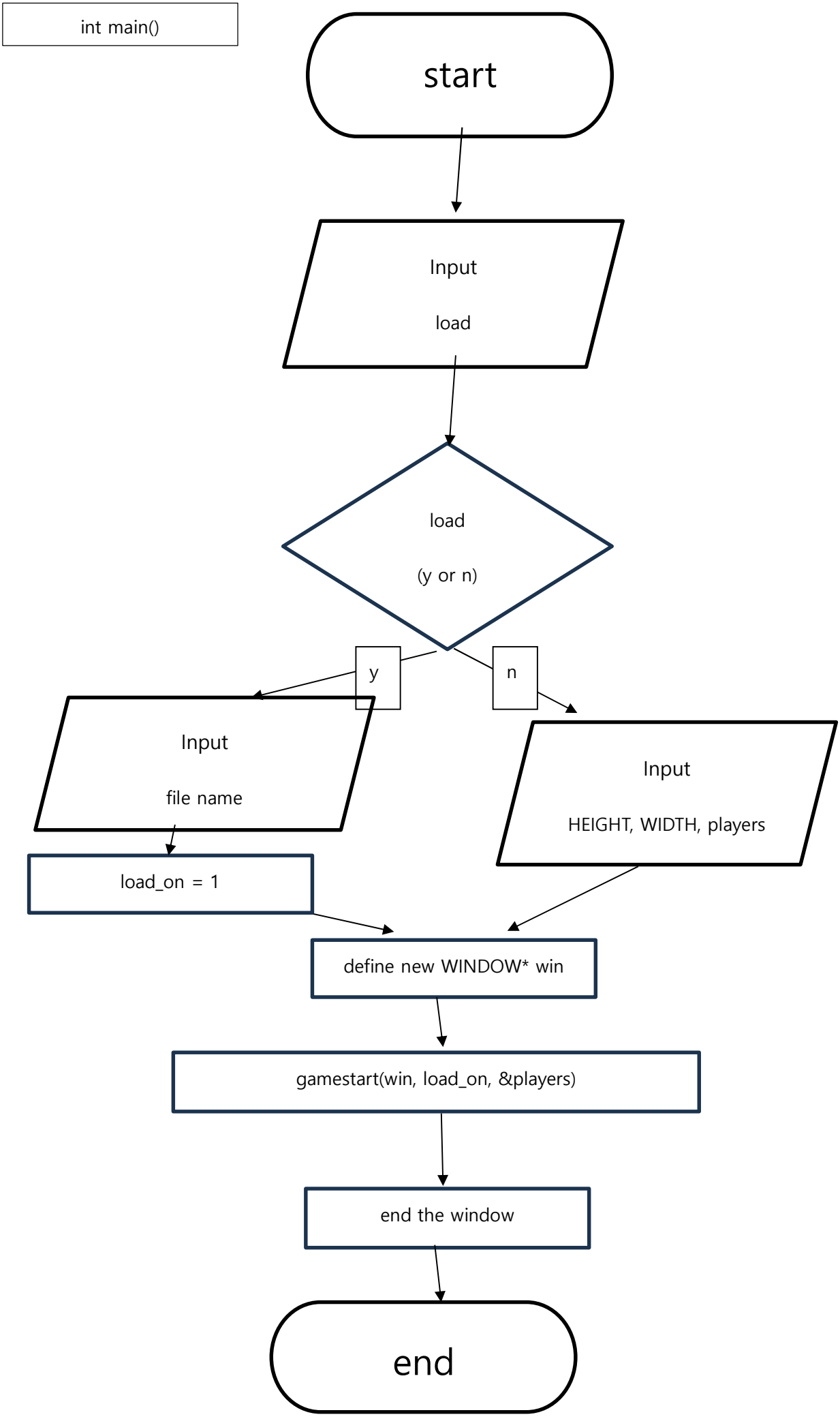
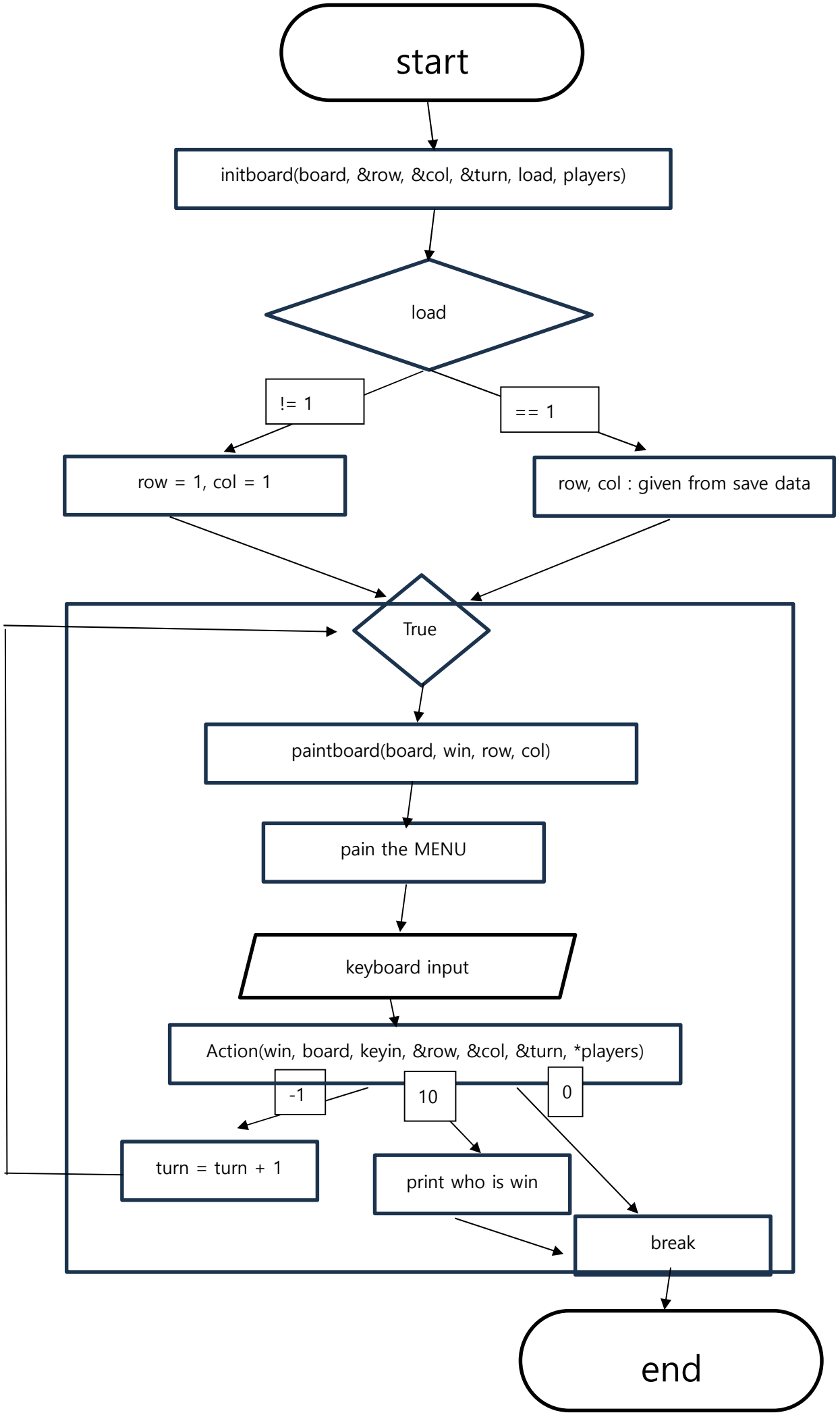
HW1 결과보고서

20211522

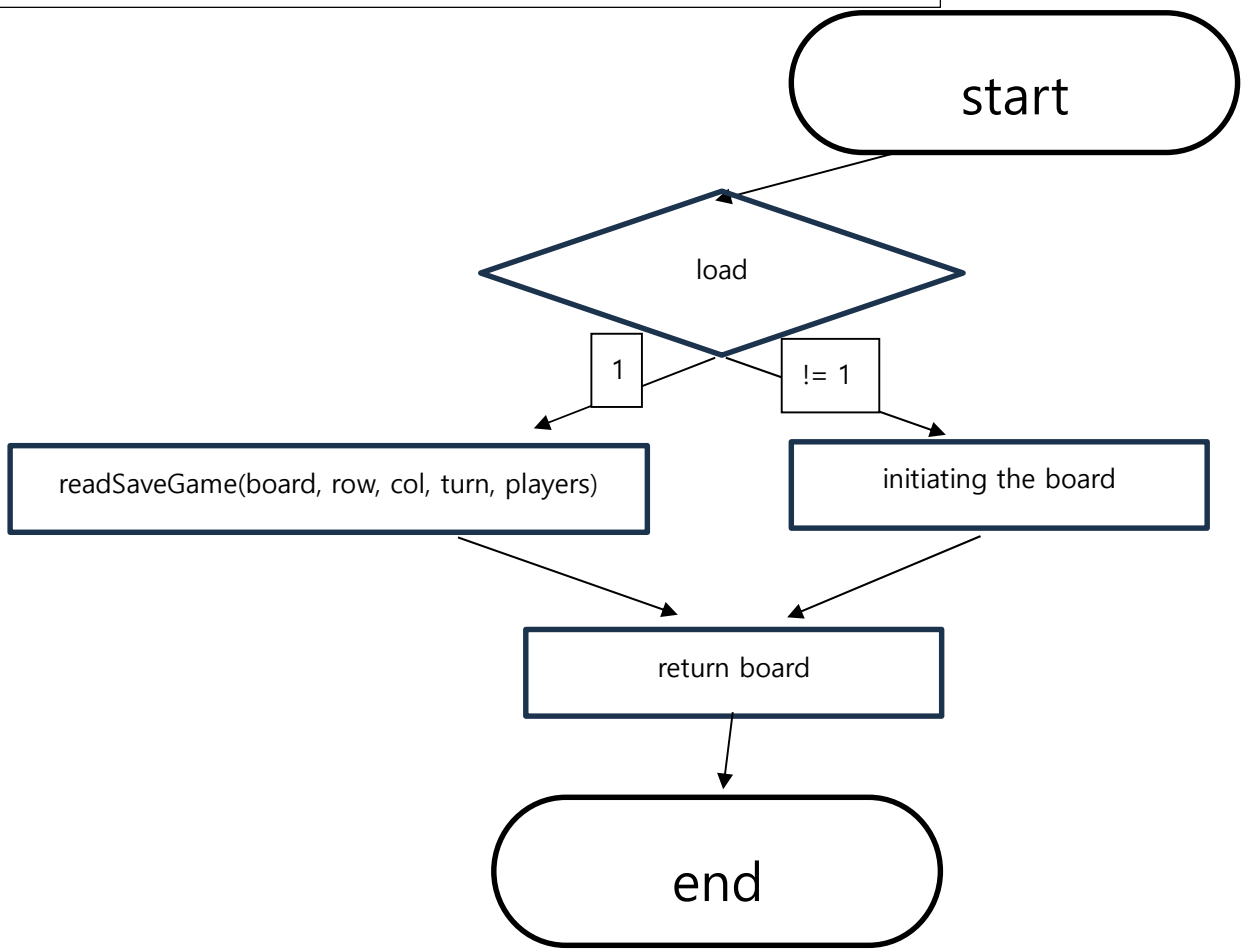
김정환


int main()
start
Input
load
load
(y or n)
y
n
Input
file name
load_on = 1
Input
HEIGHT, WIDTH, players
define new WINDOW* win
gamestart(win, load_on, &players)
end the window
end

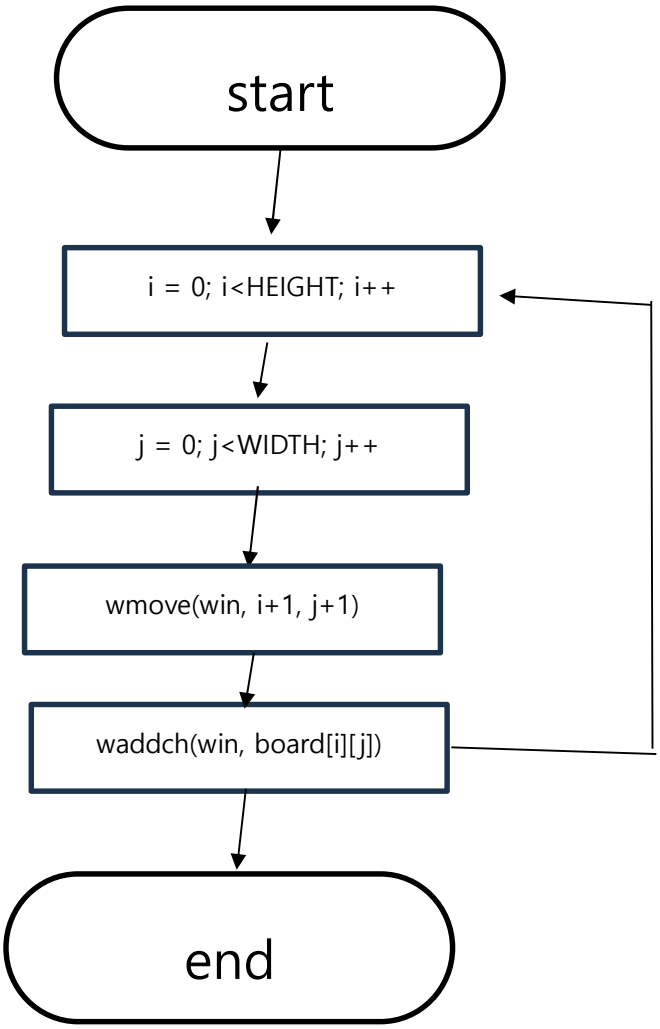
void gamestart(WINDOW* win, int load, int*players)



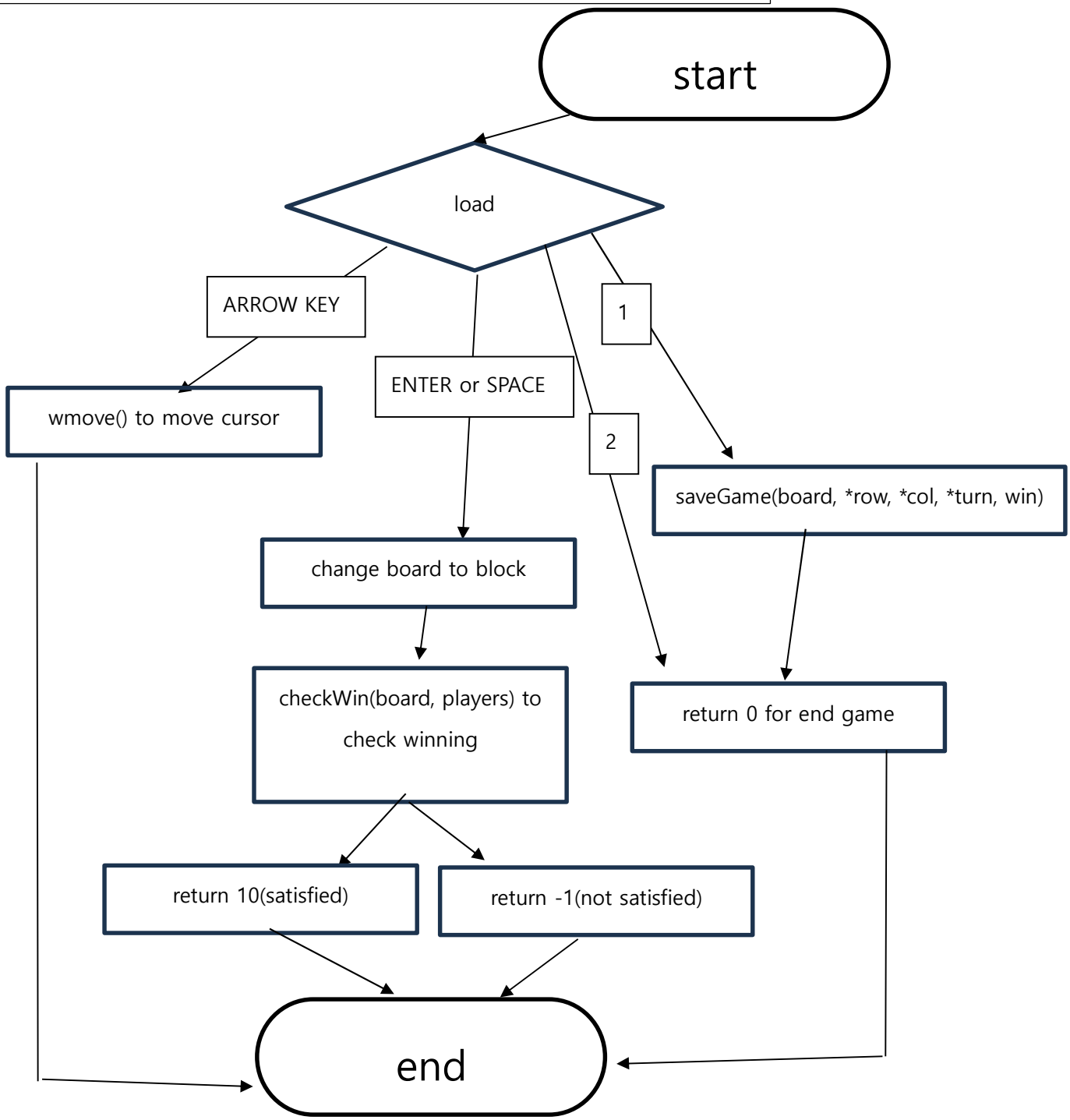
int** initboard(int** board, int* row, int* col, int* turn, int load, int* players)



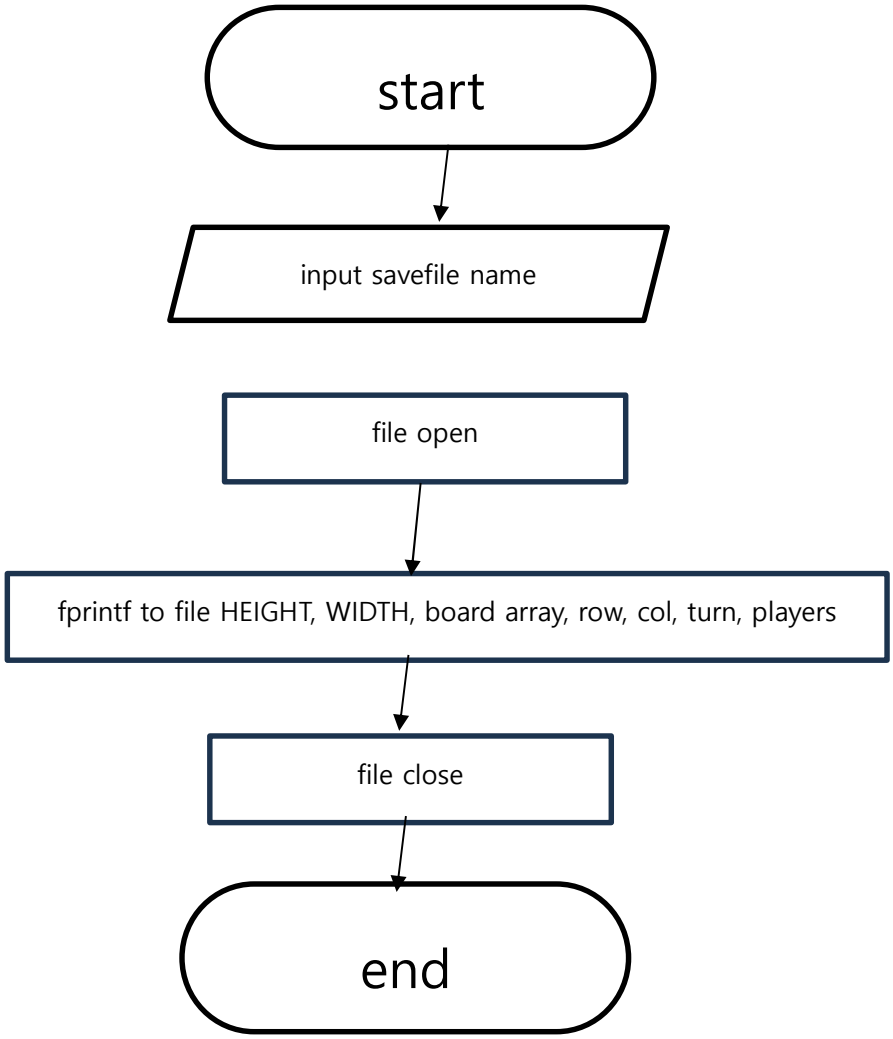
void paintBoard(int** board, WINDOW* win, int row, int col)



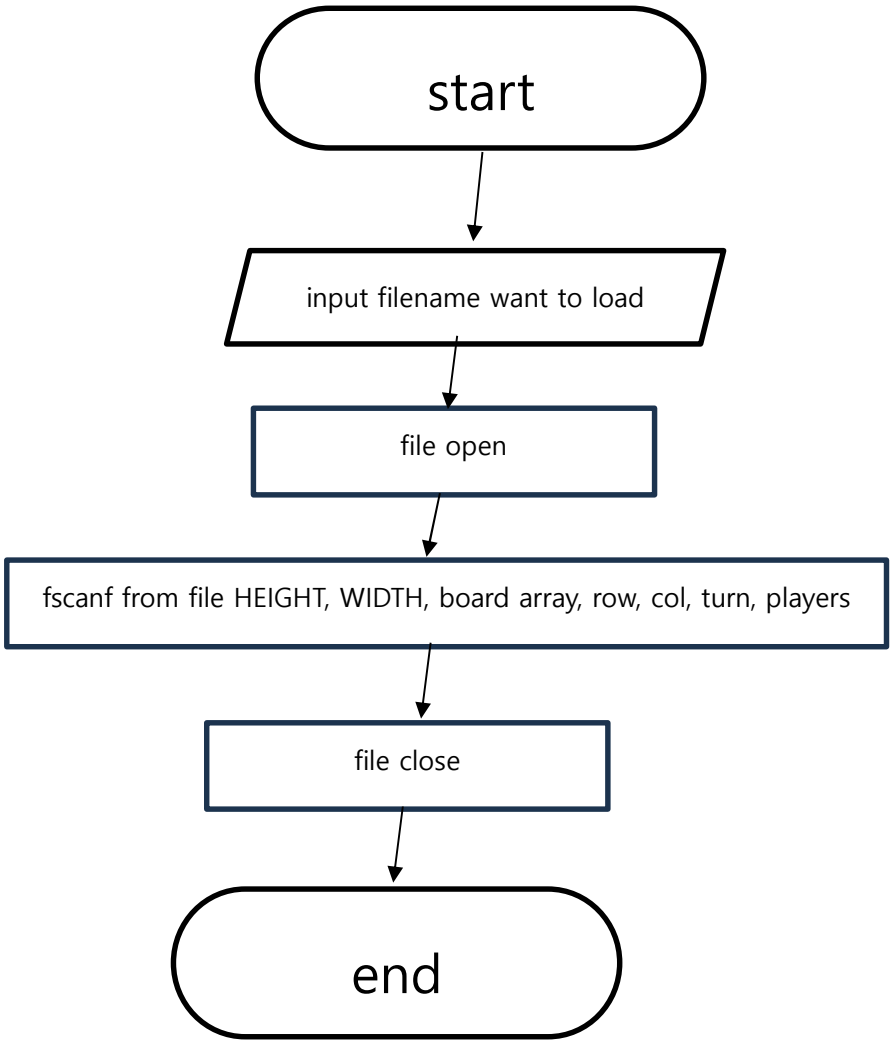
int Action(WINDOW* win, int** board, int keyin, int* row, int* col, int* turn, int players)



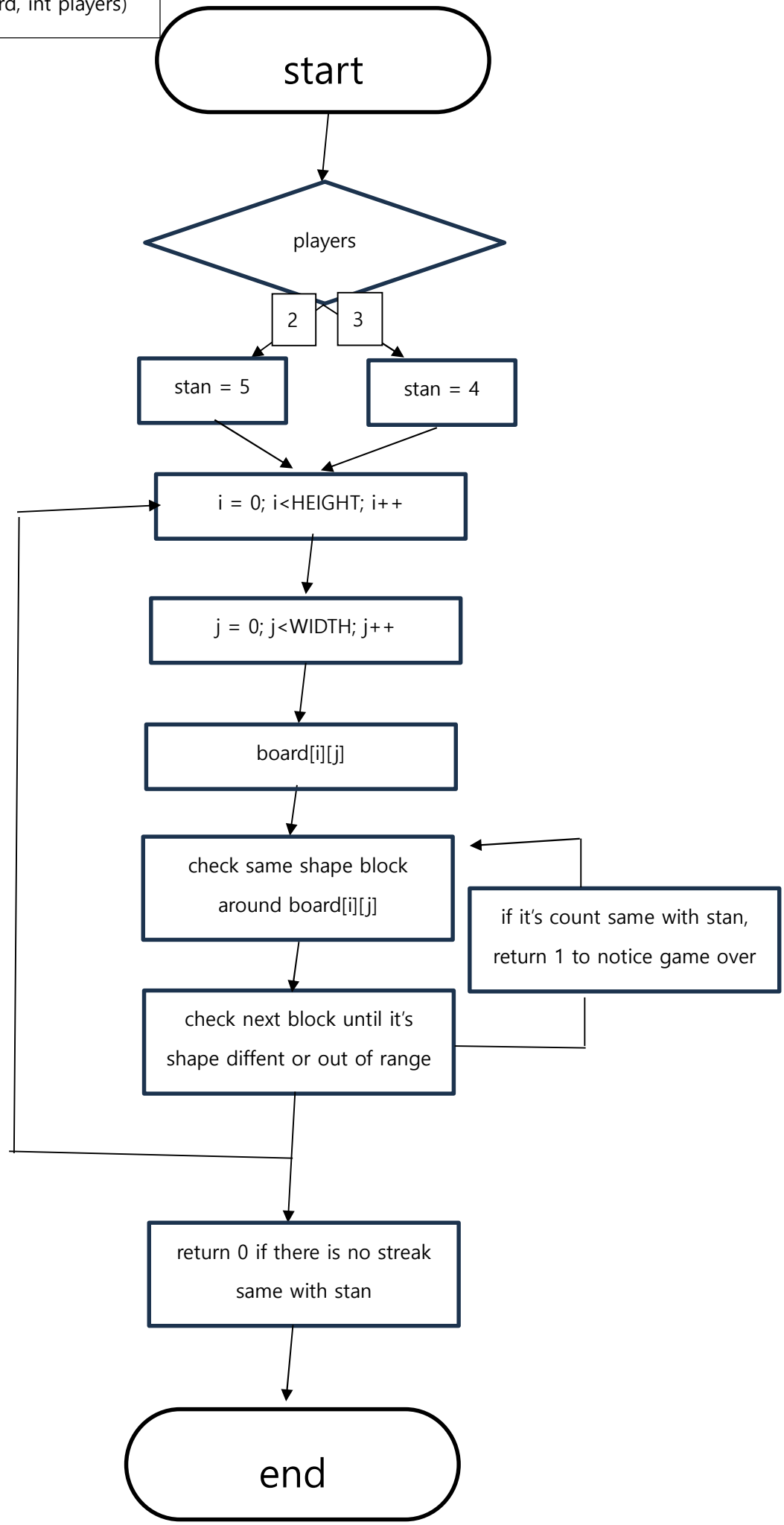
void saveGame(int** board, int row, int col, int turn, int players, WINDOW* win)



int** readSaveGame(int** board, int* row, int* col, int* turn, int* players)



int checkWin(int** board, int players)



<구현 설명>

main 함수가 실행되었을 때 load로 게임을 로드하고 싶은지 확인하는 문장 출력과 함께 load 변수에 입력을 받는다. 여기에서 load 값에는 'y'나 'n'만 받을 수 있도록 이외의 값이 들어왔을 때 다시 입력하도록 반복문 안에 넣어주었다. load가 'y'인 경우에는 이후 다른 함수에 로드된 게임인지 전달해줄 인자인 정수 변수인 load_on의 값을 1로 하고, 로드 시에 파일이름을 입력하는 줄을 고려하여 이후 window의 위치가 1칸 아래 가도록 window의 위치를 x = 0, y = 1로 설정해주었다. load가 'n'인 경우에는 load_on의 값을 0으로 하고 window의 위치를 x = 0, y = 0으로 정했다. 이후에 HEIGHT, WIDTH 값을 입력받고 플레이어 수를 입력받았다. 그 후 initscr()로 window가 시작할 수 있도록 하고 WINDOW* win 변수에 newwin()함수를 통해 새로운 window를 정의하였다. 위치 인자에는 위에서 정한 y, x 값을 넣어주었고, 크기는 밑에 MENU 칸을 고려하여 여유있는 크기가 되도록 했다. intrflush() 함수와 keypad() 함수로 화살표 키와 같은 특수키 입력을 인식할 수 있도록 활성화했다. 이후에 입력되는 키보드 입력은 원할 때만 보이도록 noecho()로 평소에는 출력하지 않도록 했고 그 상태에서 gameStart() 함수를 실행했다. gameStart() 함수의 동작이 끝난 후에는 endwin()으로 본래 명령 스크립트로 돌아갈 수 있도록 window를 종료했다.

gameStart() 함수는 main함수로부터 WINDOW의 주소인 win과 게임의 로드여부를 나타낸 load_on을 정수값으로, 플레이어의 수인 players를 정수 포인터값 인자로 받는다. 스켈레톤 코드 제외하고 추가한 변수들은 플레이어들의 순서를 판단할 정수 turn과 플레이어별 블록의 기호의 배열인 block이 있다. board는 initBoard() 함수를 호출하여 board 배열을 초기화하는 작업을 거친다. 그 후 저장된 게임을 로드한 것이 아닐 경우 가장 왼쪽 위 칸에서 시작할 수 있도록 row값과 col값을 1로 초기화해준다. 그 후 while 반복문을 항상 참으로 하여 무한 반복하면서 paintBoard함수를 호출하여 board를 화면에 출력해주고 주어진 대로 MENU를 출력해주는 명령어들을 배치했다. 그 후 커서가 row, col의 위치로 이동하도록 했고, wgetch()함수로 키 입력값을 keyin에 입력받는다. keyin값은 Action

함수에 같이 전달되어 Action 함수의 결과 return 값이 act에 저장되고 act가 -1일 때는 다음 턴으로 진행하고, 10일 때는 그 턴의 플레이어가 승리 판정이므로 주어진 양식에 맞게 승리 판정을 출력하고 아무 키를 입력받을 때까지 기다리다가 입력받으면 break로 반복문을 종료한다. act가 0일 때는 종료하는 명령이므로 break로 반복문을 탈출한다. 반복문 종료 후에는 동적할당해준 board의 배열들을 free로 해제해주고 함수를 종료한다.

initBoard() 함수는 나머지는 스켈레톤 코드 그대로이지만 load 인자가 1일 때 readSaveGame() 함수를 호출하고 board를 return하는 부분을 추가로 구현하였다. 또한 players의 수가 2인 또는 3인으로 설정하는 기능을 추가하면서 데이터를 load할 때 player의 수도 load할 수 있도록 인자에 int* players를 추가하였고 이를 readSaveGame() 함수에 전달한다.

paintBoard() 함수는 board 배열을 포인터 주소로 받고 WINDOW를 포인터로 받고 row와 col 값을 받는다. i는 0부터 HEIGHT-1까지 j는 0부터 WIDTH-1까지 for문을 통해 반복하면서 wmove로 위치를 이동시켜놓고 waddch로 인자로 받은 WINDOW에 board 배열을 출력하도록 하였다.

Action() 함수는 WINDOW를 포인터 값으로 인자를 받고, board를 정수 배열 포인터 값으로 인자를 받고, keyin과 players 값을 정수로 인자를 받는다. 그리고 row와 col, turn을 정수 포인터 값을 받는다. Action() 함수 안에도 block 배열을 gameStart() 함수와 똑같이 만들어 놓았고, 그 뒷부분에 keyin의 값에 따른 동작들을 if 문으로 구분해놓았다. 우선 keyin 값이 화살표 키일 때이다. 우선 KEY_UP일 때는 wmove(win, --*row, *col)로 우선 해당 위치로 커서로 이동하도록 하였고 *row--; 로 원래 주소의 값도 수정되어 새로고침되도록 하였다. 또한 *row>1의 조건일 때만 실행되도록 하여서 보드의 칸을 넘어가지 않도록 구현했다. KEY_DOWN일 때는 wmove(win, ++*row, *col)로 커서를 이동하고 *row++로 원래의 값도 수정되게 하였다. 이 경우에는 *row<HEIGHT인 경우에만 실행하도록 해서 보드의 칸을 넘어가지 않게 하였다. KEY_LEFT와 KEY_RIGHT의 경우에는 row가 아닌 col의 값을 변화시키고 실행 범위가 구간 (1, WIDTH)의 범위가 되도록 하여

구현하여 화살표 키에 따른 커서 이동을 구현했다. 그 다음은 Enter와 Space 키를 눌렀을 때 돌을 놓도록 구현하는 것이다. 우선 플레이어가 2명일 때와 3명일 때 모두 돌이 놓여있을 때 돌을 놓지 않도록 board의 해당 칸이 'O', 'X', 'Y'일 때는 돌을 놓지 않도록 필터링하였다. 돌의 종류는 turn 값과 players 값을 이용하여 구분하였다. players 수로 나눈 나머지를 이용하여 구현하였고 가장 처음이 'O', 2번째가 'X', 마지막이 'Y'로 되어 있고 Action() 함수 내에 구현된 block 배열을 통해서 놓도록 하였다. 돌을 놓고 나서는 checkWin() 함수를 res 변수에 값을 리턴하도록 하였고 이 값에 따라 res가 1이 될 경우 10을 return하여 승리판정을 전달하도록 하였다. 승리판정이 아닐 경우 -1을 return하여 게임이 진행되도록 했다. 10을 return하는 이유로는 1을 return 하였을 때 다른 결과에 섞이는 오류가 있어서 다른 값으로 특정하였다. '1'을 눌렀을 때는 saveGame() 함수를 호출하여 게임을 저장한 후 0을 return하여 게임을 끄는 값을 전달하도록 하였다. '2'를 눌렀을 때는 다른 동작없이 게임을 끄는 값을 전달하도록 하여 save없이 게임 종료하도록 했다.

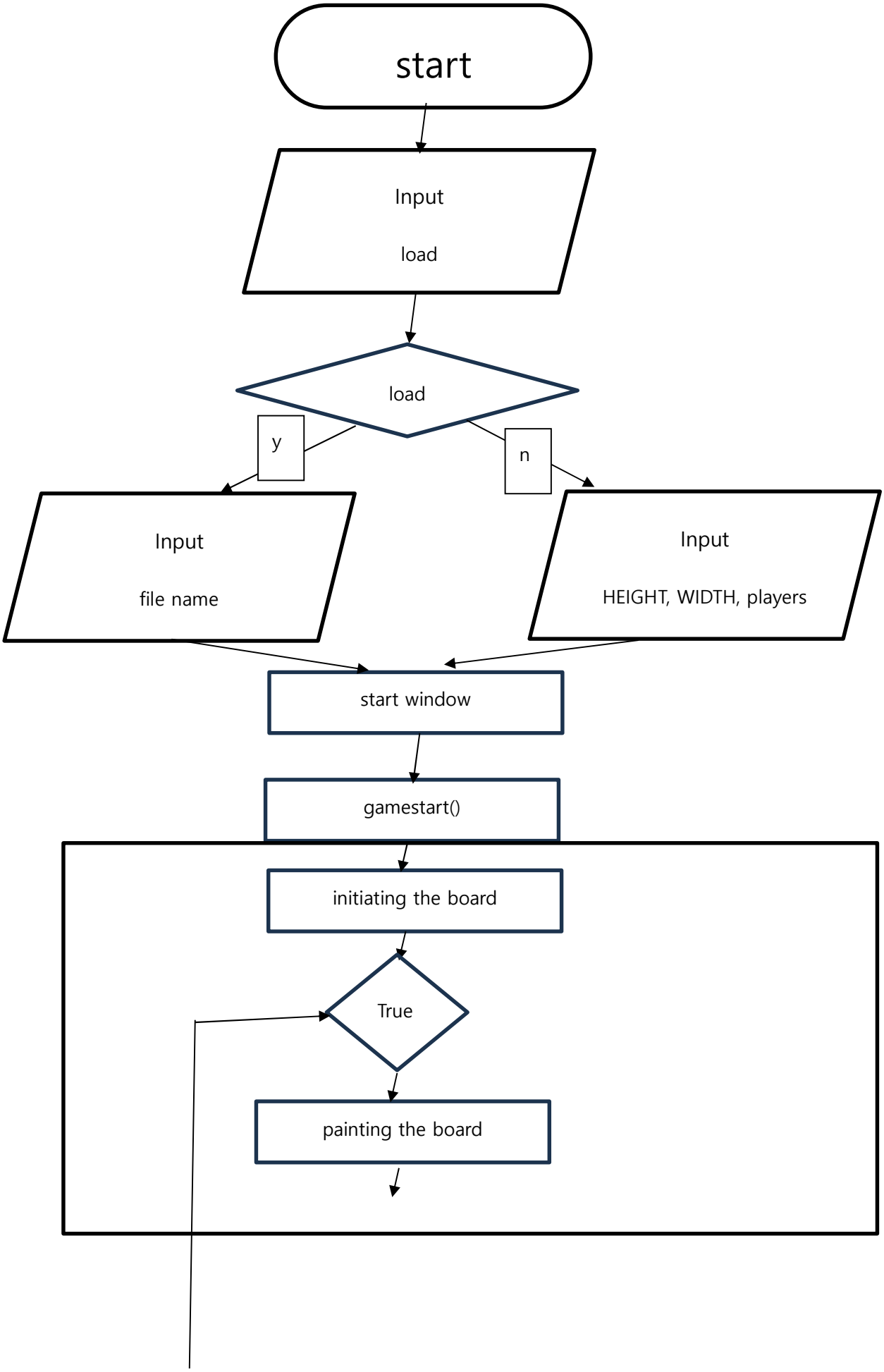
다음으로 checkWin() 함수이다. checkWin() 함수는 board를 전부 탐색하는 방식으로 구현하였다. 함수가 받는 인자는 board 배열을 포인터로 받는 인자와 players 즉 player 수만 받도록 했다. 구현은 우선 함수 내에 정수 배열 dx, dy를 선언하여 승리 판정 시에 탐색해보아야 하는 방향 4가지를 단위 벡터로 나타냈다. 그 뒤 승리의 기준을 인자로 받은 players 값에 따라서 players가 2일 때는 stan 값을 5로, players가 3일 때는 4로 정해서 승리의 기준이 유저 수에 따라 달라지도록 하였다. 그 후 블록의 연속으로 이어지는 개수를 세기 위한 cnt 변수를 선언하고, 이중 for문으로 i는 0부터 HEIGHT-1까지, j는 0부터 WIDTH-1까지 반복하도록 하였다. 그 후 board[i][j]가 'O', 'X', 'Y' 중 하나일 때 미리 선언해둔 단위 벡터의 방향으로의 탐색을 진행한다. 각각의 방향에서 탐색 후 board의 범위 내에 있고 기존의 board[i][j]와 같은 문자일 경우에 cnt 값을 2로 정해주면서 다른 문자가 나오거나 board의 범위를 벗어날 때까지 while문을 통해서 같은 방향의 벡터로 탐색하고 while문을 탈출했을 때 stan 값과 비교하여 승리조건을 달성했는지 확인한다. 마찬가지로 승리조건 확인 시에도 다음 블록이 다른 값을 갖고 있

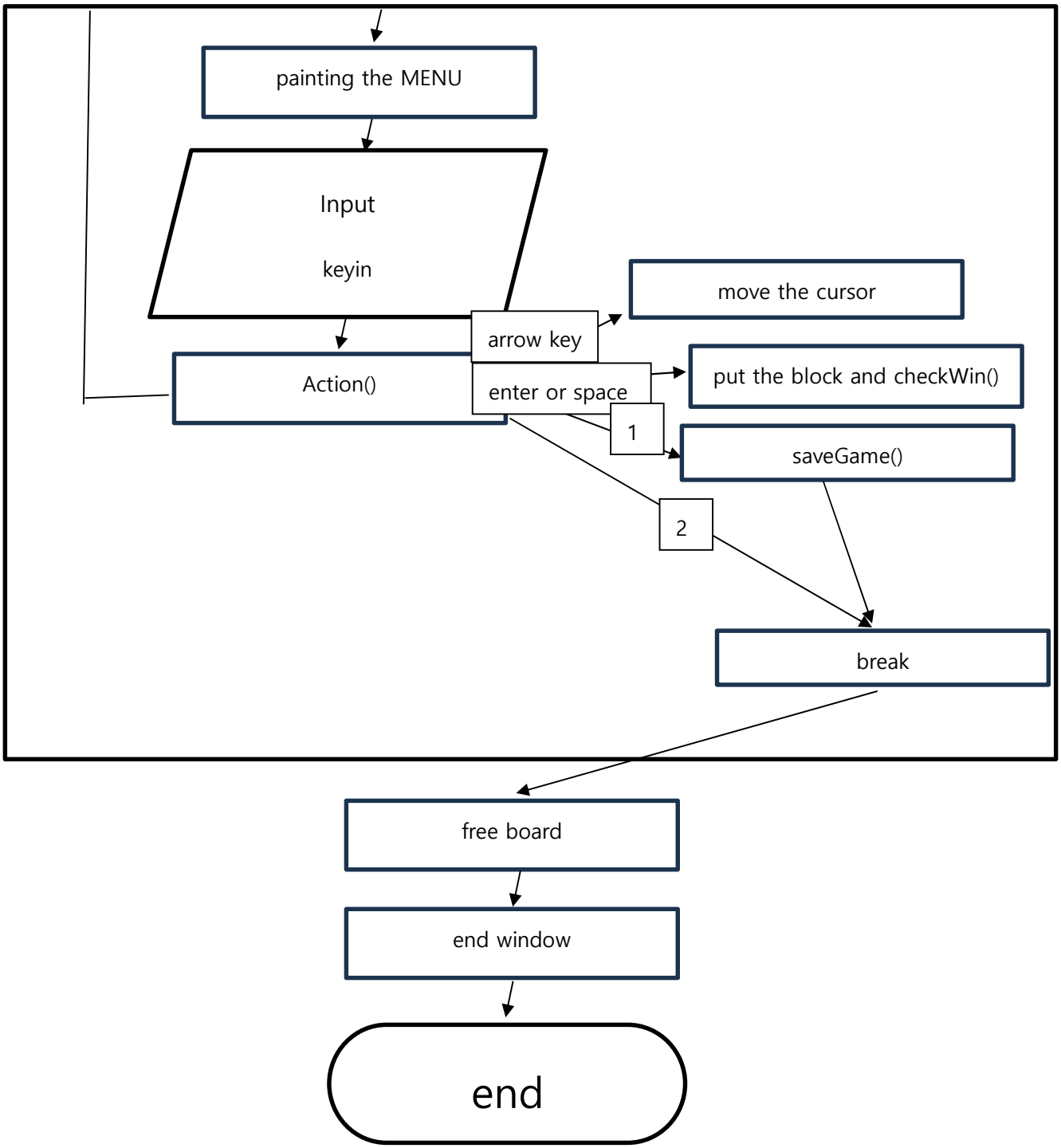
는지 필터링하는 것까지 마친 후 1을 return하여 승리를 전달하고 전체 탐색을 했는데도 승리 조건을 만족하는 경우가 없었을 경우에는 0을 return한다.

saveGame() 함수는 주어진 함수의 인자에 WINDOW* win와 int players 인자를 추가하여 구현하였다. 파일 이름을 입력받을 char 타입의 str 변수를 선언하였고, 여기서는 키보드 입력이 보이도록 echo() 함수를 사용하였다. 그 후 커서가 마지막 돌을 둔 위치로 이동하도록 wmove로 (row, col) 위치로 이동하고, wgetstr을 이용하여 인자로 전달받은 window에서 입력받도록 하였다. fopen으로 str에 입력받은 파일이름으로 열고 HEIGHT, WIDTH, board 배열 전체, row, col, turn, players 값을 입력한다. row, col은 커서도 이어서 할 수 있도록 해주고, turn은 이어서 할 때 순서가 맞을 수 있도록 해준다. players는 플레이어의 수도 로드하여 할 수 있도록 해준다. 그 후 fclose()로 파일을 닫아주며 함수가 종료된다.

readSaveGame() 함수는 인자에 int* players 값을 추가하였다. player의 수도 저장하고 불러오는데 필요하였다. 처음에 찾을 파일이름을 입력받을 char 타입의 str 변수를 선언하였고, 키보드 입력이 보이도록 echo() 함수를 사용하였다. 또한 wgetstr()로 stdscr window에서 입력을 받았다. 또한 파일이름의 마지막의 개행문자를 for문을 통해서 제거하고 이 후 게임 시작 이후에는 키보드 입력이 보이지 않도록 noecho()를 사용하였다. 입력 받은 파일의 이름으로 "r" 모드로 열어준 후 HEIGHT, WIDTH를 파일에서 입력 받고, board에 HEIGHT * WIDTH 크기의 정수 배열을 동적할당하고, board 배열을 파일로부터 입력 받는다. 그리고 row, col, turn, players 값을 입력 받아 기존의 파일에서 게임을 불러왔을 때 필요한 데이터를 모두 불러온다.

<프로그램의 전체 흐름도>


start
Input
load
load
y
n
Input
file name
Input
HEIGHT, WIDTH, players
start window
gamestart()
initiating the board
True
painting the board


painting the MENU
Input
keyin
move the cursor
arrow key
Action()
enter or space
put the block and checkWin()
1
saveGame()
2
break
free board
end window
end

<프로그램에 대한 평가>

구현한 함수들 중에서 아쉬운 점을 가지고 있는 부분은 checkWin() 함수의 효율성에 대한 부분이 있다. checkWin() 함수를 구현하는 과정에 있어서 board를 전체 탐색하는 방식을 사용하여 구현하였다. 이러한 탐색 방식에서 time complexity는 O(HEIGHT * WIDTH)라 할 수 있으므로 HEIGHT와 WIDTH의 값이 커지게 되면 시간이 늘어나게 될 것이다. 또한 checkWin() 함수는 돌을 놓을 때마다 실행되는 함수이므로 자주 실행되기에 프로그램 전체의 효율에 큰 영향을 미치게 된다. 따라서 좀 더 탐색하는 범위를 줄일 수 있는 알고리즘을 사용한다면 더 빠른 속도로 구현되는 오목이 될 것이다. 또한 checkWin()은 돌을 놓을 때마다 실행하며 승리 조건을 판단하므로 승리 조건을 만족하는 경우는 해당 턴에 돌을 놓은 플레이어일 것이므로 누가 승리했는지 출력하는데 문제는 없으나 돌의 문자를 바탕으로 검증하는 것은 아니므로 해당 부분에 대한 아쉬움이 있다.

이 부분에 대해서 개선해볼 수 있는 점에 대해서는 플레이어가 놓은 위치와 돌의 문자를 인자로 받아서 해당 위치의 주변만 탐색하는 방식이 있을 것이다. 위에서 구현한 방식은 탐색하는 방향을 4가지로 했으나 이러한 방식으로 구현한다면 각각의 방향에서 완전 반대 방향 4가지를 추가로 탐색하면서 확인하는 방법이 있을 것이다.

두 번째 아쉬운 점은 게임의 내용을 저장하는 방식에 대한 것이다. 저장에 대한 구현은 필요한 변수들에 더하여 배열 전체를 저장하는 방식을 사용하였다. 이 방식에 대해서 돌이 많이 놓여있지 않을 때 저장할 때는 비효율적일 것이다. 위에서 구현한 방식에서는 애초에 initBoard() 함수 안에서 실행하도록 하여 배열 전체를 받아오도록 했으나 좀 더 효율적으로 저장하려면 initBoard()가 실행된 후 실행되도록 하는 방식도 있을 것이다. 저장에 대해서는 sparse matrix를 저장하는 방식처럼 돌이 있는 row, col과 돌의 문자만 저장하는 방식으로 저장하면 더 적은 메모리로 저장할 수 있을 것이다.

이외에는 구현하고자 하는 바를 구현하는데 문제 없이 작동하고 필요한 부분도 구현되어 있다.