오목 프로그램 구현

대구가톨릭대학교 컴퓨터소프트웨어학부

20117853 정윤지

1. 서론

2. 관련 연구

3. 프로그램 설계

4. 프로그램 구현

5. 프로그램의 동작과 결과

6. 결과 분석 및 토의

7. 결론

8. 참고문헌

9. 소스코드

1. 서 론

오목 프로그램을 구현하기 위해 C 프로그래밍 수업 시간에 배운 내용들을 최대한 응용하였고 그 밖의 필요한 기능들을 조사해서 참고하여 프로그램을 구현하였음. 새롭게 알게 된 내용과 프로젝트를 수행하는 과정에서 얻은 깨달음을 중심으로 결론 지었음. 관련연구, 프로그램 설계, 구현, 동작과 결과, 분석 및 토의, 결론, 참고문헌 순으로 구성함.

2. 관련 연구

STEAM R&E 연구결과보고 (오목 인공지능 게임 운용을 통한 출력시간과 승률 연구) - 이리고, 자바 GUI 를 활용한 오목 게임 구현 - 한국정보처리학회 2016 년도 춘계학술발표대회

3. 프로그램 설계

오목 프로그램을 구현하기 위해 먼저 내가 기존에 알고 있는 오목 규칙이 맞는지 확인하고 무엇부터 어떻게 만들어내야 하는지 순서를 정하며 계획을 세웠음. 가장 먼저 오목 게임을 실행하기 위해서 가장 기본적인 오목판을 만들어 대략적인 틀을 잡고, 오목알을 이동시킬 수 있도록 만들고, 오목 규칙을 적용한 알고리즘을 작성하고, 한 판의 게임이 종료된 후 다시 실행 또는 종료할 수 있게 만들며, 콘솔 화면에 제목을 달거나 색을 추가하여 꾸미고, 마지막으로 코드를 간결하게 다듬는 순으로 계획했다.

- 다음은 실제로 오목을 구현한 과정의 대략적인 순서이다.

1 번 -> 19X19 오목 판을 구성하는 기호들(┌, ┐, ┼, ╋ ...)을 2 차원 배열에 선언하여 저장한 후 출력한다.

2 번 -> kbhit()를 사용하여 키보드의 입력을 감지하며 getch()를 사용하여 입력을 받고, 방향키나 스페이스바의 아스키코드를 if 문과 swicth 문을 이용하여 조건을 설정한다.

3 번 -> 오목판의 기호가 저장된 2 차원 배열과 gotoxy 를 사용해 오목알이 움직이는 좌표와 동일한 위치로 설정하고 이동할 때 오목알을 출력하고 또 이동하면 그 전에 오목알이 출력했던 위치에는 오목판을 출력하도록 한다.

4 번 -> 스페이스바를 누르면 오목알이 고정되도록 하는데 해당 위치의 오목판 기호를 지우고 그 위치의 2 차원 배열에 오목알로 다시 초기화 시켜준다.

5 번 -> 하나의 오목알을 스페이스바로 고정시킬 때마다 색상을 바꾼다. (흑돌 차례, 백돌 차례)

6 번 -> 한번 고정된 오목알의 위치(2 차원 배열 위치)에 또 다시 오목알이 놓이는 일이 없도록, 스페이스를 누른 위치가 오목판이 아니라 오목알이라면 고정되지 않도록 설정한다.

7 번 -> 가로 세로 대각선에 연속적으로 같은 색상의 5 개 오목알이 있다면 승리하도록 하는 오목 규칙을 적용한 알고리즘을 설계하여 적용한다.

8 번 -> 한 판의 게임이 승부가 나면 또 다시 오목을 할 것인지 묻고 하지 않는다면 종료가 되도록 만든다.

-> 콘솔 시작 화면을 꾸미고 오목판을 가운데로 옮긴다. 즉 gotoxy 의 시작좌표가 (0,0)이 아닌 (30,12).

-> 하나의 알고리즘마다 함수를 만들어 간결한 코드로 다듬는다.

4. 프로그램 구현

< 1 번 > 19x19 오목판을 구성하는 기호 "┌, ┐, ┼, ╋"들은 2 바이트인 유니코드이기 때문에 1 바이트인 char 자료형으로 출력하면 ?와 같이 원치 않는 기호들이 나오는 오류가 발생한다. 그렇기 때문에 2 바이트인 wchar_t 자료형을 사용하면 원하는 기호가 나온다. wchar_t 를 사용하려면 printf 를 -> wprintf 로 사용해야 하고 (L"%c", WB)c 처럼 앞에 대문자 L 을 써야한다. 그리고 locale 헤더파일을 사용하여, setlocale(LC_ALL, "")처럼 사용하면 된다. 여기서 LC_ALL 이란 자신이 출력하고 싶은 기호들의 범위를 설정하는 것인데 편하게 ALL 을 사용했고 콤마 뒤에 큰따옴표사이를 비워두었는데 한글 같은 경우는 KOREAN 을 넣으면 된다.

< 2 번 > 오목알을 오목판 위에서 이동할 수 있도록 만드는데 필요한 kbhit 함수, getch 함수를 사용한다. kbhit 함수는 keybordhit 의 약자이며 사용자로부터 키 입력을 받았는지 안 받았는지 검사한다. 입력을 받았으면 C 언어에서 참을 의미하는 1 을 반환하고 입력을 받지 않았으면 거짓을 의미하는 0 을 반환한다. getch 함수는 conio 헤더파일을 넣어야 하고 버퍼를 사용하지 않으며, 문자를 입력 받는데 입력 문자들을 따로 출력하지 않으면 보이지 않는다. 여기서 오목알을 이동시키기 위해 kbhit 와 getch 를 사용하는 이유는 방향키를 계속 입력받아야 하는데 getch 로 입력받고 kbhit 로 키보도로부터 입력을 받았으면 실행시키도록 한다. 그런데 방향키는 아스키코드표에 존재하지 않는 2 바이트짜리 확장키의 하나이기 때문에 getch 를 두번 입력 받아서 두번째에 입력받은 값으로 방향키를 제어한다.

- > 그래서 while 문을 무한 반복하게 만들고 그 안에 if 문으로 kbhit 가 1 이라는 조건이 맞다면 변수 key 에 getch 로 입력받은 값을 저장한다. 그리고 중첩 if 문으로 그 저장값 key 가 2 바이트인 224 이거나 스페이스바의 아스키코드 값 32 일 때 또

if 문을 만일 입력받은 key 의 값이 스페이스이면 오목알을 고정시키게 하고 아니면 if 문이 성립하지 않아 실행을 하지 않는다. 그리고 또 key 의 값에 getch 를 넣어 아까 만일 방향키 중 하나를 입력 받았더라면 그 나머지 값인 72, 77, 75, 80 중에 하나가 저장이 되어 switch 문을 실행한다. Key 의 값이 up 이라면 위로 이동하지만 그 위치의 좌표가 0 이면 즉, 오목판의 가장자리이면 break 를 사용하여 위로 이동하지 못하게 하고 0 이 아닌 값이면 y 의 값에서 1 을 빼준다. 그리고 그 위치에 맞는 오목판이나 오목알을 출력한다. 나머지도 case 도 같은 방식이다. switch 문이 끝나면 오목의 승패를 판단해주는, 규칙을 쓴 함수를 호출하여 실행한다.

< 3 번 > 오목판에서 오목알을 이동시켰는데 오목알이 고정되어 이동을 했던 위치에 오목알이 출력이 계속된다. 그래서 이를 해결하기 위해 오목알을 지우고 그 위치에 맞는 오목판을 출력한다. 그러면 이동할 때마다 이동하기 전의 위치에 오목판이 출력되며 오목알을 지우고 이동한 위치에 오목알을 출력하고 이런식으로 반복하게 된다.

4 번 -> 스페이스바를 누르면 오목알이 고정되도록 한다.

< 4 번 > 오목알을 이동시키게 만든 후 스페이스바를 누르면 오목알이 고정되도록 만든다. 앞서 말한 무한 반복 while 문 안에 if 문의 4 중 if 문에 getch 로 입력받은 값이 스페이스이면 또 if 문에 흑돌이나 백돌에 그 위치의 좌표와 2 차원 배열 값에 해당 오목알을 그 배열값에 넣어준다. 그러면 커서를 이동할 때마다 출력되는 오목판 대신 고정된 오목알을 출력할 수 있게 된다.

5 번 -> 하나의 오목알을 고정시킬 때마다 색상을 바꾼다. (흑돌 차례, 백돌 차례)

< 5 번 > 2 인용 오목 게임이기 때문에 처음에 흑백돌의 차례일 때 하나의 위치를 선택한 뒤 흑백 돌의 차례가 되어서 원하는 위치에 선택하고 이런 식으로 하나의 오목알을 고정시키면 색상을 변경해야 한다. 그래서 if (key == SPACE)구문 안에 if, else if, else if 이렇게 3 개의 if 문이 있는데 첫번째 if 문은 흑돌 of 백돌이 이미 놓아져 있을 때 변경하지 못하도록 알림음으로 알려주며 2 번째 if 문은 백돌 차례가 되었을 때 그 위치에 고정된 백돌을 저장하고 WB 는 BLACK 인 블록 차례로 변경시켜준다. 3 번째는 흑돌 차례일 때 고정된 흑돌을 저장하고 차례를 백돌로 변경해준다.

6 번 -> 오목 규칙을 적용한 알고리즘을 설계한다.

< 6 번 > 오목 게임의 규칙은 누군가 먼저 가로, 세로, 대각선에 오목알을 연속해서 5 개를 두면 승리하는 게임이다. 그래서 하나의 오목판 안에 가로, 세로, 대각선에서 연속된 5 개의 오목알이 있는지 찾아내도록 만든다. 2 중 for 문을 2 개로 나누어 사용하는데 하나는, i 는 2 부터 16 까지 2 중 for 문의 j 는 0 부터 18 까지 돌리도록 해놓고 if 문을 사용한다. 그러면 if 문 조건을 보면 if ((board[j][i - 1] == WHITE) && (board[j][i - 2] == WHITE) && (board[j][i] == WHITE) && (board[j][i + 1] == WHITE) && (board[j][i + 2] == WHITE)) 인데 여기서 board[j][i]를 기준으로 5 개의 연속된 오목알을

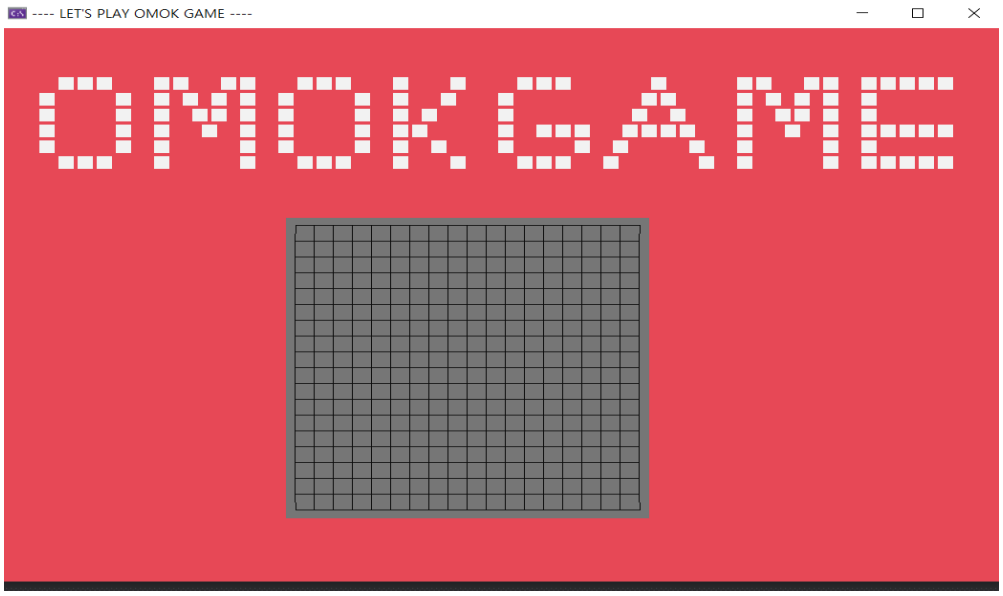
즉, j 라는 행은 고정되어 있고 i 라는 열은 -1, -2, +1, +2 로 해준다. 이것은 가로에 5 개의 연속된 백돌이 있는지 체크해주기에 이 if 문이 성립된다면 백돌이 승리하게 된다. 반대로 열인 i 에서 더하거나 빼주는 것이 아니라 행인 j 에서 빼주거나 더하면 세로측의 승리 체크 조건문이 된다. 그리고 if ((board[j + 2][i + 2] == WHITE) && (board[j + 1][i + 1] == WHITE) && (board[j][i] == WHITE) && (board[j - 1][i - 1] == WHITE) && (board[j - 2][i - 2] == WHITE)) 이 조건문은 오른쪽 하단에서 왼쪽 상단(x 는 감소, y 도 감소) + 왼쪽 상단에서 오른쪽 하단(x 는 증가, y 도 증가)으로 가는 대각선이다. 또 다른 대각선은 if ((board[j - 1][i + 1] == WHITE) && (board[j - 2][i + 2] == WHITE) && (board[j][i] == WHITE) && (board[j + 1][i - 1] == WHITE) && (board[j + 2][i - 2] == WHITE))인데 이 대각선은 왼쪽 하단에서 오른쪽 상단(x 는 증가 y 는 감소) + 오른쪽 상단에서 왼쪽 하단(x 는 감소, y 는 증가)으로 가는 대각선이다. 다음으론 한 판의 오목 게임이 종료된 후 다시 게임을 하고 싶은지 여부를 묻고 하고 싶다면 y 를 종료하고 싶다면 아무키나 누를시 있도록 만든다. Char 자료형에 ask 라는 변수를 선언하고 출력문을 실행한 뒤, ask 변수에 getch 를 저장해 if 문을 사용하여 ask 의 값이 'y'라면 화면을 초기화 시키고 다시 오목판을 출력시키고 게임을 다시 시작한다. 아무키나 눌렀다면 게임은 끝이 난다. 마지막으로 콘솔 타이틀을 LET'S PLAY OMOK GAME

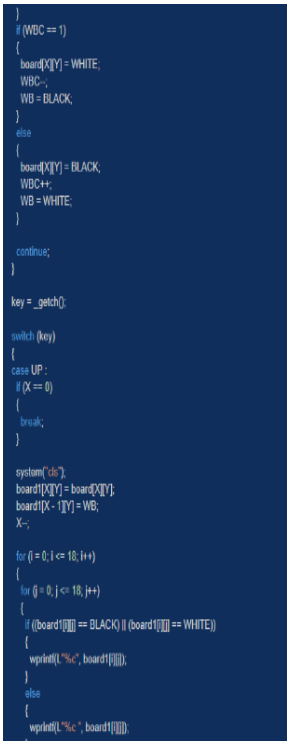
7 번 -> 알고리즘을 적용해 오류가 없는지 확인한다.

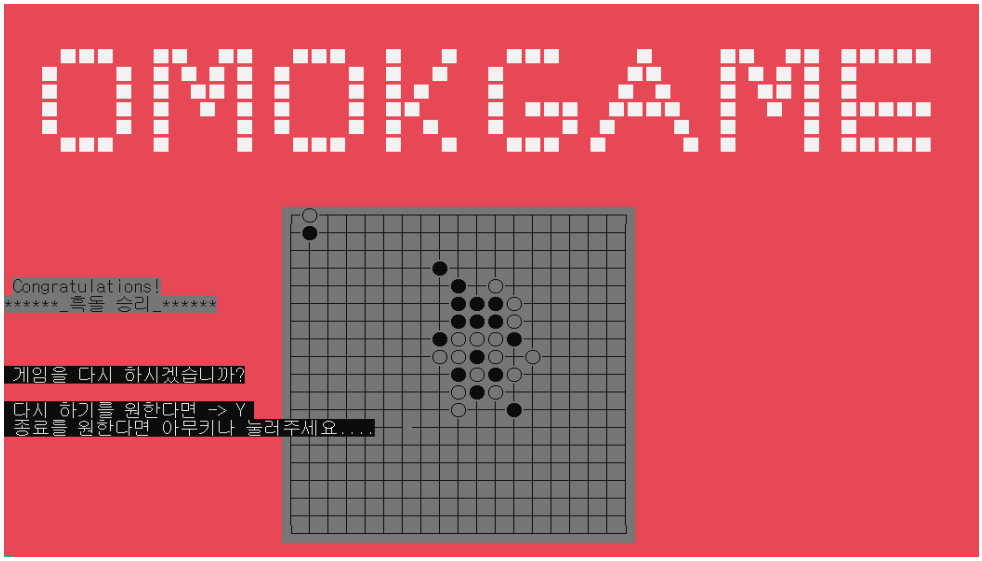
-> 오목 게임을 다시 시작하거나 종료할 수 있게 만든다.

-> 시작 화면을 꾸미고 그에 맞게 오목판을 위치를 이동시킨다

5. 프로그램의 동작과 결과







6. 결과 분석 및 토의

wchar_t 자료형을 사용하여 오목판이 정상적으로 출력 되었으며 오목알을 이동시킬 때 오목알의 흔적이 남지 않는다. 스페이스를 누르면 오목알이 고정이 되고, 오목알을 이미 두었던 위치에 스페이스를 눌러도 변하지 않는다. 오목을 이동시킬 때 깜빡임 현상을 제거 하였으며, 오목의 규칙을 적용해 가로 세로 대각선 오목의 결과가 성공적이다.

원래는 gotoxy 함수를 쓰지 않는 방법으로 코드를 작성했다. 위의 소스코드 사진과 같이 오목판을 만드는 2 차원 배열 2 개를 선언하여 한 개의 배열은 오로지 오목판과 고정된 오목알로만 구성되고, 또 하나의 배열은 오목알을 한 칸씩 이동시킬 때마다 그 위치의

배열에 오목알을 넣으며, 다시 지우며, for 문을 돌려 처음부터 끝까지 다시 출력하도록 반복하여 오목알을 이동시키는 방법으로 코드를 작성했다. 그런데 이동했던 오목알의 위치를 다시 오목판으로 바꾸기 위해서는 system("cls") 을 사용하여 화면을 초기화 시켜줘야만 했다. 그런데 그렇게 화면을 초기화 시키면 오목알을 이동시킬 때마다 깜빡임 현상이 발생한다. 그렇기 때문에 깜빡임 현상을 없애려고 결국 gotoxy 좌표 함수를 사용해서 앞서 말한 코드처럼 된다. 물론 gotoxy 를 사용하지 않고도, system("cls")를 사용하지 않고도 깜빡임 현상을 없애는 방법도 물론 있을것이라고 생각한다. 그치만 난 아직 찾지 못해 gotoxy 좌표 함수를 사용하였다.

7. 결론

C 프로그래밍 수업 시간에 배운 내용들을 응용해서 하나의 프로젝트를 만드는 작업은 처음에 지금 나의 수준에 현실적으로 가능한 일인가 하고 엄두가 나지 않을 정도로 어렵게 느껴졌었다. 난 난이도 상 중 하 중에서도 상인 오목을 하기로 결정했다. 높은 난이도를 하려고 시작하니 엄두가 나지 않은 건 사실이지만 난 어떤 일이든 결국 잘될 수 밖에 없다는 믿음이 뇌에 박혀 있는 그런 생각을 항상 가지고 있기 때문에 도전했다. 오목을 만들 때 오모판 기호를 2 차원 배열에 넣어 저장하고 출력하는데 원치 않는 기호들이 나와서 정말 당황했다. 그 기호들을 이용해 오목판으로 구현하는데만 꽤 많은 시간을 투자했었던 것 같다. 그래서 종목을 바꿀까 하던 찰나, 계속 검색을 하던 중 wchar_t 라는 자료형을 블로그에서 알게 되었다. 그 자료형을 사용하니 결국 2 차원 배열에 오목판 기호를 저장했던 것들이 잘 출력이 되었다. 정말 그 날 하루동안 가장 기뻤던 일이 오목판이 완성되었던 것, 바로 그 일이다. 아직 완성하진 않았지만 오목판을 구현한 것만으로도 성공한 것 같았다. 그리고 gotoxy 를 사용하기 전에 화면을 초기화 하는 방식으로 오목을 구현하였더니 이동시킬 때마다 깜빡임 현상이 발생했다. 그래서 난 깜빡임을 해결하기 위하여 gotoxy 를 사용해 다시 수정하여 만들었다. 그리고 오목 규칙을 만들고 적용해서 결국 오목 게임을 구현하였고 혹시나 콘솔 색을 바꿀 수 있나 하고 찾아보았는데 정말 바꿀 수가 있었다. 게다가 콘솔 제목도 바꿀 수가 있어서 블로그를 보며 참고했다. 결과물에 대해 엄청 만족한다. 그리고 프로그래밍이라는 것은 하나의 창조물을 만드는 것이라는 것은 알고 있었지만 그냥 눈으로 보기만 했지 실제로 하나의 프로그램을 구현해 보니까 창조, 창작한다는 것은 정말 매력적이라는 것을 한층 더 깨닫게 되는 계기가 되었다. 이번 프로젝트를 수행하는 과정에서 C 프로그래밍 수업 이외의 내용들을 찾아 응용하여 프로그램에 적용하였고 생각보다 C 언어는 광범위한 내용을 담고 있다는 것을 느꼈다. 나에게 C 언어에 대해 누군가가 질문을 하면 대답을 다 할 수 있을 정도의 관련 지식을 습득하기 위해서 나 스스로 관심 분야에 따로 공부를

해야 한다는 것을 알았고, 생각보다 많은 사람들이 열정과 열망을 가지고, 목표를 향해 많은 노력을 투자한다는 것을 알게 되어 자극과 동기를 많이 얻게 되었다. 지금 이 프로젝트를 통해 하나의 프로그램을 구현했다는 것도 너무 신기하고 대단한 거 같고, 내가 앞으로 밟아야 하는 많은 과정 중에서 첫 걸음을 디딘 것이라고 생각한다. 나의 이 마음가짐이 변치 않도록 주위의 경쟁자들을 보면서 자극도 얻고 동기도 얻는 긍정적 순환을 일으키도록 노력할 것이다. 이 프로젝트를 제작하는 과정에서 많은 것들을 얻게 되었고 깨닫게 되어서 매우 뿌듯하고 만족한다.

8. 참고 문헌

[1] Wchar_t : http://ehpub.co.kr/wchar_t-korean-c-language/

[2] Kbhit, getch : http://blog.naver.com/PostView.nhn?blogId=sharonichoya&logNo=220875372940

[3] 오목 룰 : https://cafe.naver.com/gubass/9858

[4] 커서 이동 함수 : https://blog.naver.com/sharonichoya/220873844942

[5] 방향키 이동 : https://medium.com/@dnjswbf/c%EC%96%B8%EC%96%B4-%EB%B0%A9%ED%96%A5%ED%82%A4%EB%A1%9C-%EC%9D%B4%EB%8F%99%EC%8B%9C%ED%82%A4%EA%B8%B0-46c6daf23719

9. 별첨 (소스코드와 주석)

```
#include <stdio.h> // 표준 입출력 함수를 사용하기 위한 라이브러리.
#include <conio.h> // getch 를 사용하기 위한 라이브러리.
#include <locale.h> // wchar_t 를 사용하기 위한 라이브러리.
#include <windows.h> // 화면 초기화를 사용하기 위한 라이브러리.
#define UP 72 // 방향키↑의 아스키 코드 값 72 를 UP 으로 정의.
#define DOWN 80 // 방향키 ↓의 아스키 코드 값 80 를 DOWN 으로 정의.
#define LEFT 75 // 방향키 ←의 아스키 코드 값 75 를 LEFT 로 정의.
#define RIGHT 77 // 방향키 →의 아스키 코드 값 77 를 RIGHT 로 정의.
#define SPACE 32 // 스페이스의 아스키 코드 값 32 를 SPACE 로 정의.
#define WHITE L'○'// 백돌의 색상을 WHITE 로 정의.
#define BLACK L'●' // 흑돌의 색상을 BLACK 로 정의.
#define WHITE_WIN printf("Congratulations!\n******_백돌 승리_******") // 백돌 승리 출력문을 WHITE_WIN 으로 정의.
#define BLACK_WIN printf("Congratulations!\n******_흑돌 승리_******") // 흑돌 승리 출력문을 BLACK_WIN 으로 정의.
void Screen(void); // 콘솔 화면 "OMOKGAME"출력 함수.
int Winner_rule(void); // 오목 승패 판단 함수.
void control(void); // 오목 게임 동작 함수.
void Draw_OmokBoard(void); // 오목판을 만들고 출력하는 함수.
void gotoxy(int x, int y); // gotoxy 좌표 함수.
void Erase_Stone(int x, int y); // 오목알 지우고 바둑판 출력하는  함수.(오목판에 놓은 오목알은 고정)
void Again_GAME(void); // 1 번의 실행이 끝난 후, 다시 실행 여부를 물어 실행 or 종료하는 함수.
wchar_t board[19][19] = { 0, }; // 오목 판 (19X19)을 만드는 2 차원 배열을 전역으로 선언.
int R = 0; // if 문 실행을 종료 판단하는 전역 변수 선언.
```

```
void textcolor(int color_number) // 콘솔 글자색 변경 함수.
{
	SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), color_number);
}
int main(void)
{
	system("color Cf"); // 콘솔 배경화면을 핑크색으로 설정
	system("mode con cols=105 lines=35 | title ---- LET'S PLAY OMOK GAME ----"); //
콘솔 창크기, 콘솔 제목 설정.
	setlocale(LC_ALL, ""); // wchar_t 자료형의 사용을 위해 setlocale 지정.
	Draw_OmokBoard(); // 오목판 출력 함수 호출.
	control(R); // 오목 게임 컨트롤 함수 호출.
	return 0;
}
void gotoxy(int x, int y) // 좌표이동을 위한 함수.
{
	COORD Pos;
	Pos.X = x;
	Pos.Y = y;
	SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Pos);
}
void Screen(void) // 시작화면에 OMOK GAME 글씨 출력하여 꾸미기.
{
	gotoxy(15, 2);
	printf("\n     ■■■    ■■  ■■   ■■■   ■   ■    ■■■       ■      ■■  ■■
■■■■■■\n");
	printf("   ■     ■ ■ ■ ■ ■    ■ ■  ■   ■          ■■     ■ ■ ■ ■ ■
\n");
	printf("   ■     ■ ■ ■■ ■ ■    ■ ■ ■    ■          ■ ■    ■ ■■ ■ ■\n");
	printf("   ■     ■ ■  ■ ■ ■    ■ ■■     ■ ■■■   ■■■■   ■  ■ ■
■■■■■■\n");
	printf("   ■     ■ ■     ■ ■    ■ ■ ■    ■    ■ ■    ■  ■     ■
■\n");
	printf("    ■■■   ■      ■   ■■■   ■   ■    ■■■  ■      ■ ■     ■
■■■■■■\n");
}
void Draw_OmokBoard(void) // 오목판을 만드는 함수.
{
	Screen(); // 시작 화면 글씨를 출력하는 Screen 호출.
	system("color Cf"); // 핑크 배경색.
	textcolor(128); // 바둑판은 보기 편하게 그레이 색으로 결정.
	int i, j; // for 문을 돌리기 위한 i, j 선언.
	board[0][0] = L'┌';
	board[0][18] = L'┐';
	board[18][0] = L'└';
	board[18][18] = L'┘';
	for (i = 1; i < 18; i++)
	{
		board[0][i] = L'┬';
		board[i][0] = L'├';
		board[i][18] = L'┤';
		board[18][i] = L'┴';
		for (j = 1; j < 18; j++)
		{
			board[i][j] = L'┼';
		}
	}
	for (i = 0; i <= 18; i++)
	{
		gotoxy(30, 12 + i); // 콘솔 시작화면인 가운데 좌표가 (0, 0) -> (30, 12)
		for (j = 0; j <= 18; j++)
		{
			if ((board[i][j] == BLACK) || (board[i][j] == WHITE))
```

```
                                        wprintf(L"%c", board[i][j]); // 띄어쓰기 필요없음.
                                else
                                        wprintf(L"%c ", board[i][j]); // 오목판 기호는
2 바이트를 차지하기 때문에 정사각형으로 출력하기 위해선 띄어쓰기가 필요함.
                        }
                        printf("\n");
                }
        }
void control(void) // 게임 컨트롤 하는 함수.
{
        int key; // key 는 방향키 아스키코드를 저장할 변수.
        int  x = 30, y = 12; // gotoxy(x, y)좌표를 (30, 12)부터 시작. (30, 12)가 (0, 0)인 셈.
        wchar_t WB = WHITE; // 시작은 백돌로 시작.

        while (1) // 누군가 승리하기 전까지 계속 무한 반복.
        {
                if (_kbhit()) // 키보드로부터 입력받으면 1 반환.
                {
                        key = _getch(); // getch 로 방향키 or 스페이스바 입력받기.
                        if (key == 224 || key == SPACE) // 입력 받은 값이
2byte 거나(224 부터 2byte) 스페이스바일 때
                        {
                                if (key == SPACE) // 입력받은 값이 스페이스이면 if 문
실행.
                                {
                                        if ((board[y - 12][(x - 30) / 2] == WHITE)
|| (board[y - 12][(x - 30) / 2] == BLACK)) // 고정된 오목알을 또다시 고정시키려고 하는 경우
                                        {
                                                printf("\a"); //알림음으로 알려준다.
                                                continue; // 고정된 오목알의 위치이니
아래를 실행시키지 않고 다음 위치로 변경한다.
                                        }
                                        else  if  (WB == WHITE) // 백돌 차례.
                                        {
                                                board[y - 12][(x - 30) / 2] = WHITE;
// 좌표위치와 같은 배열 위치에 백돌을 저장하여 고정시킨다.
                                                WB = BLACK; // 흑돌 차례로 변경한다.
                                        }
                                        else if (WB == BLACK) // 흑돌 차례
                                        {
                                                board[y - 12][(x - 30) / 2] = BLACK;
// 좌표위치와 같은 배열 위치에 흑돌을 저장하여 고정시킨다.
                                                WB = WHITE; // 백돌 차례로 변경한다.
                                        }
                                        continue; // 스페이스를 입력받았으니, 즉
오목알을 판에 놓아 이동할 필요가 없으니 continue 를 이용해 아래의 코드를 건너뛴다.
                                }
                                key = _getch(); // 2 바이트인 방향키의 남은 아스키코드
값을 다시 입력받는다.
                                switch (key) // key 의 switch 문을 실행시킨다.
                                {
                                case UP: // 방향키가 UP 이면 즉, 오목알을 위로 이동하면
다음을 실행.
                                        if (y == 12) // (30, 12)이 (0, 0)이니,
오목판의 가장 위쪽으로 이동한 좌표값 12 로 지정.
                                                break; // 오목판의 가장자리이니 더
이상 위로 못가게 break 해준다.
                                        Erase_Stone(x, y); // 오목을 둔 알은 고정하고
오목알 지우고 바둑판 출력하는 함수 호출.
```

```
                                        gotoxy(x, y - 1); // 다음 좌표는 위로 이동하니
y - 1
                                        wprintf(L"%c", WB); // 오목알 출력.
                                        y--; // y - 1 을 해준다.
                                        break;
                                case DOWN: // 방향키가 DOWN 이면 즉, 오목알을 아래로
이동하면 다음을 실행.
                                        if (y == 30) // 오목판의 가장 아래쪽으로 이동한
좌표값 30 으로 지정.
                                                break;
                                        Erase_Stone(x, y);
                                        gotoxy(x, y + 1); // 다음 좌표는 아래로
이동하니 y + 1
                                        wprintf(L"%c", WB);
                                        y++;

                                        break;
                                case LEFT: // 방향키가 LEFT 이면 즉, 오목알을 왼쪽으로
이동하면 다음을 실행.
                                        if (x == 30) // 오목판의 가장 왼쪽으로 이동한
좌표값 30 로 지정.
                                                break;
                                        Erase_Stone(x, y);
                                        gotoxy(x - 2, y);  // 다음 좌표는 왼쪽으로
이동하니 x - 2 -> - 2 인 이유는 가로가 2 바이트라서.
                                        wprintf(L"%c", WB);
                                        x -= 2;

                                        break;
                                case RIGHT: // 방향키가 RIGHT 이면 즉, 오목알을 오른쪽으로
이동하면 다음을 실행.
                                        if (x == 66) // 오목판의 가장 오른쪽으로 이동한
좌표값 66 로 지정.
                                                break;
                                        Erase_Stone(x, y); // 다음 좌표는 오른쪽으로
이동하니 x - 2 -> - 2 인 이유는 가로가 2 바이트라서.
                                        gotoxy(x + 2, y);
                                        wprintf(L"%c", WB);
                                        x += 2;
                                        break;
                                }
                        }
                }
                Winner_rule();  // 오목 승패 판단 함수 호출.
                if (R == 1)  // 승리가 판별되면 for 문 종료.
                        break;
        }
}
void Erase_Stone(int x, int y) // 고정된 오목알은 제외, 오목알 지우고 바둑판 출력하는  함수
{
        gotoxy(x, y); // 현재 오목알이 위치한 곳으로 이동,

        if (board[y - 12][(x - 30) / 2] == WHITE || board[y - 12][(x - 30) / 2] ==  BLACK)
// 좌표값을 뺀 값이 배열에선 좌표위치.
        {
                wprintf(L"%c", board[y - 12][(x - 30) / 2]); // 만일 현재 board 의 값이
오목알이면 띄어쓰기를 하지 않은 채로 출력.
        }
        else
        {
```

```
                    wprintf(L"%c ", board[y - 12][(x - 30) / 2]); // 만일 현재 board의 값이
오목알이면 띄어쓰기를 해준다.
          }
}
}
int Winner_rule(void) // 오목 승패 판단 함수.
{
          int i, j;
          for (i = 2; i < 17; i++) {
                    for (j = 0; j < 19; j++) {
                              if ((board[j][i - 1] == WHITE) && (board[j][i - 2] == WHITE) &&
(board[j][i] == WHITE) && (board[j][i + 1] == WHITE) && (board[j][i + 2] == WHITE)) // 가로
                              {
                                        gotoxy(1, 16); // 출력문을 보기 좋은 위치로 좌표 이동.
                                        WHITE_WIN; // 백돌 승리 출력문.
                                        R = 1;
                                        Again_GAME(); // 실행 여부 묻는 함수.
                                        break; // for문 종료.
                              }
                              else if ((board[i - 2][j] == WHITE) && (board[i - 1][j] ==
WHITE) && (board[i][j] == WHITE) && (board[i + 1][j] == WHITE) && (board[i + 2][j] ==
WHITE)) // 세로
                              {
                                        gotoxy(1, 16);
                                        WHITE_WIN;
                                        R = 1;
                                        Again_GAME();
                                        break;
                              }
                              else if ((board[j][i - 1] == BLACK) && (board[j][i - 2] ==
BLACK) && (board[j][i] == BLACK) && (board[j][i + 1] == BLACK) && (board[j][i + 2] ==
BLACK)) // 가로
                              {
                                        gotoxy(1, 16);
                                        BLACK_WIN; // 흑돌 승리 출력문.
                                        R = 1;
                                        Again_GAME();
                                        break;
                              }
                              else if ((board[i - 2][j] == BLACK) && (board[i - 1][j] ==
BLACK) && (board[i][j] == BLACK) && (board[i + 1][j] == BLACK) && (board[i + 2][j] ==
BLACK)) // 세로
                              {
                                        gotoxy(1, 16);
                                        BLACK_WIN;
                                        R = 1;
                                        Again_GAME();
                                        break;
                              }
                    }
                    if (R == 1) {
                              break;
                    }
          }
          for (i = 2; i < 17; i++)
          {
                    for (j = 2; j < 17; j++)
                    {
                              if ((board[j + 2][i + 2] == WHITE) && (board[j + 1][i + 1] ==
WHITE) && (board[j][i] == WHITE) && (board[j - 1][i - 1] == WHITE) && (board[j - 2][i - 2]
== WHITE)) // ↘ ↖ 대각선
                              {
                                        gotoxy(1, 16);
                                        WHITE_WIN;
```

```
                                        Again_GAME(); // 실행 여부 묻는 함수..
                                        R = 1;
                                        break;
                                }
                                else if ((board[j - 1][i + 1] == WHITE) && (board[j - 2][i + 2]
== WHITE) && (board[j][i] == WHITE) && (board[j + 1][i - 1] == WHITE) && (board[j + 2][i -
2] == WHITE)) // ↗ ↙ 대각선
                                {
                                        gotoxy(1, 16);
                                        WHITE_WIN;
                                        R = 1;
                                        Again_GAME();
                                        break;
                                }
                                else if ((board[j + 2][i + 2] == BLACK) && (board[j + 1][i + 1]
== BLACK) && (board[j][i] == BLACK) && (board[j - 1][i - 1] == BLACK) && (board[j - 2][i -
2] == BLACK)) // ↘ ↖ 대각선
                                {
                                        gotoxy(1, 16);
                                        BLACK_WIN;
                                        R = 1;
                                        Again_GAME();
                                        break;
                                }
                                else if ((board[j - 1][i + 1] == BLACK) && (board[j - 2][i + 2]
== BLACK) && (board[j][i] == BLACK) && (board[j + 1][i - 1] == BLACK) && (board[j + 2][i -
2] == BLACK)) // ↗ ↙ 대각선
                                {
                                        gotoxy(1, 16);
                                        BLACK_WIN;
                                        R = 1;
                                        Again_GAME();
                                        break;
                                }
                        }
                        if (R == 1) {
                                break; // 승리하였으면 for 문 종료...
                        }
                }
}
void Again_GAME(void) // 해당 판의 게임이 종료된 후, 다시 게임을 할 것인지 묻고 실행 or 종료하는
함수.
{
    char ask; // y 의 입력값을 위하여 ask 선언.
        gotoxy(2, 20); // 글을 출력하기 좋은 위치로 좌표 이동.
        textcolor(15); // 텍스트 컬러 설정.
        printf("\n 게임을 다시 하시겠습니까?\n");
        printf("\n 다시 하기를 원한다면 -> Y \n 종료를 원한다면 아무키나
눌러주세요....\n\n\n\n\n\n\n");
        ask = _getch(); // ask 변수에 입력받은 값 저장.
        if (ask == 'y')    { // ask 변수가 y 이면 화면 초기화 시키고 다시 오목 게임 시작.
                system("cls"); // 화면 초기화.
                Draw_OmokBoard(); // 오목판 출력.
                control(); // 다시 게임 컨트롤.
        }
}
```