if (check_board_complete(node->boardWhite) == true)
	{
		print_board_com(node->boardBlack, node->boardWhite);
		gotoxy(80, 13);
		printf("COM 이 승리하셨습니다.");
		break;
```

```
                }
        }
}

void game_table(MYSQL *conn)
{

        Board* board_1 = makeBoard((ll)0, (ll)0, (ll)0, (ll)0);
        Board* board_2 = makeBoard((ll)0, (ll)0, (ll)0, (ll)0);

        char buff1[255];
        char buff2[255];
        char buff3[255];

        int n, first_n;
        int i, j;

        srand(time(NULL));
        n = rand() % 2;
        first_n = n;

        x_c = 36, y_c = 13;

        draw_rectangle(25, 35);

        if (n % 2 == 1)
        {
                while (1)
                {
                        print_board(board_1, board_2, first_n);

                        gotoxy(80, 7);
                        printf("player1 : %s (%s)\n", user.name, user.stone_shape2);
                        gotoxy(80, 8);
                        printf("player2 : %s (%s)\n", user2.name, user2.stone_shape1);
                        gotoxy(80, 9);
                        printf("============================\n");

                        if (turn_two_players(n) == 1)
```

```
{
        thread = CreateThread(NULL, 0, ThreadFunc, NULL, 0, NULL);

        if (timeover == 1)
        {
                print_board(board_1, board_2, first_n);
                gotoxy(80, 13);
                printf("player1 이 승리하셨습니다\n");
                timeover = 0;
                user.win++;
                user2.lose++;
                sprintf(buff1, "update data set win = win + 1 where name = '%s'",
user.name);
                mysql_query(conn, buff1);
                sprintf(buff1, "update data set lose = lose + 1 where name='%s'",
user2.name);
                mysql_query(conn, buff1);
                break;
        }

        gotoxy(80, 10);
        printf("player1 의 차례입니다. ");
        gotoxy(80, 11);
        printf("제한시간 : ");
        move_control(board_1, board_2);
        n++;
}

else
{
        thread = CreateThread(NULL, 0, ThreadFunc, NULL, 0, NULL);

        if (timeover == 1)
        {
                print_board(board_1, board_2, first_n);
                gotoxy(80, 13);
                printf("player2 이 승리하셨습니다\n");
                timeover = 0;
                user2.win++;
```

```
                        user.lose++;
                        sprintf(buff1, "update data set lose = lose + 1 where name = '%s'",
user.name);
                        mysql_query(conn, buff1);
                        sprintf(buff1, "update data set win = win + 1 where name='%s'",
user2.name);
                        mysql_query(conn, buff1);
                        break;
                }
                gotoxy(80, 10);
                printf("player2 의 차례입니다. ");
                gotoxy(80, 11);
                printf("제한시간 :");
                move_control(board_2, board_1);
                n++;
        }

        if (check_board_complete(board_1) == true)
        {
                print_board(board_1, board_2, first_n);
                gotoxy(80, 13);
                printf("player1 이 승리하셨습니다\n");
                user.win++;
                user2.lose++;
                sprintf(buff1, "update data set win = win + 1 where name = '%s'", user.name);
                mysql_query(conn, buff1);
                sprintf(buff1, "update data set lose = lose + 1 where name='%s'", user2.name);
                mysql_query(conn, buff1);
                break;
        }

        if (check_board_complete(board_2) == true)
        {
                print_board(board_1, board_2, first_n);
                gotoxy(80, 13);
                printf("player2 이 승리하셨습니다\n");
                user2.win++;
                user.lose++;
                sprintf(buff1, "update data set lose = lose + 1 where name = '%s'", user.name);
```

```
                        mysql_query(conn, buff1);
                        sprintf(buff1, "update data set win = win + 1 where name='%s'", user2.name);
                        mysql_query(conn, buff1);
                        break;
                }

        }
        sprintf(buff1, "update data set rate = win*100/(win+lose) where name='%s'", user.name);
        mysql_query(conn, buff1);
        sprintf(buff1, "update data set rate = win*100/(win+lose) where name='%s'", user2.name);
        mysql_query(conn, buff1);
}
else
{
        while (1)
        {
                print_board(board_1, board_2, first_n);

                gotoxy(80, 7);
                printf("player1 : %s (%s)\n", user.name, user.stone_shape1);
                gotoxy(80, 8);
                printf("player2 : %s (%s)\n", user2.name, user2.stone_shape2);
                gotoxy(80, 9);
                printf("==============================\n");

                if (turn_two_players(n) == 1)
                {
                        thread = CreateThread(NULL, 0, ThreadFunc, NULL, 0, NULL);

                        if (timeover == 1)
                        {
                                print_board(board_1, board_2, first_n);
                                gotoxy(80, 13);
                                printf("player1 이 승리하셨습니다\n");
                                timeover = 0;
                                user.win++;
                                user2.lose++;
                                sprintf(buff1, "update data set win = win + 1 where name = '%s'",
user.name);
```

```
                        mysql_query(conn, buff1);
                        sprintf(buff1, "update data set lose = lose + 1 where name='%s'",
user2.name);
                        mysql_query(conn, buff1);
                        break;
                }
                gotoxy(80, 10);
                printf("player1 의 차례입니다. ");
                gotoxy(80, 11);
                printf("제한시간 : ");
                move_control(board_1, board_2);

                n++;

        }

        else
        {
                thread = CreateThread(NULL, 0, ThreadFunc, NULL, 0, NULL);

                if (timeover == 1)
                {
                        print_board(board_1, board_2, first_n);
                        gotoxy(80, 13);
                        printf("player2 이 승리하셨습니다\n");
                        timeover = 0;
                        user2.win++;
                        user.lose++;
                        sprintf(buff1, "update data set lose = lose + 1 where name = '%s'",
user.name);
                        mysql_query(conn, buff1);
                        sprintf(buff1, "update data set win = win + 1 where name='%s'",
user2.name);
                        mysql_query(conn, buff1);
                        break;
                }

                gotoxy(80, 10);
                printf("player2 의 차례입니다. ");
```

```
                gotoxy(80, 11);
                printf("제한시간 :");
                move_control(board_2, board_1);

                n++;
        }

        if (check_board_complete(board_1) == true)
        {
                print_board(board_1, board_2, first_n);
                gotoxy(80, 13);
                printf("player1 이 승리하셨습니다\n");
                user.win++;
                user2.lose++;
                sprintf(buff1, "update data set win = win + 1 where name = '%s'", user.name);
                mysql_query(conn, buff1);
                sprintf(buff1, "update data set lose = lose + 1 where name='%s'", user2.name);
                mysql_query(conn, buff1);
                break;
        }

        if (check_board_complete(board_2) == true)
        {
                print_board(board_1, board_2, first_n);
                gotoxy(80, 13);
                printf("player2 이 승리하셨습니다\n");
                user2.win++;
                user.lose++;
                sprintf(buff1, "update data set lose = lose + 1 where name = '%s'", user.name);
                mysql_query(conn, buff1);
                sprintf(buff1, "update data set win = win + 1 where name='%s'", user2.name);
                mysql_query(conn, buff1);
                break;
        }
}
sprintf(buff1, "update data set rate = win*100/(win+lose) where name='%s'", user.name);
mysql_query(conn, buff1);
sprintf(buff1, "update data set rate = win*100/(win+lose) where name='%s'", user2.name);
mysql_query(conn, buff1);
```

```
		}
}
void move_point()
{
		int i;
		for (i = 53; i < 110; i++)
		{
				gotoxy(i, 13);
				printf("  ↖");

				Sleep(20);
		}
}

void move_control(Board* board_A, Board* board_B)
{
		char key;
		int put = 0;

		do
		{
				key = getch();

				if (count_time < 1)
				{
						break;
				}

				switch (key)
				{
				case 32:
						//printf("%d %d\n", x_c, y_c);
						if (board_isSet(board_A, (x_c - 22) / 2, y_c - 7) || board_isSet(board_B, (x_c - 22) / 2, y_c
- 7))
								continue;
						*board_A = *(board_set(board_A, (x_c - 22) / 2, y_c - 7)); /*수를 놓았을때 표현*/
						put = 1;
						break;
```

```
                case 72:
                        y_c--;
                        if (y_c < 1)
                                y_c = 1;
                        break;
                case 75:
                        x_c -= 2;
                        if (x_c < 1)
                                x_c = 1;
                        break;
                case 77:
                        x_c += 2;
                        break;
                case 80:
                        y_c++;
                        break;
                }

                gotoxy(x_c, y_c);
        } while (put != 1);

        if (key == 32)
        {
                TerminateThread(thread, 0);
        }
}

void print_board_com(Board* boardBlack, Board* boardWhite) {
        int i, j;

        draw_omok();

        for (i = 0; i < MAX_LINE; i++)
        {
                for (j = 0; j < MAX_LINE; j++)
                {
                        gotoxy(i * 2 + 22, j + 7);

                        if (board_isSet(boardBlack, i, j)) {
```

```
                        printf("%s", user.stone_shape2);
                }

                if (board_isSet(boardWhite, i, j)) {
                        printf("○");
                }
        }
    }
}

void print_board(Board* board_1, Board* board_2, int n)
{
    draw_omok();

    int i, j;

    for (i = 0; i < MAX_LINE; i++)
    {
        for (j = 0; j < MAX_LINE; j++)
        {
            gotoxy(i * 2 + 22, j + 7);

            if (n % 2 == 1)
            {
                if (board_isSet(board_1, i, j))
                    printf("%s", user.stone_shape2); // user 오목알

                if (board_isSet(board_2, i, j))
                    printf("%s", user2.stone_shape1); // user2 오목알
            }

            else
            {
                if (board_isSet(board_1, i, j))
                    printf("%s", user.stone_shape1); // user 오목알

                if (board_isSet(board_2, i, j))
                    printf("%s", user2.stone_shape2); // user2 오목알
            }
```

```
                }
        }
}

void move_control_tuto(Board* board)
{
        char key;
        int put = 0;

        do
        {
                key = getch();

                switch (key)
                {
                case 32:
                        if (board_isSet(board, (x_c - 22) / 2, y_c - 7))
                                continue;
                        //*board = *(board_set(board, (x_c - 22) / 2, y_c - 7)); /*수를 놓았을때 표현*/
                        *board = *(board_set(board, (x_c - 22) / 2, y_c - 7));
                        put = 1;
                        break;
                case 72:
                        y_c--;
                        if (y_c < 1)
                                y_c = 1;
                        break;
                case 75:
                        x_c -= 2;
                        if (x_c < 1)
                                x_c = 1;
                        break;
                case 77:
                        x_c += 2;
                        break;
                case 80:
                        y_c++;
                        break;
                }
```

```
                gotoxy(x_c, y_c);
        } while (put != 1);
}

DWORD WINAPI ThreadFunc(void *data)
{
        count_time = 30;
        time_t startTime, endTime;

        startTime = time(NULL);
        while (1)
        {
                endTime = time(NULL);

                if (endTime - startTime >= 1) {
                        startTime = endTime;
                        count_time--;
                        gotoxy(92, 11);

                        printf("%d              ", count_time);
                        gotoxy(x_c, y_c);

                        if (count_time < 0) {
                                gotoxy(92, 11);
                                printf("시간초과!!!\n");

                                timeover++;

                                return 0;
                        }
                }
        }

        return 0;
}

void draw_omok()
{
```

```
int i, j;
unsigned char a = 0xa6;
unsigned char b[13];

for (i = 1; i < 12; i++)
        b[i] = 0xa0 + i;

gotoxy(22, 7);
printf("%c%c", a, b[3]);

for (i = 0; i < 13; i++)
{
        gotoxy(i * 2 + 24, 7);
        printf("%c%c", a, b[8]);

}
printf("%c%c", a, b[4]);
printf("\n");

///

for (i = 0; i < 13; i++)
{
        gotoxy(22, 8 + i);
        printf("%c%c", a, b[7]);

        for (j = 0; j < 13; j++)
        {
                gotoxy(j * 2 + 24, 8 + i);
                printf("%c%c", a, b[11]);
        }

        gotoxy(50, 8 + i);
        printf("%c%c", a, b[9]);
        printf("\n");
}

////
gotoxy(22, 21);
```

```
		printf("%c%c", a, b[6]);

		for (i = 0; i < 13; i++)
		{
			gotoxy(i * 2 + 24, 21);
			printf("%c%c", a, b[10]);
		}

		gotoxy(50, 21);
		printf("%c%c", a, b[5]);
		printf("\n");
}

void ready() {
		if (realRoot == NULL) {
			realRoot = node_init(realRoot);
			realRoot->boundWhite.max_x = 8;
			realRoot->boundWhite.min_x = 6;
			realRoot->boundWhite.max_y = 8;
			realRoot->boundWhite.min_y = 6;
			realRoot->boundBlack.max_x = 8;
			realRoot->boundBlack.min_x = 6;
			realRoot->boundBlack.max_y = 8;
			realRoot->boundBlack.min_y = 6;
		}

		MCTS(realRoot);

		tmpRoot = realRoot;
}

Node* findNext(Node* node) {
		int i;
		Node *child = NULL, *grandChild, *bestChild;
		double bestValue = 0.0f, tmpValue;


		child = findChildSameBoard(tmpRoot, node->boardBlack, node->boardWhite);
		if (child == NULL)
```

```
                toFind = node;

        MCTS(tmpRoot);

        child = findChildSameBoard(tmpRoot, node->boardBlack, node->boardWhite);


        for (i = 0; i < child->nodes->size; i++) {
                grandChild = child->nodes->array + i;
                tmpValue = (grandChild->N == 0) ? 0.0f : (double)grandChild->win / grandChild->N;

                if (tmpValue >= bestValue) {
                        bestChild = grandChild;
                        bestValue = tmpValue;
                }
        }

        tmpRoot = bestChild;
        return node_copy(tmpRoot, false);
}

Node* findChildSameBoard(Node* node, Board* black, Board* white) {
        int i;
        bool found = false;
        Node *child;

        for (i = 0; i < node->nodes->size; i++) {
                child = node->nodes->array + i;
                if ((board_isEqual(child->boardBlack, black) == true) && (board_isEqual(child->boardWhite, white)
== true)) {
                        found = true;
                        break;
                }
        }

        if (found == true)
                return child;
        else
                return NULL;
```

```
}

void MCTS(Node *root) {

	ll startTime, endTime;

	srand(time(NULL));
	startTime = time(NULL);
	while (1) {

		// time out check to break while loop.
		endTime = time(NULL);
		if ((endTime - startTime) >= MAX_SEARCH_TIME_IN_SEC) break;

		simulate(root);
	}
}

int simulate(Node *node) {
	Node *randomMove = NULL;
	int result;

	if (check_board_complete(node->boardBlack)) {
		return 0;
	}
	else if (check_board_complete(node->boardWhite)) {
		return 1;
	}

	randomMove = selectHeuristic(node);
	if (randomMove == (Node*)NULL) {
		return 0;
	}
	else {
		result = simulate(randomMove);
		result = ((result == 0) ? 1 : 0);
		node->win = node->win + result;
		node->N = node->N + 1;
		return result;
```

```
		}
}

Node* selectHeuristic(Node *root) {
		int x, y;
		int i;
		Node *selectedNode = NULL;
		Node *childNode = NULL;
		Turn childTurn = (root->turn == Black) ? White : Black;
		Board *selectedBoard = NULL;
		int min_x, min_y, max_x, max_y;
		//time_t startTime, endTime;

		// limit boundary of oposite player's bound.
		if (childTurn == White) {
			min_x = root->boundBlack.min_x;
			min_y = root->boundBlack.min_y;
			max_x = root->boundBlack.max_x;
			max_y = root->boundBlack.max_y;
		}
		else {
			min_x = root->boundWhite.min_x;
			min_y = root->boundWhite.min_y;
			max_x = root->boundWhite.max_x;
			max_y = root->boundWhite.max_y;
		}

		while (1) {
			if (toFind != NULL) {
				if (childTurn == Black)
					board_find_diff_one(toFind->boardBlack, root->boardBlack, &x, &y);
				else
					board_find_diff_one(toFind->boardWhite, root->boardWhite, &x, &y);
			}
			else {
				x = (rand() % (max_x - min_x + 1)) + min_x;
				y = (rand() % (max_y - min_y + 1)) + min_y;
			}
```

```
                if (board_isSet(root->boardBlack, y, x) || board_isSet(root->boardWhite, y, x))
                        continue;

                if (toFind != NULL) toFind = NULL;

                if (root->nodes != NULL) {
                        for (i = 0; i < root->nodes->size; i++) {
                                childNode = (root->nodes->array + i);
                                selectedBoard = (childTurn == Black) ? childNode->boardBlack : childNode-
>boardWhite;

                                // when there is matched child.
                                if (board_isSet(selectedBoard, y, x))
                                        return childNode;
                        }
                }

                // When there is no matchted child with proper x, y.
                selectedNode = node_copy(root, false);
                if (childTurn == Black) selectedNode->boardBlack = board_set(selectedNode->boardBlack, y, x);
                else selectedNode->boardWhite = board_set(selectedNode->boardWhite, y, x);
                setBound(selectedNode, x, y, childTurn);
                selectedNode->turn = childTurn;
                selectedNode->N = 0;
                selectedNode->win = 0;

                narray_insert(root->nodes, *selectedNode);
                return selectedNode;
        }
}

Node* node_init(Node* node) {
        node = (Node*)malloc(sizeof(Node));
        node->boardWhite = makeEmptyBoard();
        node->boardBlack = makeEmptyBoard();
        node->win = 0;
        node->N = 0;
        node->turn = White;
```

```
	node->nodes = narray_init();
	node->boundWhite.max_x = 0;
	node->boundWhite.max_y = 0;
	node->boundWhite.min_x = 0;
	node->boundWhite.min_y = 0;
	node->boundBlack.max_x = 0;
	node->boundBlack.max_y = 0;
	node->boundBlack.min_x = 0;
	node->boundBlack.min_y = 0;
	return node;
}

Node* node_copy(Node* target, bool liveChild) {
	Node *node = (Node*)malloc(sizeof(Node));
	node->boardWhite = board_copy(target->boardWhite);
	node->boardBlack = board_copy(target->boardBlack);
	node->win = target->win;
	node->N = target->N;
	node->turn = target->turn;
	if (liveChild == true)
		node->nodes = narray_copy(target->nodes);
	else
		node->nodes = narray_init();
	node->boundBlack.max_x = target->boundBlack.max_x;
	node->boundBlack.min_x = target->boundBlack.min_x;
	node->boundBlack.max_y = target->boundBlack.max_y;
	node->boundBlack.min_y = target->boundBlack.min_y;
	node->boundWhite.max_x = target->boundWhite.max_x;
	node->boundWhite.min_x = target->boundWhite.min_x;
	node->boundWhite.max_y = target->boundWhite.max_y;
	node->boundWhite.min_y = target->boundWhite.min_y;
	return node;
}

void setBound(Node* node, int x, int y, Turn turn) {
	if (turn == Black) {
		node->boundBlack.max_x = min(max(node->boundBlack.max_x, x + 1), 14);
		node->boundBlack.min_x = max(min(node->boundBlack.min_x, x - 1), 0);
		node->boundBlack.max_y = min(max(node->boundBlack.max_y, y + 1), 14);
```

```
                node->boundBlack.min_y = max(min(node->boundBlack.min_y, y - 1), 0);
        }
        else {
                node->boundWhite.max_x = min(max(node->boundWhite.max_x, x + 1), 14);
                node->boundWhite.min_x = max(min(node->boundWhite.min_x, x - 1), 0);
                node->boundWhite.max_y = min(max(node->boundWhite.max_y, y + 1), 14);
                node->boundWhite.min_y = max(min(node->boundWhite.min_y, y - 1), 0);
        }

}

bool check_board_complete(Board* board) {
        int i, j;
        bool isComplete = false;

        Board* boardHorizontal = makeBoard((ll)0, (ll)2181431069507584, (ll)0, (ll)0);   //가로 직선
        Board* boardReverseSlash = makeBoard((ll)0, (ll)281479271743488, (ll)65537, (ll)0);   //역슬래시
        Board* boardVertical = makeBoard((ll)0, (ll)9223653520421683200, (ll)16384, (ll)0);     //세로 직선
        Board* boardSlash = makeBoard((ll)0, (ll)4611967510585016320, (ll)4096, (ll)0);     //슬래시

        isComplete = check_board_mask(board, boardHorizontal);
        if (isComplete) return true;

        isComplete = check_board_mask(board, boardReverseSlash);
        if (isComplete) return true;

        isComplete = check_board_mask(board, boardVertical);
        if (isComplete) return true;

        isComplete = check_board_mask(board, boardSlash);
        if (isComplete) return true;

        return isComplete;
}

bool check_board_mask(Board* board, Board* mask) {
        int i;
        int max = MAX_LINE * MAX_LINE;
        int mid = max / 2;
```

```
		bool result;
		Board *checker;
		Board *target = board_copy(board);

		for (i = 0; i < max; i++) {
			if (i < mid) {
				checker = board_shift_left(mask, mid - i);
				result = board_isEqual(board_shift_right(board_and(target, checker), mid - i), mask);
			}
			else if (i == mid) {
				result = board_isEqual(board_and(target, mask), mask);
			}
			else {
				checker = board_shift_right(mask, i - mid);
				result = board_isEqual(board_shift_left(board_and(target, checker), i - mid), mask);

			}

			if (result == true) return true;
		}


		return false;
}

Board* makeBoard(ll A, ll B, ll C, ll D) {
		Board* board = (Board*)malloc(sizeof(Board));
		board->OM1 = A;
		board->OM2 = B;
		board->OM3 = C;
		board->OM4 = D;

		board_mask(board);

		return board;
}

/* It's Same As makeBoard(0, 0, 0, 0)*/
Board* makeEmptyBoard() {
```

```
	Board* board = (Board*)malloc(sizeof(Board));
	board->OM1 = 0ULL;
	board->OM2 = 0ULL;
	board->OM3 = 0ULL;
	board->OM4 = 0ULL;

	return board;
}

//operators
Board* board_mask(Board* board) {
	static const ll mask = (~0ULL) >> 31;

	board->OM4 = (board->OM4) & mask;

	return board;
}

Board* board_shift_left(Board* board, int shift) {
	int bigMove = shift / MAX_BIT_CNT;
	int smallMove = shift % MAX_BIT_CNT;
	ll mask = (ll)~0;
	ll tmp;
	Board* target = board_copy(board);

	while (bigMove-- > 0) {
		target->OM1 = target->OM2;
		target->OM2 = target->OM3;
		target->OM3 = target->OM4;
		target->OM4 = 0ULL;
	}

	if (smallMove > 0) {
		mask = mask >> (MAX_BIT_CNT - smallMove);
		target->OM1 = (target->OM1 >> smallMove) | ((mask & target->OM2) << (MAX_BIT_CNT -
smallMove));
		target->OM2 = (target->OM2 >> smallMove) | ((mask & target->OM3) << (MAX_BIT_CNT -
smallMove));
		target->OM3 = (target->OM3 >> smallMove) | ((mask & target->OM4) << (MAX_BIT_CNT -
```

```
smallMove));
                target->OM4 = (target->OM4 >> smallMove);
        }

        board_mask(target);

        return target;
}

Board* board_shift_right(Board* board, int shift) {
        int bigMove = shift / MAX_BIT_CNT;
        int smallMove = shift % MAX_BIT_CNT;
        ll mask = (ll)~0;
        ll tmp;
        Board* target = board_copy(board);

        while (bigMove-- > 0) {
                target->OM4 = target->OM3;
                target->OM3 = target->OM2;
                target->OM2 = target->OM1;
                target->OM1 = 0ULL;
        }

        if (smallMove > 0) {
                mask = mask << MAX_BIT_CNT - smallMove;
                target->OM4 = (target->OM4 << smallMove) | ((mask & target->OM3) >> (MAX_BIT_CNT -
smallMove));
                target->OM3 = (target->OM3 << smallMove) | ((mask & target->OM2) >> (MAX_BIT_CNT -
smallMove));
                target->OM2 = (target->OM2 << smallMove) | ((mask & target->OM1) >> (MAX_BIT_CNT -
smallMove));
                target->OM1 = (target->OM1 << smallMove);
        }

        board_mask(target);

        return target;
}
```

```
Board* board_and(Board* baseBoard, Board* layerBoard) {

	Board* target = board_copy(baseBoard);

	target->OM1 = target->OM1 & layerBoard->OM1;
	target->OM2 = target->OM2 & layerBoard->OM2;
	target->OM3 = target->OM3 & layerBoard->OM3;
	target->OM4 = target->OM4 & layerBoard->OM4;

	board_mask(target);
	return target;
}

Board* board_or(Board* baseBoard, Board* layerBoard) {

	Board* target = board_copy(baseBoard);

	target->OM1 = target->OM1 | layerBoard->OM1;
	target->OM2 = target->OM2 | layerBoard->OM2;
	target->OM3 = target->OM3 | layerBoard->OM3;
	target->OM4 = target->OM4 | layerBoard->OM4;

	board_mask(target);
	return target;
}

Board* board_xor(Board* baseBoard, Board* layerBoard) {

	Board* target = board_copy(baseBoard);

	target->OM1 = target->OM1 ^ layerBoard->OM1;
	target->OM2 = target->OM2 ^ layerBoard->OM2;
	target->OM3 = target->OM3 ^ layerBoard->OM3;
	target->OM4 = target->OM4 ^ layerBoard->OM4;

	board_mask(target);
	return target;
}
```

```
Board* board_not(Board* board) {
	Board* target = board_copy(board);

	target->OM1 = ~(target->OM1);
	target->OM2 = ~(target->OM2);
	target->OM3 = ~(target->OM3);
	target->OM4 = ~(target->OM4);

	board_mask(target);
	return target;
}

Board* board_copy(Board* board) {
	return makeBoard(board->OM1, board->OM2, board->OM3, board->OM4);
}

Board* board_set(Board* board, int i, int j) {
	ll mask = 1ULL;
	int which = 0;
	int shift;
	int toMove = i * MAX_LINE + j;
	Board* target = board_copy(board);

	if (0 <= j && j < MAX_LINE && 0 <= i && i < MAX_LINE) {
		which = toMove / MAX_BIT_CNT;
		shift = toMove % MAX_BIT_CNT;

		mask = mask << shift;

		switch (which) {
		case 0:
			target->OM1 = target->OM1 | mask;
			break;
		case 1:
			target->OM2 = target->OM2 | mask;
			break;
		case 2:
			target->OM3 = target->OM3 | mask;
			break;
```

```
                case 3:
                        if (shift <= 32)
                                target->OM4 = target->OM4 | mask;
                        else
                                printf("Error Occur In Board Set : Out Of Index\n");
                        break;
                default:
                        printf("Error Occur In Board Set : Out Of Index\n");
                        break;
                }
        }
        else {
                printf("Error Occur In Board Set : Out Of Index\n");
        }
        return target;
}

bool board_isSet(Board* board, int i, int j) {
        ll mask = 1ULL;
        ll target;
        int which = 0;
        int shift;
        int toMove = i * MAX_LINE + j;

        if (0 <= j && j < MAX_LINE && 0 <= i && i < MAX_LINE) {
                which = toMove / MAX_BIT_CNT;
                shift = toMove % MAX_BIT_CNT;

                mask = mask << shift;

                switch (which) {
                case 0:
                        target = board->OM1;
                        break;
                case 1:
                        target = board->OM2;
                        break;
                case 2:
                        target = board->OM3;
```

```
                        break;
                case 3:
                        target = board->OM4;
                        break;
                default:
                        printf("Error Occur In Board Set : Out Of Index\n");
                        break;
                }
        }
        else {
                printf("Error Occur In Board Set : Out Of Index\n");
        }


        return (target & mask) != 0ULL;
}

bool board_isEmpty(Board* board) {
        return (board->OM1 == 0ULL) && (board->OM2 == 0ULL) && (board->OM3 == 0ULL) && (board->OM4
== 0ULL);
}

bool board_isEqual(Board* boardA, Board* boardB) {
        return (boardA->OM1 == boardB->OM1) && (boardA->OM2 == boardB->OM2) && (boardA->OM3 ==
boardB->OM3) && (boardA->OM4 == boardB->OM4);
}

void board_find_diff_one(Board* bigger, Board* smaller, int *x, int *y) {
        int i, j;
        for (i = 0; i < MAX_LINE; i++) {
                for (j = 0; j < MAX_LINE; j++) {
                        if (board_isSet(bigger, i, j) == true && board_isSet(smaller, i, j) == false) {
                                *x = j;
                                *y = i;
                                return;
                        }
                }
        }
}
```

```
NodeArray* narray_init() {
	NodeArray *array = (NodeArray*)malloc(sizeof(NodeArray));
	array->array = (Node*)malloc(NODE_ARRAY_INIT_SIZE * sizeof(Node));
	array->size = 0;
	array->allocated = NODE_ARRAY_INIT_SIZE;
	return array;
}

void narray_insert(NodeArray *a, Node element) {
	if (a->size == a->allocated) {
		a->allocated *= 2;
		a->array = (Node*)realloc(a->array, a->allocated * sizeof(Node));
	}
	a->array[a->size++] = element;
}

NodeArray* narray_copy(NodeArray *target) {
	NodeArray* pointer = (NodeArray*)malloc(sizeof(NodeArray));
	int i;

	pointer->array = (Node*)malloc(target->allocated * sizeof(Node));
	pointer->size = target->size;
	pointer->allocated = target->allocated;

	for (i = 0; i < pointer->size; i++)
		(pointer->array)[i] = (target->array)[i];
	return pointer;
}

void narray_free(NodeArray *a) {
	int i;

	if (a->size > 0) {
		for (i = 0; i < a->size; i++)
			free(&(a->array)[i]);
	}

	free(a->array);
```

```
        a->array = NULL;
        a->allocated = a->size = 0;
}
```