Data Structures

Assignment #1 Report

Subject No.:CSE3080-02

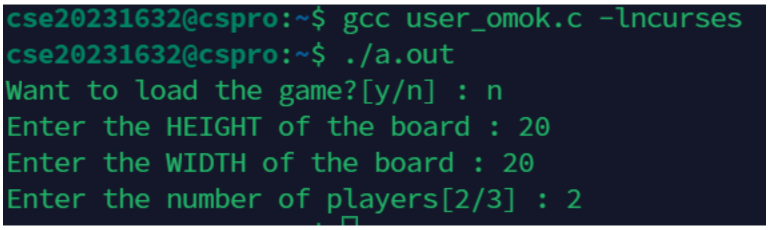
Name: Jumagul Alua

Student ID: 20231632

(I) Program explanation

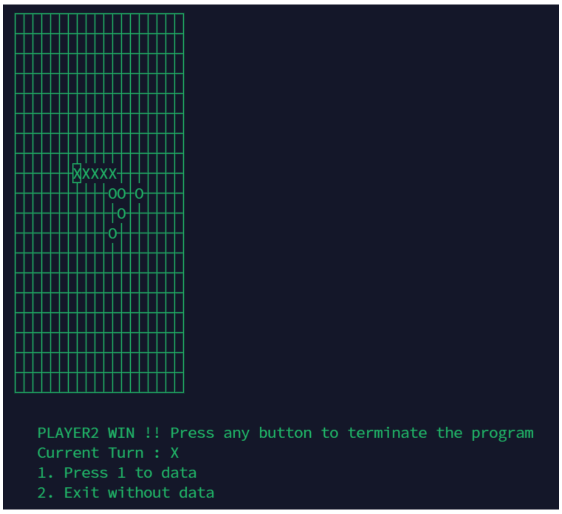
The program implements Omok game using the ‘ncurses’ library in Unix. The program used the C language compiled with ‘gcc’ in a result we got executable ‘./a.out’. After, the program will get input from a user whether we if we want to load a saved game, size of the board and number of players.

User Input:



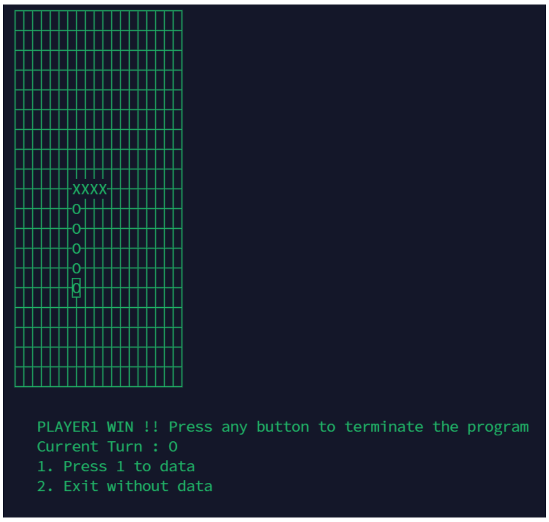
```
cse20231632@cspro:~$ gcc user_omok.c -lncurses
cse20231632@cspro:~$ ./a.out
Want to load the game?[y/n] : n
Enter the HEIGHT of the board : 20
Enter the WIDTH of the board : 20
Enter the number of players[2/3] : 2
```

(II) Ouput of the program:



```
PLAYER2 WIN !! Press any button to terminate the program
Current Turn : X
1. Press 1 to data
2. Exit without data
```

(a)



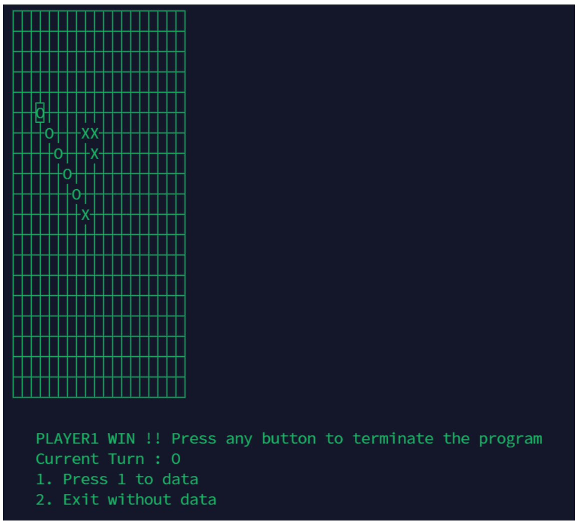
```
PLAYER1 WIN !! Press any button to terminate the program
Current Turn : O
1. Press 1 to data
2. Exit without data
```

(b)

When there are 2 players,

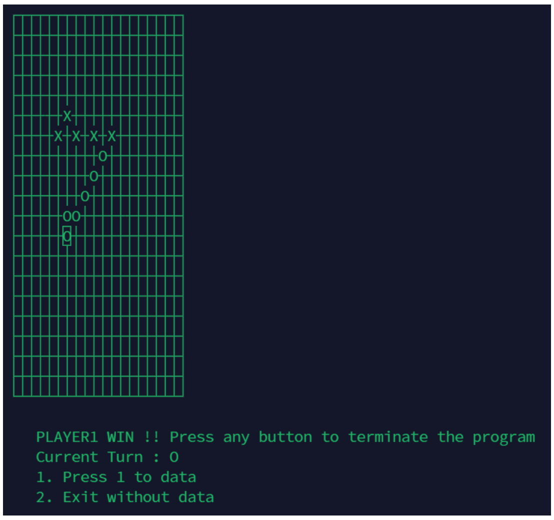
(a) if user's five stones are positioned vertically in a column, it outputs that the user has won.

(b) if user's five stones are positioned horizontally in a row, it outputs that the user has won.



```
PLAYER1 WIN !! Press any button to terminate the program
Current Turn : O
1. Press 1 to data
2. Exit without data
```

(c)



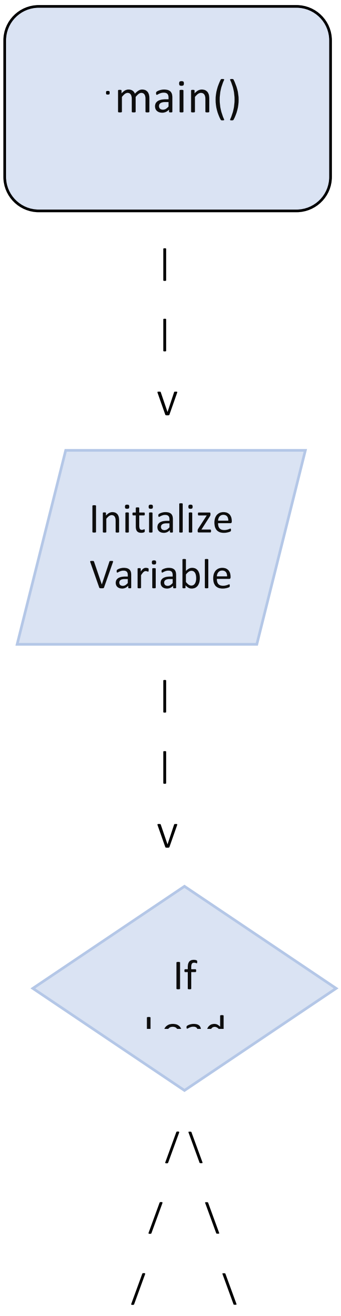
```
PLAYER1 WIN !! Press any button to terminate the program
Current Turn : O
1. Press 1 to data
2. Exit without data
```

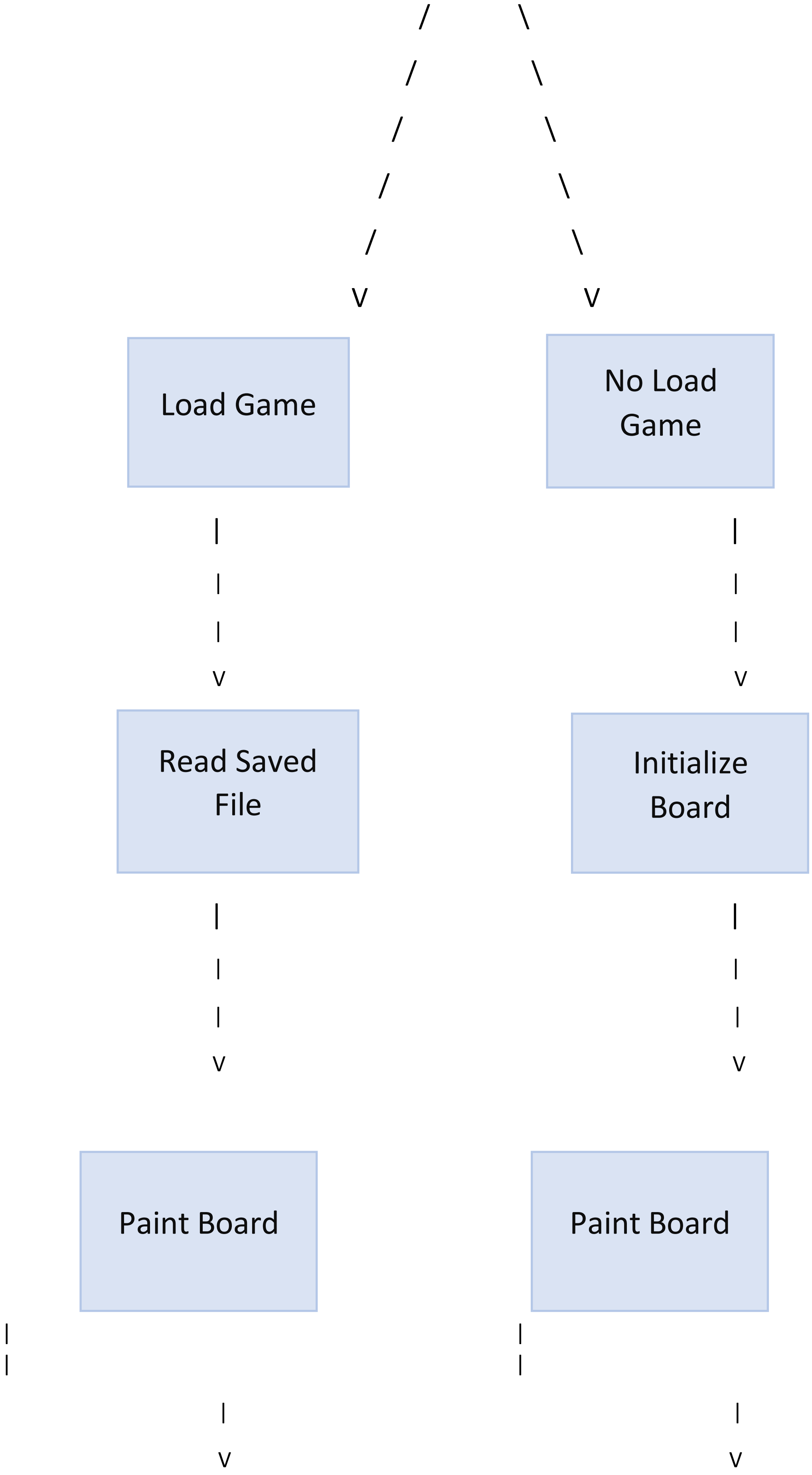
(d)

(c) if user's five stones are positioned diagonally from left-top to right-bottom, it outputs that the user has won.

(d) if user's five stones are positioned diagonally from right-top to left-bottom, it outputs that the user has won.

(III) Overall Flow Chart





Load Game

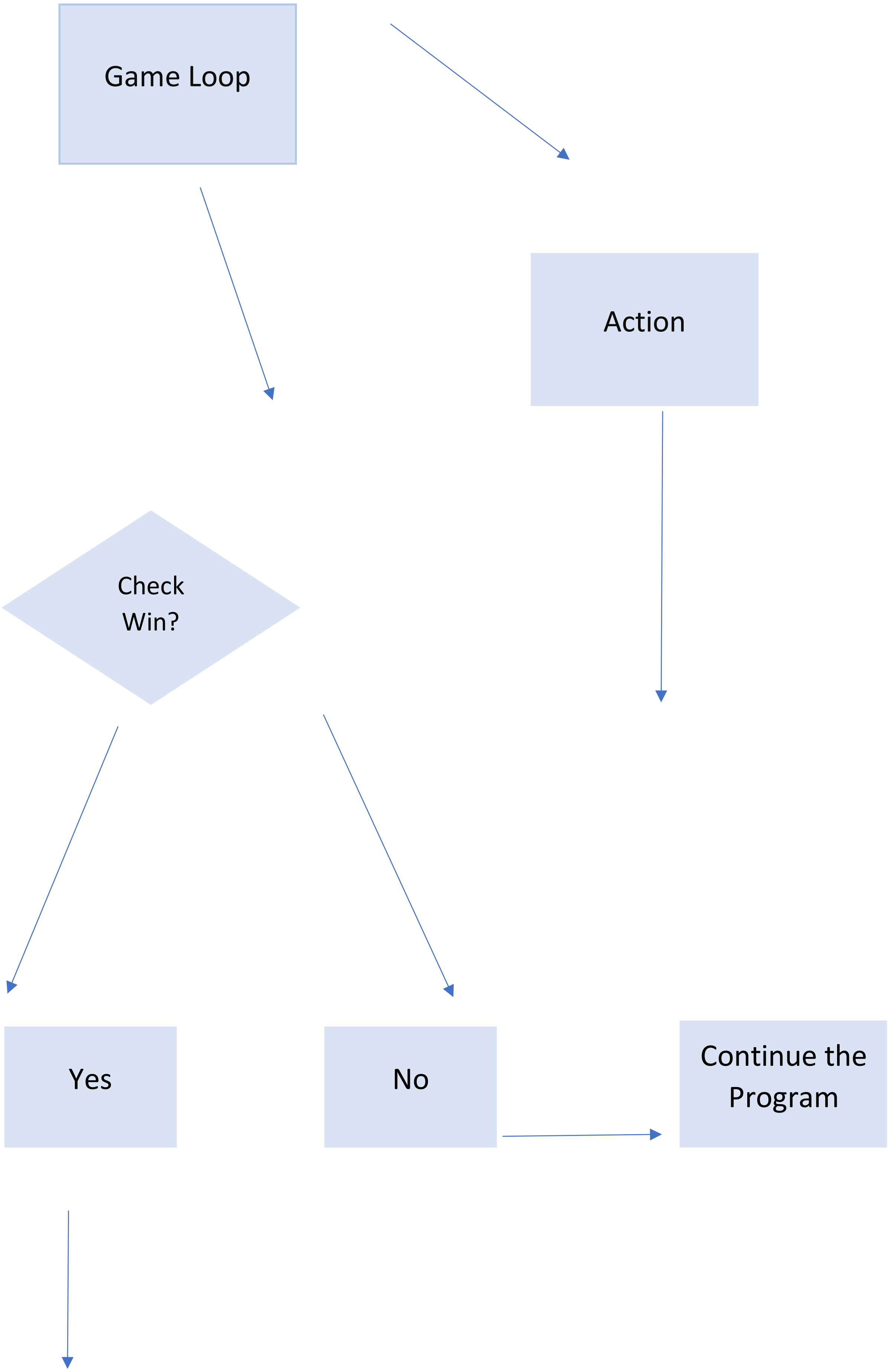
No Load Game

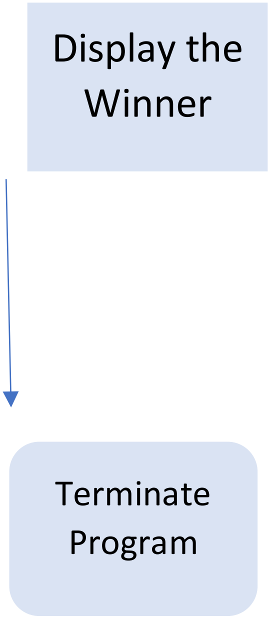
Read Saved File

Initialize Board

Paint Board

Paint Board


Game Loop
Action
Check Win?
Yes
No
Continue the Program



(1) The program begins execution.

(2) Asks the user if they want to load a saved game.

If yes, proceed to load the game.

If no, proceed to ask for board size and number of players.

(3) Initialize the game board, including its size and players.

(4) Ask the current player for their action.

If the action is to save the game, save it and exit.

If the action is to exit without saving, exit.

Otherwise, update the game board according to the action.

(5) Determine if any player has won.

If yes, display the winner and end the game.

If no, continue to the next turn.

(6) The game ends, and the program exits.

<u>Explanation to each function</u>

- **MainFunction**: It prompts the user to choose whether to load a saved game or start a new game, initializes the ncurses environment, and calls the `gameStart` function to start the game.

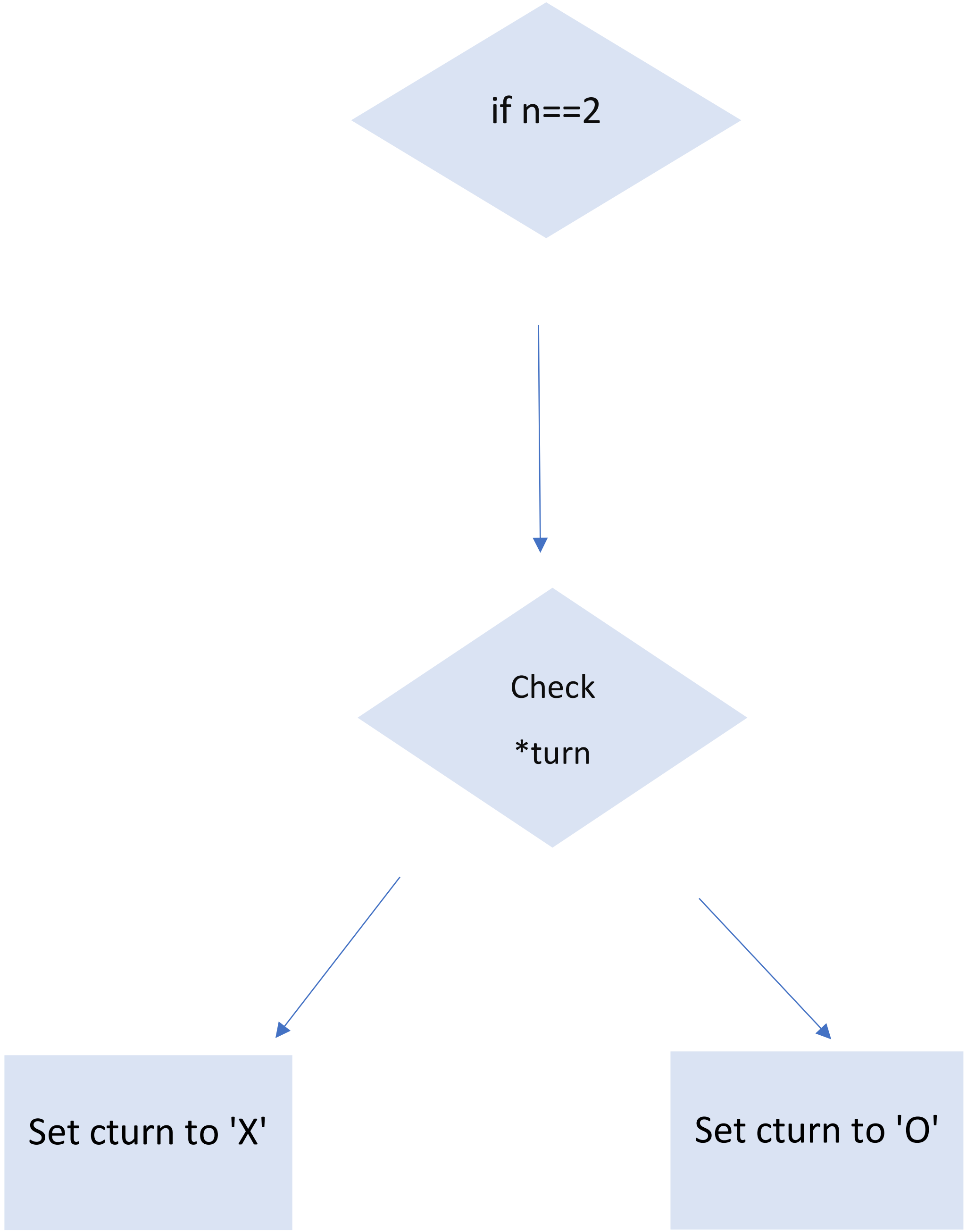
- **InitBoardFunction**: This function is responsible for initializing a new game board.
- **ReadSavedGameFunction**: This function is called when the user chooses to load a saved game.
- **SaveGameFunction**: This function is called when the user chooses to save the current game state.
- **PaintBoardFunction**: This function is responsible for updating the game board display in the ncurses window.
- **CheckWinFunction**: This function checks if the current player has won the game.
- **ActionFunction**: This function handles the user's input and updates the game state accordingly.

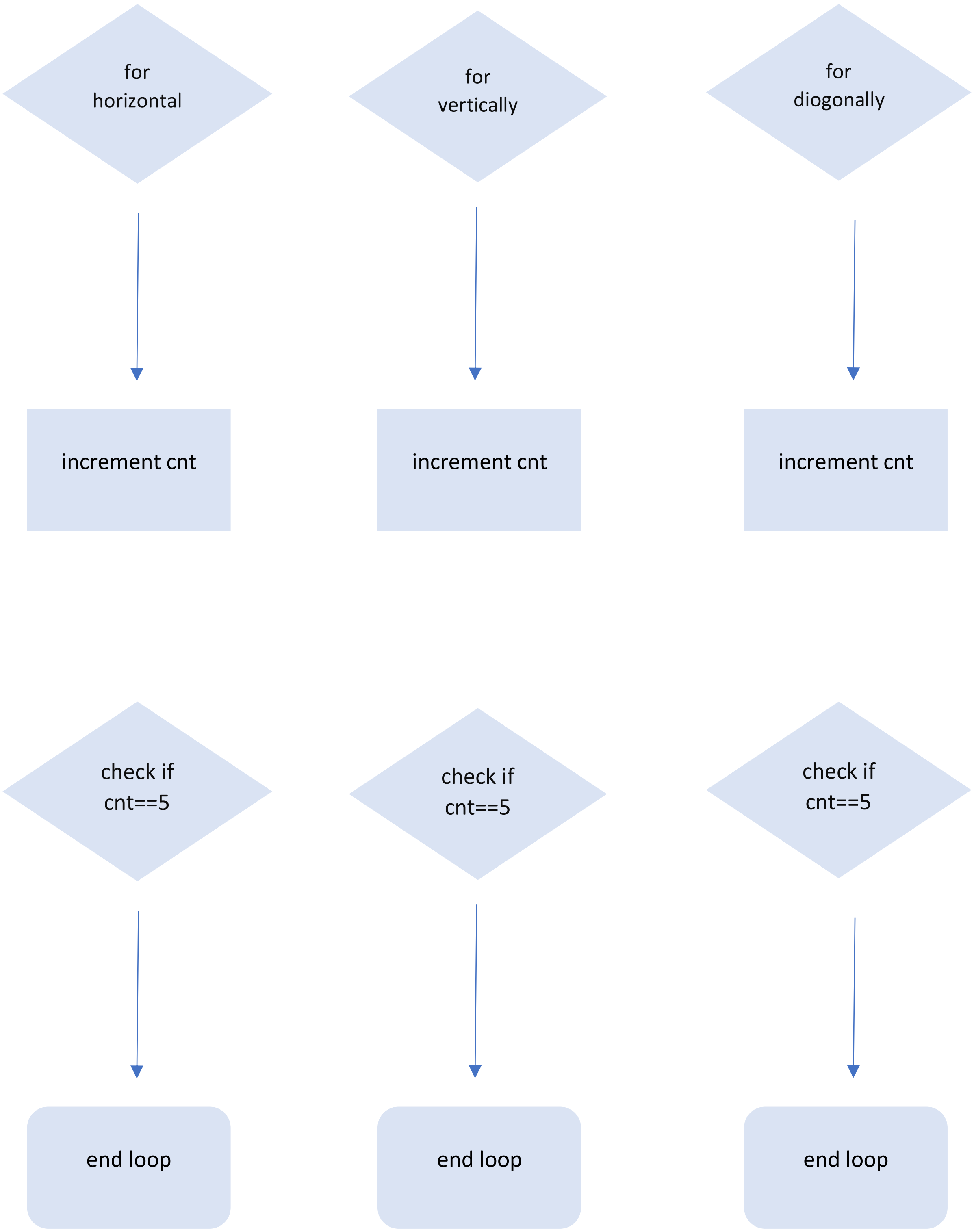
- **GameStartFunction**: This function is the main entry point for the game.

(III) CheckIWin function

```
Int CheckWin (int **board, int *row, int *col, int *turn, int n, int temp)
```

This function checks the game board to determine if a player has won the game.





In this function, we have two options such as whether 2 players or 3 players. And for each option there are 4 for loops.

- For 2 players,

1) The first one is for checking horizontally win, loop goes through columns in the range (*col - 4) to *col and after counting cturn marks, if the count reaches to 5, return 1.

2) For checking vertically win, loop goes through rows in the range (*row - 4) to *row.

3) For checking ascending diagonal win, loop goes through offsets i in the range -4 to 0

4) For checking ascending diagonal win, loop goes through offsets i in the range -4 to 0

- And for 3 players, repeat similar win checking logic, but 4 loop ranges instead of 5.
- If no win condition is met, return 0 (no win)

(V) Action function

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

```
                    --------------------------

                    |        Action()        |

                    --------------------------

                                 |

                                 V

                          Get key input

                                 |

                                 V

                         Determine player

                                 |

                                 V

                            Switch (n)

        --------------                    ---------------

        |   n==2   |                      |    n==3    |

        --------------                   ----------------

              |                                 |

              V                                 V
```

```
                                        Handle actions                    Handle actions
for 2 players                            for 3 players

                                              |                                 |

                                              V                                 V

                                      Check win condition              Check win condition
```

- This function, handles the player's actions in the game based on the input received from the user.