

Design and Development of Gomoku

(설계 프로젝트 수행 결과 보고서)

© 2018 Suhyun Park
All Rights Reserved.

과목명: [CSE2003] 기초공학설계

담당교수: 서강대학교 컴퓨터공학과 김 주 호

개발자: 박수현 (201816xx)

개발기간: 2018. 5. 23 – 2018. 5. 30

각 단계별 결과 보고서

프로젝트 제목: Design and Development of Gomoku

제출일: 2018. . .

개발자: 박수현 (201816xx)

I. 개발 목표

`ncurses` 라이브러리를 이용해 UNIX Shell에서 동작하는 오목 게임을 C로 구현하면서, 기초공학 설계 과목에서 학습한 C에 관련된 내용을 완전히 체득하고, 함수를 이용해 프로그램을 모듈화 (modularization)하는 방법을 체득하며, 라이브러리에 대한 `documentation`을 찾아보고 이해한다.

II. 개발 범위 및 내용

가. 개발 범위

- 1.(기본 구현 사항) 2인 오목 게임
 - 1-1. 사용자에게서 오목판의 크기, 사용자 수를 입력 받아 오목판을 초기화하는 기능.
 - 1-2. 사용자에게서 키 입력을 받아 방향키일 경우 해당 방향으로 커서를 이동시키고 Enter 혹은 Space일 경우 해당 자리에 돌을 놓는 기능.
 - ◆ 1-2-1. (추가 구현 사항 b1) 이미 해당 위치에 돌이 놓여 있을 경우 돌을 새로 놓는 대신 경고 메시지를 출력하는 기능.
 - 1-3. 돌을 놓을 때마다 게임 종료 여부를 체크하여 승자가 있을 경우 승자를 출력하고 게임을 마치는 기능.
 - 1-4. (추가 구현 사항 b2) 오목판 위에 더 이상 돌을 놓을 수 없을 경우 ‘무승부’ 메시

지를 출력하고 게임을 마치는 기능.

- 1-5. (추가 구현 사항 b3) 오목판의 현재 상태를 색상과 함께 출력하는 기능.
- 1-6. (추가 구현 사항 a1) 게임 도중 게임을 저장하거나, 게임을 시작하기 전에 저장된 파일 이름을 입력해 게임을 재개하는 기능.

- 2. (추가 구현 사항 a2) 4인 3목 게임: 구현할 기능은 2인 오목 게임과 같다.

나. 개발 내용

게임 로직에 있어서는 2차원 배열과 `ncurses` 라이브러리를 적극적으로 사용할 필요가 있을 것이고, 저장과 재개 부분에서는 파일 입출력을 사용해야 할 것이라고 예상된다.

III. 추진 일정 및 개발 방법

가. 추진 일정

- 2018년 5월 23일: 개발 범위에 명시된 것 중 1-1. ~ 1-3., 2. 에 대한 구현.
- 2018년 5월 28일: 개발 범위에 명시된 것 중 1-4. ~ 1-5. 에 대한 구현.
- 2018년 5월 30일: 개발 범위에 명시된 것 중 1-6. 에 대한 구현.

나. 개발 방법

- 1.
 - 1-1. `scanf`를 이용하여 구현한다. 오목판은 주어진 코드(`initBoard`)를 사용하여 2차원 배열로 초기화한다.
 - 1-2. `curses` 모드에 들어간 후 `ncurses` 라이브러리가 제공하는 함수들을 이용해 키 입력, 커서 이동 등을 별도 함수에서 처리하도록 구현한다.
 - 1-3. 2차원 배열의 모든 원소에 대해 그 주변 원소들을 8개 방향 모두에 대해 체크

(bruteforcing)하여 연속으로 5개 (또는 3인 플레이일 경우 4개) 돌이 놓여 있는 경우 승자를 출력하고 게임을 종료하도록 한다.

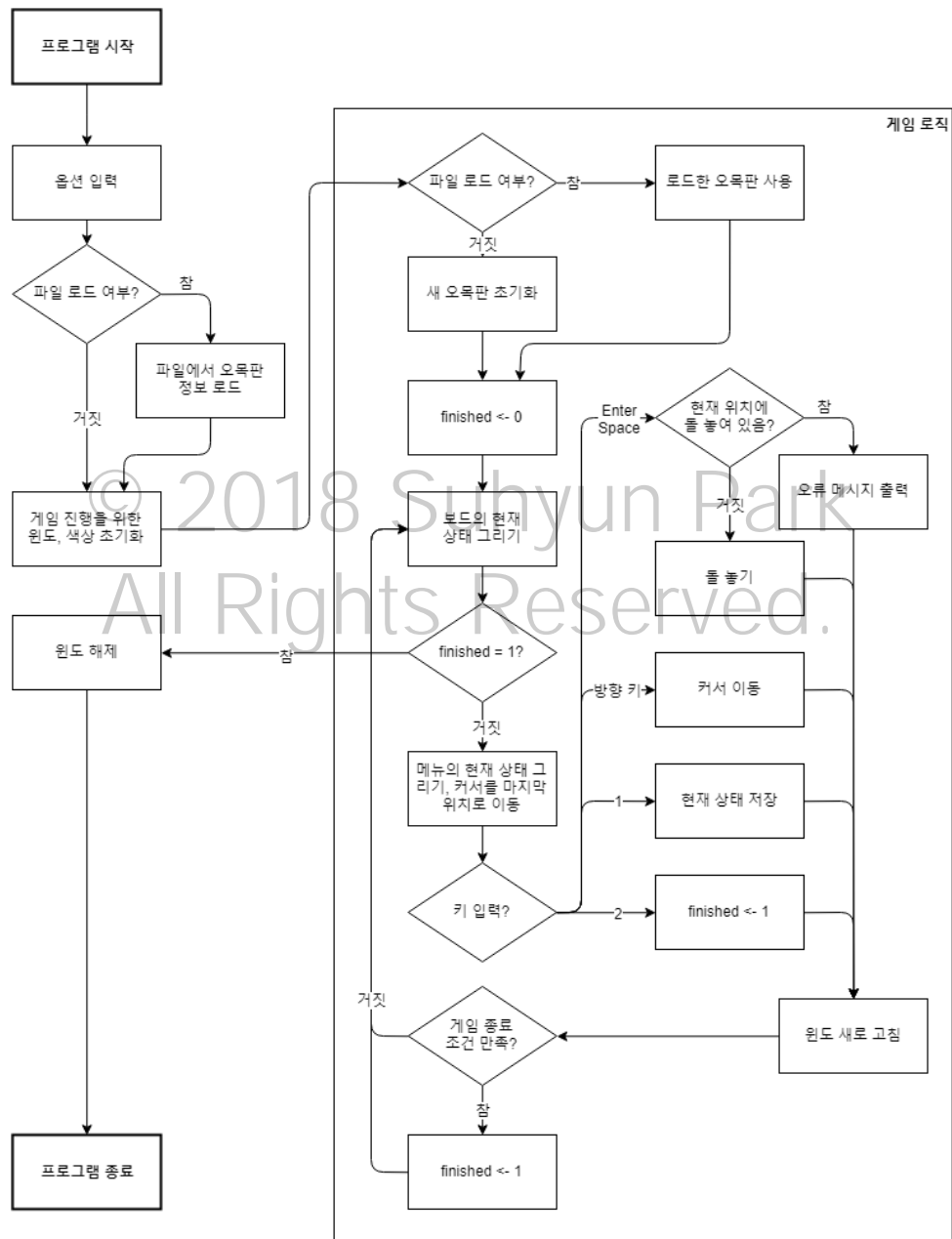
- 1-4. 2차원 배열의 모든 원소가 돌이 놓여 있는 경우 무승부를 출력하고 게임을 종료하도록 한다.
 - 1-5. `ncurses`에서 제공하는 함수들을 이용해 배경 색상과 전경 색상을 입힌다.
 - 1-6. `fscanf`와 `fprintf`를 이용해 현재 보드 상태를 파일로 저장하고 읽도록 한다. 또한 사용자에게서 파일명 입력을 받기 위해 `curses` 모드에서 자유롭게 나가고 다시 들어올 수 있게 한다.
- 2. 위의 기능들을 조건문을 사용해 플레이어가 2명일 때와 3명일 때 조금씩 다르게 동작하도록 구현한다.

© 2018 Suhyun Park
All Rights Reserved.

IV. 연구 결과

1. 합성 내용

소프트웨어 구성도



게임 종료 조건 만족은 오목판의 모든 칸에 대해 8개 방향으로 인접한 칸들을 확인해 연속으로

5개(3목의 경우 4개)가 같은 색상일 경우, 혹은 무승부일 경우 만족하도록 했다. 윈도, 색상 초기화, 보드 그리기, 메뉴 그리기, 커서 이동, 윈도 새로 고침 등은 모두 **ncurses** 라이브러리의 함수를 이용하여 구현하였다. 자세한 내용은 ‘분석 내용’에 후술한다.

함수 목록

함수명	기능 및 역할
main	옵션 입력과 게임 진행을 위한 윈도 및 색상 초기화 후, 게임을 시작하고, 게임 종료 후 콘솔 모드로 다시 돌아가게 하는 역할 수행
gameStart	보드 초기화 후 게임 로직을 관리하는 역할 수행
Action	키 입력을 처리하는 역할 수행
checkWin	현재 오목판 상태를 분석해 승자가 있을 경우 승자를, 무승부일 경우 무승부임을 확인하는 역할 수행
initBoard	오목판을 초기화하는 역할 수행
saveGame	현재 게임 상태를 저장하는 역할 수행
readSavedGame	저장된 게임 상태를 불러오는 역할 수행
paintBoard	현재 오목판 상태를 화면에 출력하는 역할 수행
paintMenu	현재 메뉴 상태를 화면에 출력하는 역할 수행
clearMenu	현재 메뉴 내용을 전부 지우는 역할 수행
paintErrorMessage	둘이 있는 위치에 둘을 놓으려 할 경우 오류 메시지 출력
clearErrorMessage	오류 메시지를 지우는 역할 수행
paintWinMessage	게임이 종료되었을 때 승자를 출력하는 역할 수행
paintDrawMessage	게임이 무승부로 종료되었을 때 무승부임을 출력하는 역할 수행
paintTerminationMessage	게임이 사용자에게 의해 종료되었을 때 안내 메시지를 출력하는 역할 수행

사용자 매뉴얼

1. 게임 실행

```
cse20181634@cspro9:~/omok$ ./a.out
Load saved game? [Y/N] : N
Enter the HEIGHT of the board [8 - 32] : 20
Enter the WIDTH of the board [8 - 64] : 20
Enter the number of players [2 - 3] : 2
```

게임을 실행하면 저장된 게임을 불러올 것인지 새로운 게임을 시작할 것인지 물어본다.

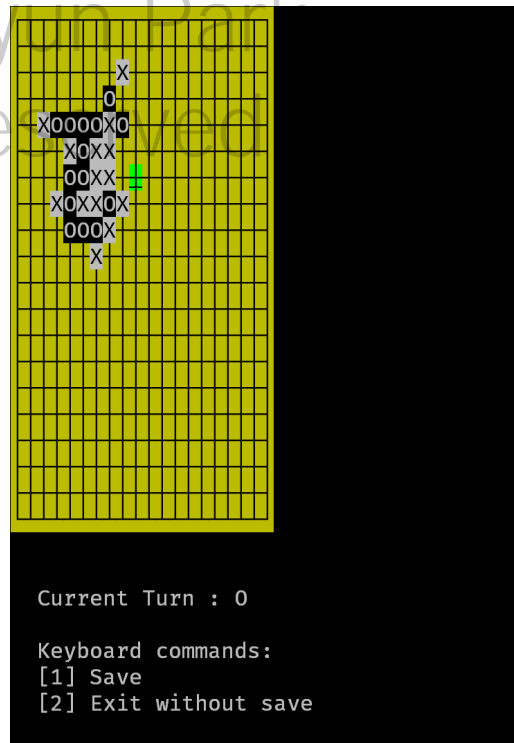
- **저장된 게임 재개:** Y 또는 y를 입력하면 저장된 게임의 파일명을 입력하여 게임을 재개할 수 있다. 파일 읽기에 실패했을 경우 파일명을 다시 입력할 수 있다.
- **새로운 게임 시작:** N 또는 n을 입력하면 오목판의 크기와 플레이어 수를 지정하여 새로운 게임을 시작할 수 있다.

2. 게임 진행

입력한 옵션으로, 혹은 재개된 파일로 오목판이 생성된다. 사용자는 키보드를 조작해 2 플레이어 모드인 경우 O → X 순으로, 3 플레이어 모드인 경우 O → X → Y의 순으로 돌을 놓을 수 있다.

2 플레이어 모드에서 O는 검은색 돌, X는 흰색 돌이며 3 플레이어 모드에서는 O는 빨간색 돌, X는 초록색 돌, Y는 파란색 돌이다.

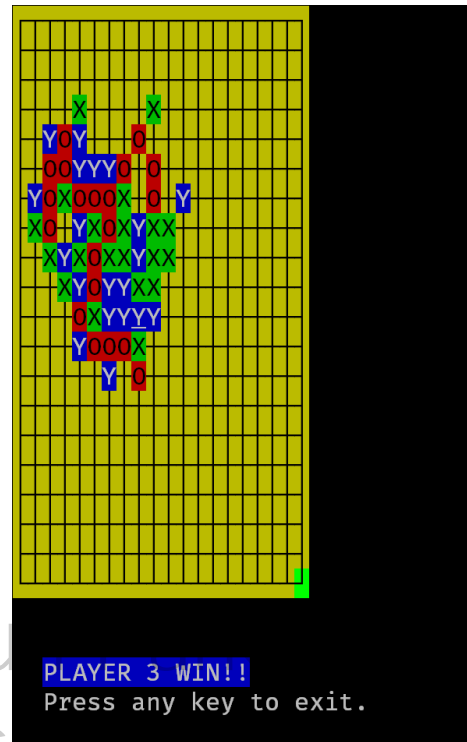
- **방향키**를 입력할 경우, 커서가 해당 방향으로 이동한다. 오목판의 가장자리에 커서가 있을 때엔 더 이상 이동하지 않을 수도 있다.
- **Enter** 혹은 **Space**를 입력할 경우, 현재 커서 위치에 다른 돌이 없을 때 현재 차례 플레이어가 돌을 현재 커서 위치에 놓을 수 있다.



- 1을 입력할 경우, 현재 게임을 저장할 수 있다. 파일명을 입력하면 해당 파일명으로 현재 게임이 저장된다.
- 2를 입력할 경우, 현재 게임이 종료된다.

3. 게임 종료

- 한 명이 이기거나 오목판의 모든 칸이 다 차서 더 이상 돌을 둘 수 없을 경우에는 게임이 종료된다.
- 누군가가 이겼을 경우에는 해당 플레이어가 이겼다는 메시지가 출력된다.
- 무승부일 경우에는 'DRAW'라는 메시지가 출력된다.
- 게임이 종료되었을 때 아무 키나 누르면 프로그램에서 나갈 수 있다.



2. 분석 내용

A. main() 함수

`main()` 함수는 프로그램의 시작점으로서, 게임의 옵션들을 읽어들이고 이를 바탕으로 게임을 실행하며, 게임이 종료된 후 다시 셸에 정상적으로 돌아갈 수 있게 하는 역할을 수행한다.

- 윈도우 선언

```
WINDOW *win, *menu;
```

오목판을 그리기 위한 윈도우 `win`과 메뉴를 그리기 위한 윈도우 `menu`를 따로 선언하였다. 따로 선언할 경우 이 윈도우들을 따로 관리할 수 있다는 장점이 있다.

- 변수 선언

```
char load;  
int players, row = 0, col = 0, turn = '0';  
int** loadedBoard;
```

게임에 필요한 변수들을 선언한다.

- `load`는 저장된 게임의 로드 여부로서, `gameStart` 함수에서 `load`이 Y 혹은 y일 경우에는 `readSavedGame`을 통해 가져온 오목판을 사용하게 된다.
- `players`, `row`, `col`, `turn`은 게임에 필수적인 변수들이다. 특히 저장된 파일을 로드할 경우 `readSavedGame`에 reference로 넘겨져, 파일에 저장된 플레이어 수, 현재 커서 위치, 현재 플레이어 순서를 가져오게 되고, 새 게임을 시작할 경우 초기에 선언되어 있는 값들로 게임을 시작하게 된다.
- `loadedBoard`는 `readSavedGame`을 통해 가져온 오목판 정보이다. 사용자가 파일을 로드하기로 선택했을 때에만 쓰인다.

- 옵션 입력

```
/*
    Prompts to ask options of the game
*/

// Option #1: load saved game
printf("Load saved game? [Y/N] : ");
scanf("%c", &load);

// Invalid option checking
while (load != 'Y' && load != 'N' && load != 'y' && load != 'n'){
    printf("Invalid option.\nAvailable options are 'Y', 'N' :");
    scanf("%c", &load)
}

if (load == 'Y' || load == 'y') {
    loadedBoard = readSavedGame(&players, &row, &col, &turn);
} else {
    // Option #2-1: height
    printf("Enter the HEIGHT of the board [8 - 32] : ");
    scanf("%d", &HEIGHT);

    while (HEIGHT < 8 || HEIGHT > 32) {
        printf("Invalid height.\nHeight should be in [8 - 32] : ");
        scanf("%d", &HEIGHT);
    }

    // Option #2-2: width
    printf("Enter the WIDTH of the board [8 - 64] : ");
    scanf("%d", &WIDTH);

    while (WIDTH < 8 || WIDTH > 64) {
        printf("Invalid width.\nWidth should be in [8 - 64] : ");
        scanf("%d", &WIDTH);
    }

    // Option #3: players
    printf("Enter the number of players [2 - 3] : ");
    scanf("%d", &players);

    // Invalid option checking
};
```

```

    while (players != 2 && players != 3) {
        printf("Invalid option.\nAvailable options are 2 - 3 :
");
        scanf("%d", &players);
    }
}

```

저장된 게임 로드 여부와, 로드하지 않는다면 8~32 사이의 높이, 8~64 사이의 너비, 플레이어 수를 입력 받는다. 잘못된 값을 입력 받았을 경우 제대로 된 값을 입력 받을 때까지 값을 다시 입력 받도록 while-loop을 수행한다.

- 오목판 크기에 제한을 건 이유는, 오목판 크기가 8보다 작을 경우 제대로 된 오목 게임이 이루어지기가 쉽지 않기 때문이며, 32×64 보다 클 경우 터미널이 전체 화면 일 경우에도 오목판이 터미널 크기를 넘어가서 넘어간 부분은 보이지 않게 되기 때문이다.

© 2018 Suhyun Park
 - 윈도 초기화

```

// Window initialization
initscr();
win = newwin(HEIGHT, WIDTH, 0, 0);
menu = newwin(7, 60, HEIGHT, 0);
noecho();
keypad(win, TRUE);
cbreak();

```

게임을 실행할 수 있도록 오목판 윈도 win과 메뉴 윈도 menu를 선언한다.

- initscr()는 ncurses 라이브러리의 기능을 사용할 수 있도록 해 주는 함수이다.
- noecho()는 키보드 입력이 화면에 표시되지 않도록 해 주는 함수이다.
- keypad(win, TRUE)는 win 윈도가 특수 키 입력을 받을 수 있도록 해 주는 함수이다. 방향키, Enter, Space 키를 입력받을 필요가 있으므로 이 함수를 호출해야 한다.

- `cbreak()`은 Enter 입력이 없어도 입력한 키를 바로 프로그램에 전달해 주는 역할을 한다.
 - 윈도우 `win`은 (0, 0)과 (HEIGHT, WIDTH)를 꼭지점으로 하는 윈도우이다.
 - 윈도우 `menu`는 (HEIGHT, 0)과 (HEIGHT + 7, 60)을 꼭지점으로 하는 윈도우로서, `win` 아래에 그려진다.
- 색상 초기화

```
// Color initialization
start_color();




init_pair(1, COLOR_WHITE, COLOR_BLACK); // [1] default = white on
black
init_pair(2, COLOR_BLACK, COLOR_YELLOW); // [2] board = black on
yellow

if (players == 2) {
    init_pair(3, COLOR_WHITE, COLOR_BLACK); // [3] O = white on
    black
    init_pair(4, COLOR_BLACK, COLOR_WHITE); // [4] X = black on
    white
} else if (players == 3) {
    init_pair(3, COLOR_BLACK, COLOR_RED); // [3] O = black on red
    init_pair(4, COLOR_BLACK, COLOR_GREEN); // [4] X = black on
    green
    init_pair(5, COLOR_WHITE, COLOR_BLUE); // [5] Y = white on
    blue
}

init_pair(6, COLOR_WHITE, COLOR_RED); // [6] error = red on white
```

`ncurses`에는 색상을 바꿀 수 있는 기능이 있다. `start_color`로 색상 관련 기능들을 사용할 수 있으며, `init_pair`로 색상 조합을 정의할 수 있다. 오목 게임에서 활용하기 위해 색상을 플레이어 수에 따라 아래와 같이 정의하였다.

	Pair 1	Pair 2	Pair 3	Pair 4	Pair 5	Pair 6

2인	ABGAbg		0	X		ABGAbg
3인	ABGAbg		0	X	Y	ABGAbg

- 게임 시작

```
gameStart(win, menu, load, loadedBoard, players, row, col, turn);
```

입력받은 정보로 게임을 시작한다.

- 게임 종료

```
// Game ended
endwin();
return 0;
```

게임 로직이 끝나면 curses 모드를 나가고 프로그램을 종료한다.

B. gameStart() 함수

gameStart() 함수는 while-loop을 이용해 게임 로직을 관리하는 함수이다. 게임이 끝나면 loop를 탈출해 main() 함수로 돌아간다. 이 함수는 아래와 같은 인자를 받는다.

- WINDOW *win, WINDOW *menu: main() 함수에서 초기화된 오목판 윈도우와 메뉴 윈도우이다.
- int load: 오목판을 파일에서 로드할지 여부이다. 로드할 경우 Y 혹은 y, 아닐 경우 N 혹은 n이다.
- int** loadedBoard: main() 함수에서 미리 로드한 오목판이다.
- int players, int row, int col, int turn: 게임과 관련된 정보들이다.

- 보드 초기화

```
int **board;
int finished = 0;
int keyin, result;
wmove(win, row, col);

// Initializing the board
if (load == 'Y' || load == 'y') {
    board = loadedBoard;
} else {
    board = initBoard(board);
}
```

오목판 정보를 파일에서 로드할 경우 `main()` 함수에서 로드된 오목판 정보를 사용하고, 아닐 경우 새로 오목판을 생성한다.

- 게임 루프

```
while (1) {
    /*
     * This While loop constantly loops in order to
     * draw every frame of the WINDOW.
     */

    // Paint board
    paintBoard(board, win, row, col);

    // Terminate game if finished
    if (finished) {
        keyin = wgetch(win);
        break;
    }

    // Paint menu
    paintMenu(menu, turn);

    // Move cursor to last action point
    wmove(win, row, col);

    // Get keyboard input
```

```
keyin = wgetch(win);

// Do action and save its result according to the keypress
result = Action(win, menu, board, keyin, &row, &col, &turn,
players);

// Refresh windows
wrefresh(win);
wrefresh(menu);

if (result == 1) {
    // Game is over
    finished = 1;
}
}
return;
```

while-loop 안은 게임 루프이다.

일단 `paintBoard()` 함수로 현재 오목판의 상태를 화면에 업데이트하고, 게임이 끝난 상태일 경우 루프를 탈출한다. 아닐 경우 `paintMenu()` 함수로 메뉴를 업데이트하고, `wmove()` 함수로 현재 위치로 커서를 이동시키며, `wgetch()`로 사용자에게서 키 입력을 받는다. 이 때 `wgetch()`는 입력이 있을 때까지 사용자를 기다리게 된다. 받은 키 입력을 `Action()` 함수로 처리하며, `wrefresh()` 함수로 윈도우를 업데이트한다.

`Action()` 함수에서 받아온 값이 1일 경우 게임을 종료 상태로 바꾼다. 게임이 종료되었다고 바로 프로그램이 종료된다면 사용자가 최종 보드 상태를 볼 수 없으므로, 아무 키 입력이나 받기 위해 루프를 한 번 더 실행하고 키 입력을 받은 경우 게임을 종료한다.

C. Action() 함수

`Action()` 함수는 사용자의 키 입력을 받아 게임이 종료되었는지 여부를 반환한다. 에러 메시지, 무승부 혹은 승리 메시지를 출력하는 일도 수행한다. 이 함수는 아래와 같은 인자를 받는다.

- WINDOW *win, WINDOW *menu: 오목판 윈도우와 메뉴 윈도우이다.

- `int keyin`: 유저에게서 입력받은 키 값이다.
- `int **board, int *row, int *col, int *turn, int players`: 현재 게임 정보이다.

```
/*  
    Following right after the keyboard input,  
    performs a corresponding action.  
  
    returns 0 if game is not over,  
    1 if game is over.  
*/  
  
int check, error = 0;  
  
switch (keyin) {  
    ...  
}
```

`switch`를 이용해 사용자의 키 입력에 따라 알맞은 동작을 수행하게 된다.

- 방향 키 입력

```
case KEY_DOWN:  
    if (*row < HEIGHT - 1) (*row)++;  
    break;  
case KEY_UP:  
    if (*row > 0) (*row)--;  
    break;  
case KEY_RIGHT:  
    if (*col < WIDTH - 1) (*col)++;  
    break;  
case KEY_LEFT:  
    if (*col > 0) (*col)--;  
    break;
```

입력받은 방향키에 따라 현재 커서 위치를 업데이트한다. 커서 위치가 오목판의 가장자리일 경우에는 업데이트하지 않는다.

- Enter / Space 키 입력

```

case KEY_Enter:
case KEY_SPACE:
    // O - X - Y - O - X - Y - ...
    if (board[*row][*col] == 'O' ||
        board[*row][*col] == 'X' ||
        board[*row][*col] == 'Y') {
        // Stone already exists
        error = 1;
    } else {
        board[*row][*col] = *turn;

        if (players == 2) {
            switch (*turn) {
                case 'O': *turn = 'X'; break;
                case 'X': *turn = 'O'; break;
            }
        } else if (players == 3) {
            switch (*turn) {
                case 'O': *turn = 'X'; break;
                case 'X': *turn = 'Y'; break;
                case 'Y': *turn = 'O'; break;
            }
        }
    }
    break;
  
```

현재 위치에 이미 돌이 놓여 있을 경우 에러 상태를 선언하고 아무 동작도 하지 않는다. 돌이 없을 경우 현재 플레이어의 돌을 놓으며, 다음 차례로 넘어간다. 2인 플레이일 경우 $O \rightarrow X \rightarrow O \rightarrow X \rightarrow \dots$ 의 순서로, 3인 플레이일 경우 $O \rightarrow X \rightarrow Y \rightarrow O \rightarrow X \rightarrow Y \rightarrow \dots$ 의 순서로 순서가 진행된다.

- 숫자 키 입력 (1~2)

```

case '1':
  
```

```

    keypad(win, FALSE);
    saveGame(board, players, *row, *col, *turn);
    keypad(win, TRUE);
    break;
case '2':
    paintTerminationMessage(menu);
    return 1;
    break;

```

숫자 1 입력의 경우, 현재 상태를 저장할 수 있도록 `saveGame()`을 호출한다. 이 때 과일명을 입력받을 때 방향키가 입력될 수 있으므로 방향키 입력은 꺼 둔다. 게임을 저장하고도 계속 게임을 할 수 있도록 게임 종료 상태를 리턴하지는 않는다(0).

숫자 2 입력의 경우, 사용자가 게임을 종료했음을 알리는 메시지를 출력하고 게임 종료 상태(1)를 리턴한다.

- 루프 이후 처리

```

wmove(win, *row, *col);

if (error) {
    paintErrorMessage(menu, "Stone already exists there!");
} else {
    clearErrorMessage(menu);
}

check = checkWin(board, players);

if (check != 0) {
    if (check == -1) { // DRAW
        paintDrawMessage(menu);
        return 1;
    } else { // WIN
        paintWinMessage(menu, check);
        return 1;
    }
}

```

```
wrefresh(win);
wrefresh(menu);

return 0;
```

커서를 현재 위치로 이동하고, 돌 중복 오류가 있을 경우에는 에러 메시지를 출력한다. `checkWin()`을 통해 게임이 끝났는지 여부를 체크하고, 끝났을 경우 무승부로 끝났느냐 승리로 끝났느냐에 따라 적절한 메시지를 출력하고 게임 종료 상태(1)을 리턴한다. 그렇지 않을 경우 게임 진행 상태(0)를 리턴한다.

D. `checkWin()` 함수

이 함수는 현재 오목판 상태를 체크해 게임이 누군가의 승리로 끝났을 경우 그 플레이어, 혹은 무승부일 경우 -1을 반환한다. 게임이 끝나지 않았을 경우 0을 반환한다. 모든 칸과 모든 방향을 체크하는 브루트포싱(bruteforcing) 기법을 이용한다.

예를 들어 현재 확인하려는 칸이 (x, y)라면, 지금 칸을 시작으로 5개(3인 플레이어의 경우 4개)의 돌이 연속적으로 같은지를 8개 방향에 대해 모두 확인해서 그런 경우가 하나 이상 있을 경우 확인을 종료하고 해당 돌을 반환한다.

또한 돌이 놓여져 있는 칸의 개수를 확인해, 오목판 면적과 돌의 개수가 같을 경우 모든 칸에 돌이 놓여있는 것으로 판단, 무승부 상태(-1)를 반환한다.

이 함수는 다음과 같은 인자를 받는다.

■ `int **board, int players`: 현재 게임의 상태.

- 선언

```
/*
    Check if the game is over.
    Returns the winner if the game is over; -1 if draw; else
    returns 0.
*/

int len = 5, stoneCount = 0;
int x, y, d, i, j, k, flag, value;
```

```
// Directions to check:
int dxarr[8] = {1, 1, 1, 0, 0, -1, -1, -1};
int dyarr[8] = {1, 0, -1, 1, -1, 1, 0, -1};
int dx, dy;

// 4-mok when 3 players
if (players == 3) len = 4;
```

오목판 체크에 필요한 변수들을 선언한다. 특히 **dxarr**와 **dyarr**는 연속적으로 돌이 놓여있는지를 확인하기 위해 현재 위치에 하나씩 더해 확인할 상수들이고, **len**개 연속적으로 놓여 있으면 확인을 종료하도록 했다. **dx**과 **dy**는 현재 확인하고 있는 방향이다.

- 승리 확인 알고리즘

```
// Check for all cells in board
for (x = 0; x < HEIGHT; x++) for (y = 0; y < WIDTH; y++) {
    value = board[x][y];

    if (
        (players == 2 && value != 'X' && value != 'O') ||
        (players == 3 && value != 'X' && value != 'O' && value !=
        'Y')
    ) continue; // Stone not placed here
    stoneCount++;
    for (d = 0; d < 8; d++) {
        dx = dxarr[d];
        dy = dyarr[d];
        flag = 1;

        i = x;
        j = y;

        for (k = 0; k < len; k++) {
            i += dx;
            j += dy;
```

```

        // If the checking coordinates goes out of bounds,
        check next case.
        if (i < 0) break;
        if (i > HEIGHT - 1) break;
        if (j < 0) break;
        if (j > WIDTH - 1) break;

        if (board[i][j] == value) {
            flag++;
            if (flag == len) return value;
        } else {
            break;
        }
    }
}
}

```

모든 칸에 대해서 확인한다. 만약 현재 확인하는 칸이 돌이 놓여 있지 않은 칸이라면 board에는 상자 그리기 문자들이 들어 있을 것이므로, 이는 고려하지 않고 확인한다.

시작점을 (x, y) 라고 하면 $k \in [0, \text{len} - 1]$ 에 대해 $(x + (k + 1) \times dx, y + (k + 1) \times dy)$ 에 돌이 놓여 있으며 그 값이 (x, y) 와 같은지 확인한다.

flag를 연속적으로 놓여 있는 돌의 개수라고 하면, 처음에는 flag를 1로 초기화시켜 두고 k가 0부터 len - 1일 때까지 순서대로 확인하며 같을 경우 flag를 1 증가시키고 시작점과 현재 위치의 값이 다르거나 좌표 $(x + (k + 1) \times dx, y + (k + 1) \times dy)$ 가 오목판의 범위를 벗어날 때에는 break 하여 다른 방향 혹은 다음 칸에 대해 확인한다.

이 때 flag = len일 경우에는 연속적으로 len개만큼 놓여 있는 것이므로 바로 승자를 반환한다. 모든 칸을 확인했는데 승자가 반환되지 않았을 경우에는 무승부 확인을 하게 된다.

- 무승부 확인 알고리즘

```

if (x * y == stoneCount) {
    return -1; // DRAW
} else {

```

```
        return 0;  
    }
```

오목판의 넓이가 놓여 있는 돌의 개수와 같을 때에는 무승부 상태(-1)를, 아닐 경우 게임 진행 상태(0)을 반환한다.

E. `initBoard()` 함수

이 함수는 전역 변수 `HEIGHT`와 `WIDTH`를 이용해 새로운 오목판을 초기화하는 역할을 한다.

F. `saveGame()` 함수

이 함수는 현재 게임 진행 상태를 저장하는 역할을 한다. 아래와 같은 인자를 받는다.

■ `int **board, int players, int row, int col, int turn`: 현재 게임 상태.

- 파일명 입력

```
/*  
    Save current game with given name  
*/  
  
FILE* data;  
char filename[100];  
int i, x, y;  
  
endwin();  
  
printf("Filename to save: ");  
scanf("%s", &filename);  
  
data = fopen(filename, "w");  
if (data == NULL) {  
    printf("Failed to write file.\n");  
    refresh();  
    return;  
}
```

```
}
```

사용자에게서 파일명을 입력받는다. 파일명을 입력받기 위해 `endwin()` 함수를 통해 `curses` 모드에서 잠시 빠져나온다. 입력받은 파일명을 생성할 수 없거나 오류가 생겼을 경우, 오류 메시지를 출력하고 다시 게임으로 돌아온다.

- 파일 출력

```
fprintf(data, "%d %d %d %d %d %c\n", players, HEIGHT, WIDTH, row,
col, turn);

for (x = 0; x < HEIGHT; x++) {
    for (y = 0; y < WIDTH; y++) {
        if (board[x][y] == 'O' || board[x][y] == 'X' ||
board[x][y] == 'Y') {
            fprintf(data, "%c", (char) board[x][y]);
        } else {
            fprintf(data, " ");
        }
    }
    fprintf(data, "\n");
}

refresh();
fclose(data);
return;
```

파일 생성 혹은 접근이 성공적이었다면, 우선 게임 정보를 저장하고 보드 상태를 그대로 저장한다. 단 상자 그리기 문자들은 저장할 때 오류가 생길 수 있으므로 공백으로 저장한다.

저장이 완료되었다면 `refresh()`를 통해 `curses` 모드로 돌아오고, 파일 스트림을 닫는다.

G. readSavedGame() 함수

이 함수는 저장된 게임 상태를 다시 로드하는 역할을 한다. 아래와 같은 인자를 받는다.

다.

- `int *players, int *row, int *col, int *turn`: 현재 게임 상태를 reference로 받는다.

- 파일명 입력

```
/*
    Read the existing game
*/
int** board;

FILE* data;
char filename[100];
char _, t;
int i, x, y;

printf("Filename to load: ");
scanf("%s", &filename);

data = fopen(filename, "r");
while (data == NULL) {
    printf("Failed to open file.\n");
    printf("Filename to load: ");
    scanf("%s", &filename);
    data = fopen(filename, "r");
}
```

파일명을 입력받는다. 파일 열기에 실패했을 경우 파일명을 다시 입력받는다.

- 게임 로드

```
fscanf(data, "%d %d %d %d %d %c", &(*players), &HEIGHT, &WIDTH,
&(*row), &(*col), &t);
*turn = (int) t; // turn is an integer

initscr(); // to use box drawing chars
```



```

board = initBoard(board);
fscanf(data, "%c", &_); // not necessarily needed

for (x = 0; x < HEIGHT; x++) {
    for (y = 0; y < WIDTH; y++) {
        fscanf(data, "%c", &t);
        if (t != ' ') {
            board[x][y] = t;
        }
    }
    fscanf(data, "%c", &_); // newline: not necessarily needed
}
fclose(data);
return board;

```

파일을 여는 데 성공했다면 먼저 게임 정보를 읽어온다.

`readSavedGame()`은 `gameStart()` 전에 호출되기 때문에, `initscr()`가 되어 있지 않으면 오목판에서 `ncurses` 라이브러리가 제공하는 상자 그리기 문자를 사용할 수 없다. 따라서 `initscr()`를 하고 파일에 적힌 크기의 오목판을 초기화하며, 캐릭터를 하나하나 읽어오면서 돌을 오목판 위에 배치한다. 완성된 오목판은 파일 스트림을 닫고 반환한다.

H. `paintBoard()` 함수

이 함수는 현재 오목판의 상태를 출력한다. 아래와 같은 인자를 받는다.

- `int **board, WINDOW *win, int row, int col`: 윈도우 및 현재 게임 정보.

```

/*
    Print the board to the given WINDOW
    using functions of the ncurses library.
*/
int x, y, attr;

for (x = 0; x < HEIGHT; x++) for (y = 0; y < WIDTH; y++) {
    switch (board[x][y]) {

```

```

        case 'O': attr = 3; break;
        case 'X': attr = 4; break;
        case 'Y': attr = 5; break;
        default: attr = 2; break;
    }
    wmove(win, x, y);
    if (x == row && y == col) {
        waddch(win, board[x][y] | COLOR_PAIR(attr) |
A_UNDERLINE);
    } else {
        waddch(win, board[x][y] | COLOR_PAIR(attr));
    }
}

wrefresh(win);
return;

```

오목판의 모든 칸에 대해 `waddch()`로 현재 `board`에 저장되어 있는 정보를 출력한다.

`COLOR_PAIR(n)`은 `main` 함수에서 선언해 둔 색상 정보에 액세스하는데, `switch` 문은 현재 칸의 정보에 알맞은 색상을 선택하는 일을 한다. 또한 현재 위치의 경우 `A_UNDERLINE`을 이용해 밑줄도 긋는다.

마지막에는 `wrefresh()`로 변경 내용을 윈도우에 반영한다.

I. `paintMenu()` 함수

이 함수는 현재 메뉴 상태를 출력한다. 아래와 같은 인자를 받는다.

- `WINDOW *menu, int turn`: 윈도우 및 현재 게임 정보.

```

clearMenu(menu);

// Paint informations
wmove(menu, 2, 2);
waddstr(menu, "Current Turn : ");
wmove(menu, 2, 17);
waddch(menu, (char) turn);

```

```
wmove(menu, 4, 2);
waddstr(menu, "Keyboard commands:");
wmove(menu, 5, 2);
waddstr(menu, "[1] Save");
wmove(menu, 6, 2);
waddstr(menu, "[2] Exit without save");

wrefresh(menu);
return;
```

waddstr()로 문자열을 추가하고, Current Turn 뒤에 붙는 현재 차례 인디케이터는 waddch()를 사용했다.

J. clearMenu(), paintErrorMessage(), clearErrorMessage(), paintWinMessage(), paintDrawMessage(), paintTerminationMessage() 함수

paintMenu()와 같은 방식으로 메뉴에 특정 문자열을 그리거나, 메뉴 윈도우의 모든 칸에 공백 문자를 addch() 함수로 메뉴 전체를 지운다. 가령 메뉴에 원래 7글자 문자열이 있었는데 11글자 문자열로 덮어쓴다면 문제가 없지만, 11글자 문자열이 있었는데 7글자 문자열로 덮어쓴다면 나머지 4글자는 업데이트되지 않기 때문에 메뉴 혹은 오류 메시지를 전부 지우는 함수가 필요하다.

3. 제작 내용

다음은 완성된 프로젝트 전체 코드이다.

```
#include <stdio.h>
#include <ncurses.h>
#include <stdlib.h>
#define KEY_SPACE ' ' // not defined in ncurses.h
#define KEY_Enter 10
#define START_ROW 5
#define START_COL 5

int HEIGHT, WIDTH;

int** initBoard(int **board) {
    board = (int**) malloc(sizeof(int*)*HEIGHT);
    int i, j;
```

```

    for (i = 0; i < HEIGHT; i++) {
        board[i] = (int*) malloc(sizeof(int)*WIDTH);
    }
    printf("%d %d\n", HEIGHT, WIDTH);
    board[0][0] = ACS_ULCORNER; //'┐'
    for (i = 1; i < WIDTH - 1; i++)
        board[0][i] = ACS_TTEE; //'┐'
    board[0][WIDTH - 1] = ACS_URCORNER; //'┐'
    for (i = 1; i < HEIGHT - 1; i++) {
        board[i][0] = ACS_LTEE; //'┐'
        for (j = 1; j < WIDTH - 1; j++)
            board[i][j] = ACS_PLUS; //'+'
        board[i][WIDTH - 1] = ACS_RTEE; //'┐'
    }
    board[HEIGHT - 1][0] = ACS_LLCORNER; //'┐'
    for (i = 1; i < WIDTH - 1; i++)
        board[HEIGHT - 1][i] = ACS_BTEE; //'┐'
    board[HEIGHT - 1][WIDTH - 1] = ACS_LRCORNER; //'┐'

    return board;
}

void saveGame(int **board, int players, int row, int col, int turn) {
    /*
        Save current game with given name
    */

    FILE* data;
    char filename[100];
    int i, x, y;

    endwin();

    printf("Filename to save: ");
    scanf("%s", &filename);

    data = fopen(filename, "w");
    if (data == NULL) {
        printf("Failed to write file.\n");
        refresh();
        return;
    }

    fprintf(data, "%d %d %d %d %d %c\n", players, HEIGHT, WIDTH, row, col, turn);

    for (x = 0; x < HEIGHT; x++) {
        for (y = 0; y < WIDTH; y++) {
            if (board[x][y] == 'O' || board[x][y] == 'X' || board[x][y] == 'Y') {
                fprintf(data, "%c", (char) board[x][y]);
            } else {
                fprintf(data, " ");
            }
        }
        fprintf(data, "\n");
    }
}

```

```

    }

    refresh();
    fclose(data);
    return;
}

int** readSavedGame(int *players, int *row, int *col, int *turn) {
    /*
        Read the existing game
    */
    int** board;

    FILE* data;
    char filename[100];
    char _, t;
    int i, x, y;

    printf("Filename to load: ");
    scanf("%s", &filename);

    data = fopen(filename, "r");
    while (data == NULL) {
        printf("Failed to open file.\n");
        printf("Filename to load: ");
        scanf("%s", &filename);
        data = fopen(filename, "r");
    }

    fscanf(data, "%d %d %d %d %d %c", &(*players), &HEIGHT, &WIDTH, &(*row), &(*col), &t);
    *turn = (int) t; // turn is an integer

    initscr(); // to use box drawing chars

    board = initBoard(board);
    fscanf(data, "%c", &_); // not necessarily needed

    for (x = 0; x < HEIGHT; x++) {
        for (y = 0; y < WIDTH; y++) {
            fscanf(data, "%c", &t);
            if (t != ' ') {
                board[x][y] = t;
            }
        }
        fscanf(data, "%c", &_); // newline: not necessarily needed
    }
    fclose(data);
    return board;
}

void paintBoard(int **board, WINDOW *win, int row, int col) {
    /*
        Print the board to the given WINDOW
        using functions of the ncurses library.
    */

```

```

int x, y, attr;

for (x = 0; x < HEIGHT; x++) for (y = 0; y < WIDTH; y++) {
    switch (board[x][y]) {
        case 'O': attr = 3; break;
        case 'X': attr = 4; break;
        case 'Y': attr = 5; break;
        default: attr = 2; break;
    }
    wmove(win, x, y);
    if (x == row && y == col) {
        waddch(win, board[x][y] | COLOR_PAIR(attr) | A_UNDERLINE);
    } else {
        waddch(win, board[x][y] | COLOR_PAIR(attr));
    }
}

wrefresh(win);
return;
}

void clearMenu(WINDOW *menu) {
    // Clear menu area
    int x, y;

    for (x = 0; x < 7; x++) for (y = 0; y < 60; y++) if (x != 3) {
        wmove(menu, x, y);
        waddch(menu, ' ');
    }

    wrefresh(menu);
    return;
}

void paintMenu(WINDOW *menu, int turn) {
    clearMenu(menu);

    // Paint informations
    wmove(menu, 2, 2);
    waddstr(menu, "Current Turn : ");
    wmove(menu, 2, 17);
    waddch(menu, (char) turn);
    wmove(menu, 4, 2);
    waddstr(menu, "Keyboard commands:");
    wmove(menu, 5, 2);
    waddstr(menu, "[1] Save");
    wmove(menu, 6, 2);
    waddstr(menu, "[2] Exit without save");

    wrefresh(menu);
    return;
}

void paintWinMessage(WINDOW *menu, int turn) {
    int attr;

```

```
switch (turn) {
    case 'O': attr = 3; break;
    case 'X': attr = 4; break;
    case 'Y': attr = 5; break;
}

clearMenu(menu);

wattron(menu, COLOR_PAIR(attr));
wmove(menu, 2, 2);
waddstr(menu, "PLAYER x WIN!!");

wmove(menu, 2, 9);
switch (turn) {
    case 'O':
        waddstr(menu, "1");
        break;
    case 'X':
        waddstr(menu, "2");
        break;
    case 'Y':
        waddstr(menu, "3");
        break;
}
wattroff(menu, COLOR_PAIR(attr));

wmove(menu, 3, 2);
waddstr(menu, "Press any key to exit.");

wrefresh(menu);
return;
}

void paintDrawMessage(WINDOW *menu) {
    clearMenu(menu);

    wmove(menu, 2, 2);
    waddstr(menu, "DRAW");
    wmove(menu, 3, 2);
    waddstr(menu, "Press any key to exit.");

    wrefresh(menu);
    return;
}

void clearErrorMessage(WINDOW *menu) {
    int y;

    for (y = 0; y < 60; y++) {
        wmove(menu, 3, y);
        waddch(menu, ' ');
    }

    wrefresh(menu);
    return;
}
```

```
}

void paintTerminationMessage(WINDOW *menu) {
    clearMenu(menu);
    clearErrorMessage(menu);

    wmove(menu, 2, 2);
    waddstr(menu, "Game terminated");

    wmove(menu, 3, 2);
    waddstr(menu, "Press any key to exit.");

    wrefresh(menu);
    return;
}

void paintErrorMessage(WINDOW *menu, char str[]) {
    wmove(menu, 3, 2);
    watttrn(menu, COLOR_PAIR(6));
    waddstr(menu, str);
    wattroff(menu, COLOR_PAIR(6));

    wrefresh(menu);
    return;
}

int checkWin(int **board, int players) {
    /*
     * Check if the game is over.
     * Returns the winner if the game is over; -1 if draw; else returns 0.
     */

    int len = 5, stoneCount = 0;
    int x, y, d, i, j, k, flag, value;

    // Directions to check:
    int dxarr[8] = {1, 1, 1, 0, 0, -1, -1, -1};
    int dyarr[8] = {1, 0, -1, 1, -1, 1, 0, -1};
    int dx, dy;

    // 4-mok when 3 players
    if (players == 3) len = 4;

    // Check for all cells in board
    for (x = 0; x < HEIGHT; x++) for (y = 0; y < WIDTH; y++) {
        value = board[x][y];

        if (
            (players == 2 && value != 'X' && value != 'O') ||
            (players == 3 && value != 'X' && value != 'O' && value != 'Y')
        ) continue; // Stone not placed here
        stoneCount++;
        for (d = 0; d < 8; d++) {
            dx = dxarr[d];
            dy = dyarr[d];
```



```

    flag = 1;

    i = x;
    j = y;

    for (k = 0; k < len; k++) {
        i += dx;
        j += dy;

        // If the checking coordinates goes out of bounds, check next case.
        if (i < 0) break;
        if (i > HEIGHT - 1) break;
        if (j < 0) break;
        if (j > WIDTH - 1) break;

        if (board[i][j] == value) {
            flag++;
            if (flag == len) return value;
        } else {
            break;
        }
    }
}

if (x * y == stoneCount) {
    return -1; // DRAW
} else {
    return 0;
}
}

int Action(WINDOW *win, WINDOW *menu, int **board, int keyin, int *row, int *col, int *turn, int
players) {
    /*
        Following right after the keyboard input,
        performs a corresponding action.

        returns 0 if game is not over,
        1 if game is over.
    */

    int check, error = 0;

    switch (keyin) {
        case KEY_DOWN:
            if (*row < HEIGHT - 1) (*row)++;
            break;
        case KEY_UP:
            if (*row > 0) (*row)--;
            break;
        case KEY_RIGHT:
            if (*col < WIDTH - 1) (*col)++;
            break;
        case KEY_LEFT:

```

```

    if (*col > 0) (*col)--;
    break;
case KEY_Enter:
case KEY_SPACE:
    // O - X - Y - O - X - Y - ...
    if (board[*row][*col] == 'O' ||
        board[*row][*col] == 'X' ||
        board[*row][*col] == 'Y') {
        // Stone already exists
        error = 1;
    } else {
        board[*row][*col] = *turn;

        if (players == 2) {
            switch (*turn) {
                case 'O': *turn = 'X'; break;
                case 'X': *turn = 'O'; break;
            }
        } else if (players == 3) {
            switch (*turn) {
                case 'O': *turn = 'X'; break;
                case 'X': *turn = 'Y'; break;
                case 'Y': *turn = 'O'; break;
            }
        }
    }
    break;
case '1':
    keypad(win, FALSE);
    saveGame(board, players, *row, *col, *turn);
    keypad(win, TRUE);
    break;
case '2':
    paintTerminationMessage(menu);
    return 1;
    break;
}

wmove(win, *row, *col);

if (error) {
    paintErrorMessage(menu, "Stone already exists there!");
} else {
    clearErrorMessage(menu);
}

check = checkWin(board, players);

if (check != 0) {
    if (check == -1) { // DRAW
        paintDrawMessage(menu);
        return 1;
    } else { // WIN
        paintWinMessage(menu, check);
        return 1;
    }
}

```

```
    }  
}  
  
wrefresh(win);  
wrefresh(menu);  
  
return 0;  
}  
  
void gameStart(WINDOW *win, WINDOW *menu, int load, int** loadedBoard, int players, int row, int col,  
int turn) {  
    int **board;  
    int finished = 0;  
    int keyin, result;  
    wmove(win, row, col);  
  
    // Initializing the board  
    if (load == 'Y' || load == 'y') {  
        board = loadedBoard;  
    } else {  
        board = initBoard(board);  
    }  
  
    while (1) {  
        /*  
         * This While loop constantly loops in order to  
         * draw every frame of the WINDOW.  
         */  
  
        // Paint board  
        paintBoard(board, win, row, col);  
  
        // Terminate game if finished  
        if (finished) {  
            keyin = wgetch(win);  
            break;  
        }  
  
        // Paint menu  
        paintMenu(menu, turn);  
  
        // Move cursor to last action point  
        wmove(win, row, col);  
  
        // Get keyboard input  
        keyin = wgetch(win);  
  
        // Do action and save its result according to the keypress  
        result = Action(win, menu, board, keyin, &row, &col, &turn, players);  
  
        // Refresh windows  
        wrefresh(win);  
        wrefresh(menu);  
  
        if (result == 1) {
```

```
        // Game is over
        finished = 1;
    }
}
return;
}

int main() {
    WINDOW *win, *menu;
    char load;
    int players, row = 0, col = 0, turn = '0';
    int** loadedBoard;

    /*
        Prompts to ask options of the game
    */

    // Option #1: load saved game
    printf("Load saved game? [Y/N] : ");
    scanf("%c", &load);

    // Invalid option checking
    while (load != 'Y' && load != 'N' && load != 'y' && load != 'n') {
        printf("Invalid option.\nAvailable options are 'Y', 'N' : ");
        scanf("%c", &load);
    }

    if (load == 'Y' || load == 'y') {
        loadedBoard = readSavedGame(&players, &row, &col, &turn);
    } else {
        // Option #2-1: height
        printf("Enter the HEIGHT of the board [8 - 32] : ");
        scanf("%d", &HEIGHT);

        while (HEIGHT < 8 || HEIGHT > 32) {
            printf("Invalid height.\nHeight should be in [8 - 32] : ");
            scanf("%d", &HEIGHT);
        }

        // Option #2-2: width
        printf("Enter the WIDTH of the board [8 - 64] : ");
        scanf("%d", &WIDTH);

        while (WIDTH < 8 || WIDTH > 64) {
            printf("Invalid width.\nWidth should be in [8 - 64] : ");
            scanf("%d", &WIDTH);
        }

        // Option #3: players
        printf("Enter the number of players [2 - 3] : ");
        scanf("%d", &players);

        // Invalid option checking
        while (players != 2 && players != 3) {
            printf("Invalid option.\nAvailable options are 2 - 3 : ");
        }
    }
}
```

```

    scanf("%d", &players);
}
}

// Window initialization
initscr();
win = newwin(HEIGHT, WIDTH, 0, 0);
menu = newwin(7, 60, HEIGHT, 0);
noecho();
keypad(win, TRUE);
cbreak();

// Color initialization
start_color();

init_pair(1, COLOR_WHITE, COLOR_BLACK); // [1] default = white on black
init_pair(2, COLOR_BLACK, COLOR_YELLOW); // [2] board = black on yellow

if (players == 2) {
    init_pair(3, COLOR_WHITE, COLOR_BLACK); // [3] O = white on black
    init_pair(4, COLOR_BLACK, COLOR_WHITE); // [4] X = black on white
} else if (players == 3) {
    init_pair(3, COLOR_BLACK, COLOR_RED); // [3] O = black on red
    init_pair(4, COLOR_BLACK, COLOR_GREEN); // [4] X = black on green
    init_pair(5, COLOR_WHITE, COLOR_BLUE); // [5] Y = white on blue
}

init_pair(6, COLOR_WHITE, COLOR_RED); // [6] error = red on white

gameStart(win, menu, load, loadedBoard, players, row, col, turn);

// Game ended
endwin();
return 0;
}

```

4. 시험 내용

다음과 같은 기준들을 시험하였다.

- 커서는 오목판을 벗어나지 않는가?
 - 커서를 가장자리에서 벗어나는 방향으로 키 입력을 넣어 보면서 시험했다. 벗어나지 않았다.
- 게임 종료 조건을 체크하는 알고리즘은 완벽하게 구현되었는가?
 - 2인 플레이에서 6~9개의 돌, 3인 플레이에서 5~7개의 돌이 연속적으로 놓이는 등의

특수한 조건에 대해서 실험하였고, 정상적으로 승자를 출력했다.

- 오목판을 여러 크기에서 전부 채워 보았고, 정상적으로 무승부로 종료되었다.
- 색상은 잘 출력되는가?
 - 새 색상을 정의해 사용하려고 했으나, `documentation`과는 다르게 색상 정의를 지원하지 않았고, 색상을 정의할 경우 정의한 색상은 검은색이 되어 정상적으로 보이지 않았다. 따라서 오목판을 그리는 데에는 기본적으로 제공하는 8개의 색상 중에서 사용했다.
- 파일은 잘 저장되고 잘 불러와지는가?
 - 상자 그리기 문자들(┌, ┐, └, ┘ 등)이 제대로 저장되지 않았다. 상자 그리기 문자들은 문자 코드가 있는 것이 아니라 `ncurses` 라이브러리에서 특수하게 처리하는 것으로 추정했고, 상자 그리기 문자가 있는 칸은 공백으로 저장했다.
 - 파일을 불러올 때 상자 그리기 문자들(┌, ┐, └, ┘ 등)이 전부 캐럿(^)으로 출력되었다. `initscr()`를 하지 않으면 상자 그리기 문자들이 초기화되지 않아 사용할 수 없었다. 오목판을 초기화하기 직전에 `initscr()`를 하는 것으로 해결했다.

5. 평가 내용

- **장점** : 2인, 3인 플레이 모드를 지원하며, 게임을 도중에 저장하여 나중에 불러올 수 있고, 오목판에 색상이 들어가 있어 색상이 없을 때보다 훨씬 가독성이 높다.
- **단점** : 2인 이상의 플레이어가 존재할 것을 필요로 하며, 저장된 파일을 조작하는 것이 가능하고, `curses`를 지원하는 Linux 환경에 국한되어 실행되기 때문에 다른 OS에서는 실행할 수 없다는 치명적인 단점이 존재한다.

색상 관련 기능을 개발하는 중 색상이 제멋대로 출력되는 현상을 경험하였다. 원인을 분석하다가, `char` 형식의 변수에 `character code %d`를 사용해 입력받았던 것을 임시 변수를 이용해 처리했더니 해결되었다. `int`는 32비트, `char`는 8비트 자료형이기 때문에 입력을 받을 때 예기치 않은 오류가 발생했던 것이다. `character code` 형식으로 입력받으려면 임시 변수를 두는 것이 좋겠다.

이 프로젝트는 오목판 크기에 한계를 둔다는 점에서 과도하게 큰 메모리를 차지해 예기치 않은 오류가 발생할 확률이 낮다는 점에서 안정성이 높지만, 파일 저장 부분에서 신뢰성이 다소 떨어진다. 저장된 파일은 오목판을 그대로 저장해 사람이 읽고 편집하기 쉬운 형태로 되어 있어 저장된 파일을 수정해 원하는 위치에 원하는 돌을 임의로 무한정 놓은 후 불러와 한 플레이어에게 유리하게 만들 수 있다. 신뢰성을 개선하려면 `checksum`을 도입해 저장된 파일을 불러올 때 사용자가 임의로 수정한 것은 아닌지 확인하는 알고리즘을 도입하거나, 사람이 편집하기 어려운 형태로 저장하는 것이 필요할 것이다.

V. 기타

1. 자체 평가

기술적인 요소 전반적으로는 모듈화에 초점을 맞춰 프로젝트를 진행하였다. 또한 주석이 없더라도 코드 내용을 이해할 수 있도록 변수 이름 지정 등에 최대한 신경을 썼으며, 간결한 코드를 작성하려고 노력하였다.

사용자 경험(UX)적인 면에서는 지정된 개발 목표 이외에도 오목 게임으로서 필수적인 추가 기능들을 더 구현해, 색상을 도입함으로써 사용자가 더 직관적으로 게임을 할 수 있도록 배려하는 등의 신경을 썼다.