CHESS

Programming in C

Sehaj Saluja	21070122152
Shubham Sadukha	21070122160
Sujay Kumar	21070122167
Swayam Pendgaonkar	21070122193

CONTENT

- Problem Statement
- Objective and Motivation
- Flowchart
- Source Code
- Output
- Future Scopes
- Conclusion

PROBLEM STATEMENT

Building a project to demonstrate skills learned in programming in C. We tried to make a program which can be fun to develop as well as use. Also. we wanted to try something different than onrthodox projects like management systems.

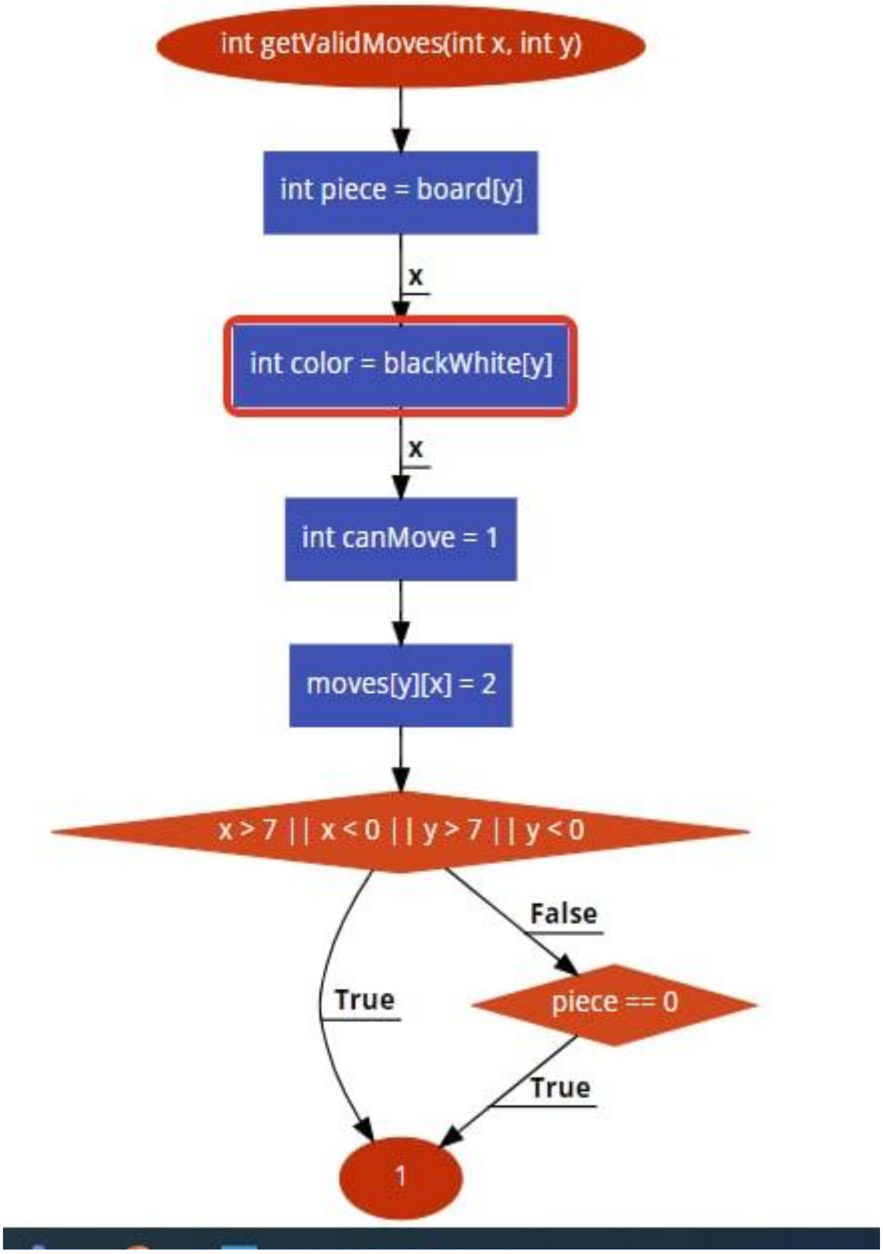
OBJECTIVE

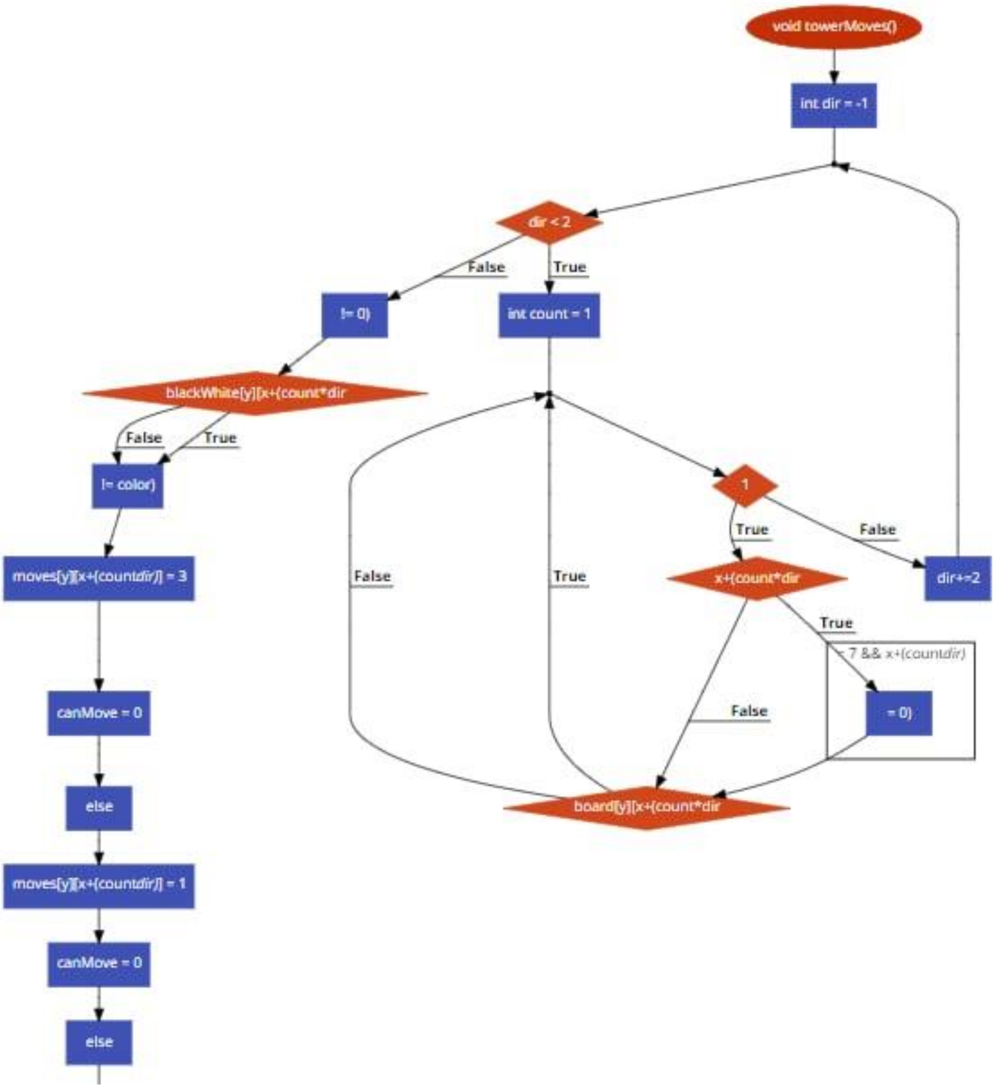
- Design a project using C programming language
- Manipulate different types of data using appropriate data structures, functions etc.
- Practice analysing and debugging techniques.
- Develop good coding skills.

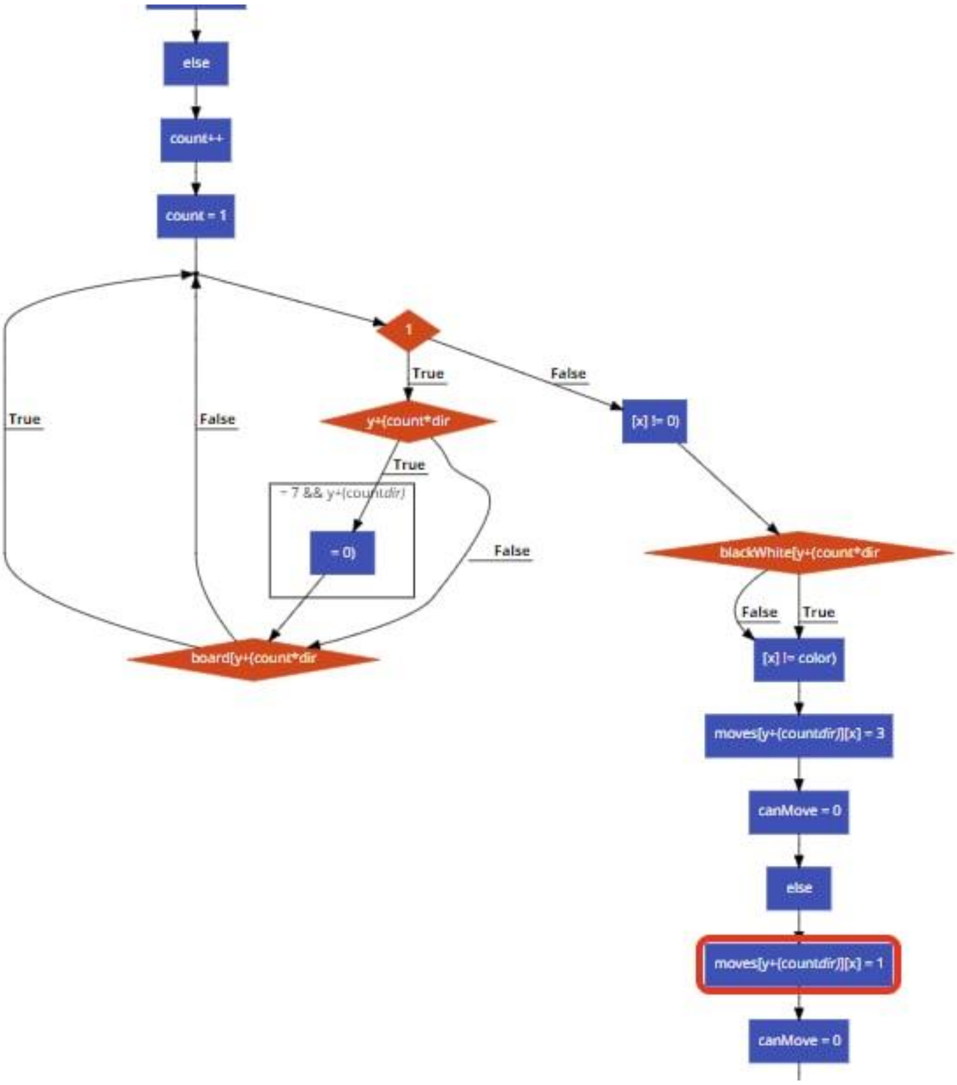
MOTIVATION

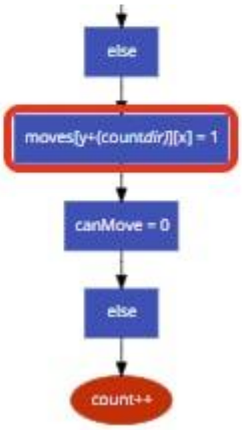
- To make a chess game which is fun and interesting to play.
- Make a classic game which will help us to learn fundamentals of game development.
- To make a project which can be fun to develop as well as use.
- To have users have fun playing the game with friends.

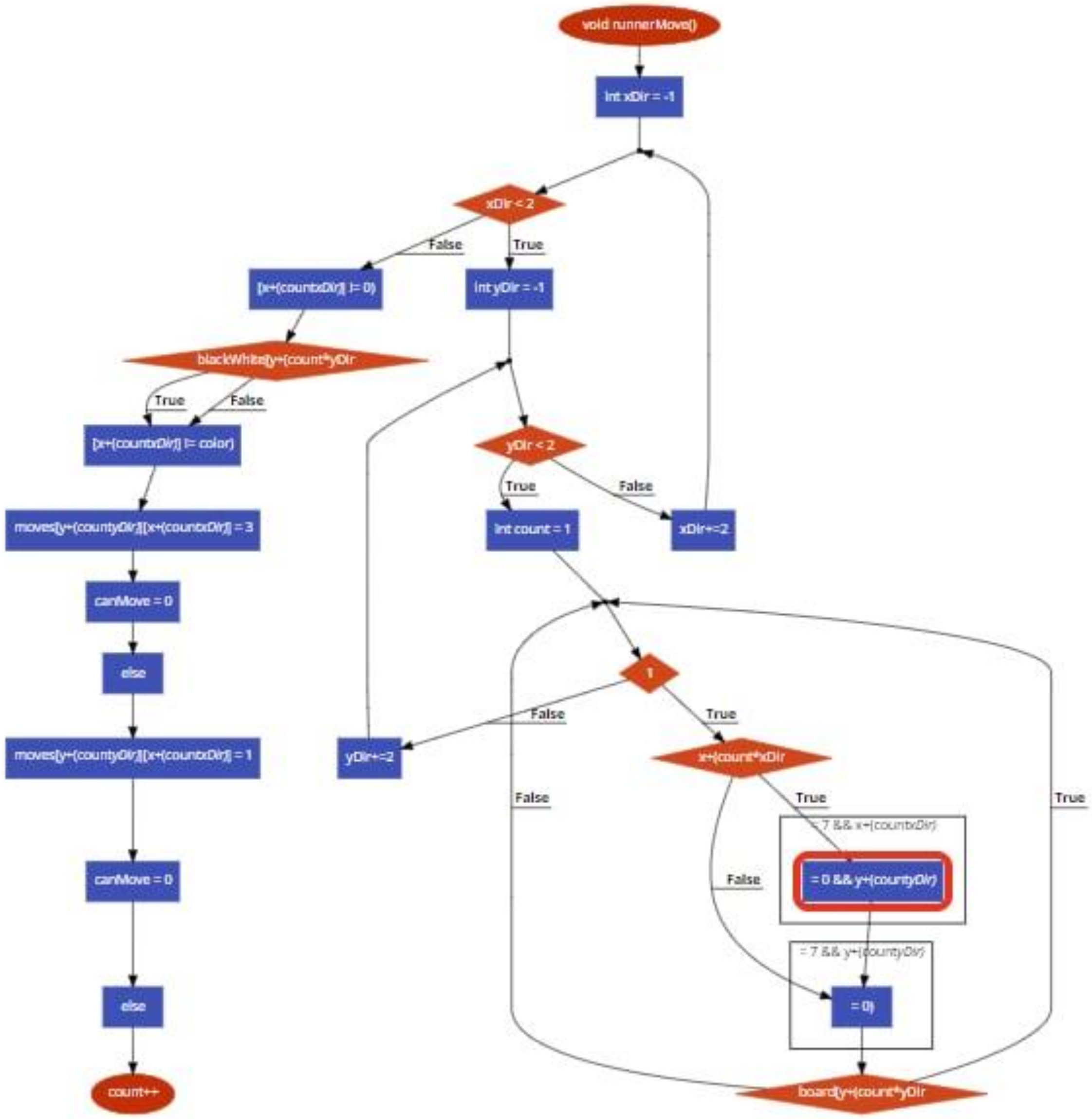
Flowcharts

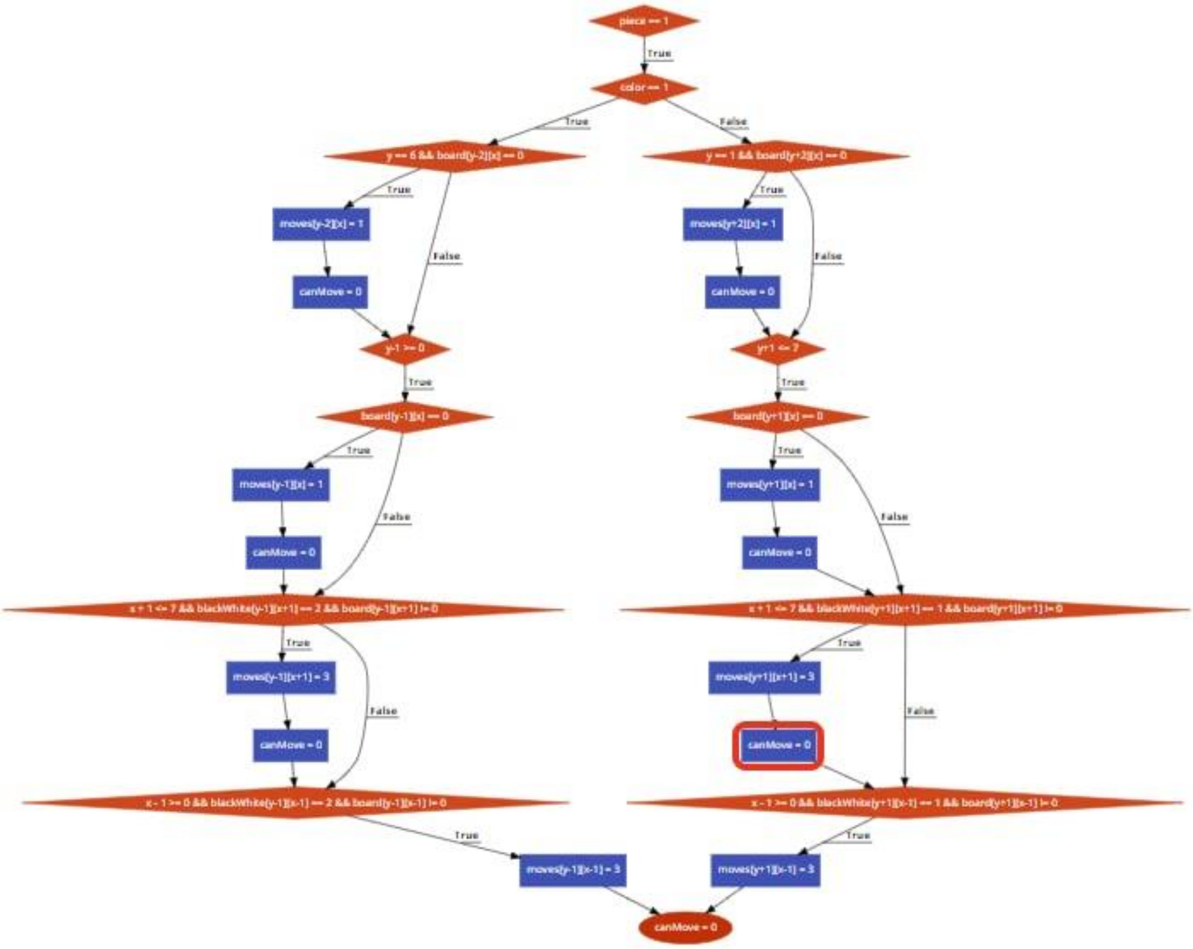

int getValidMoves(int x, int y)
int piece = board[y]
x
int color = blackWhite[y]
x
int canMove = 1
moves[y][x] = 2
x > 7 || x < 0 || y > 7 || y < 0
False
True
piece == 0
True
1

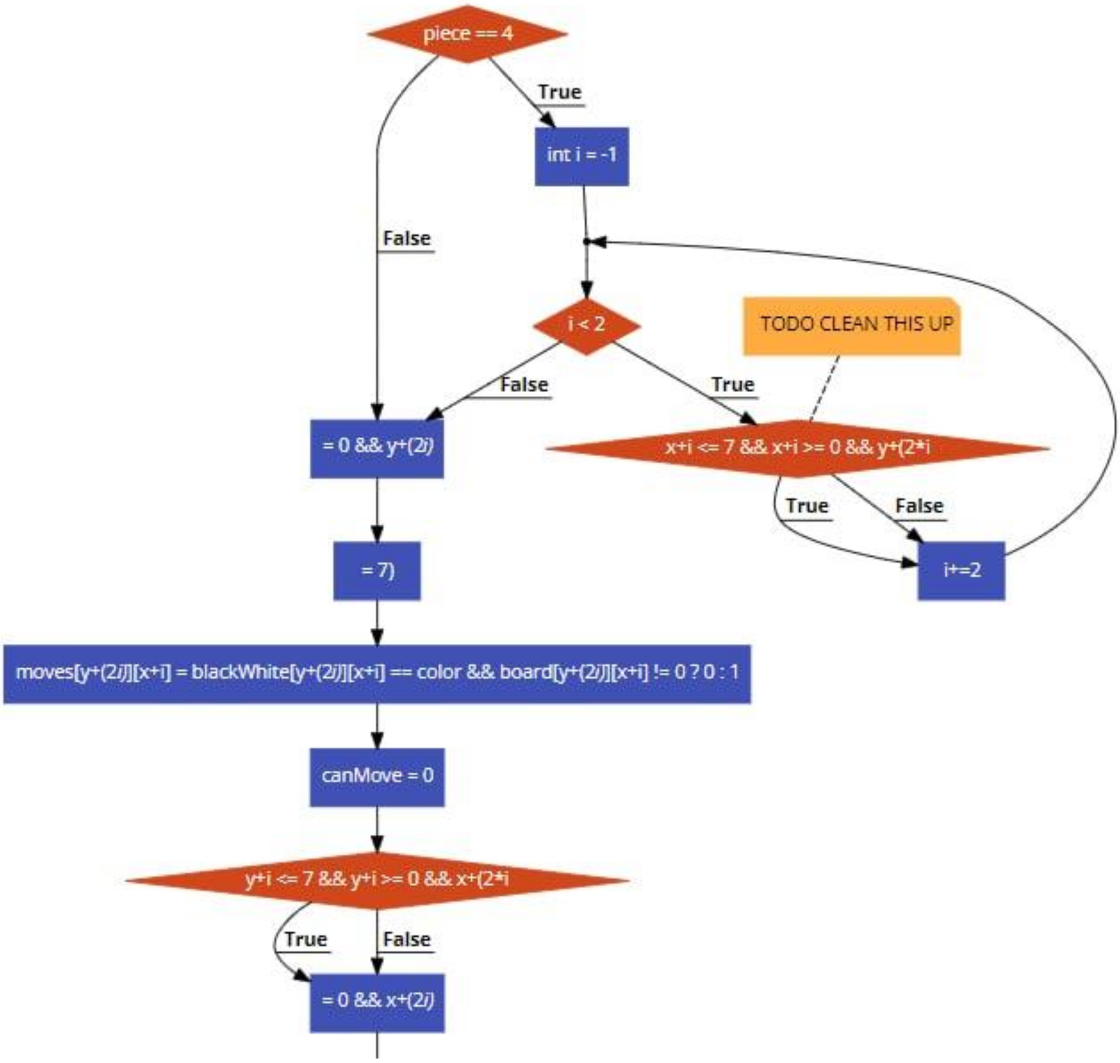

void towerMoves()
int dir = -1
dir < 2
False
True
!= 0)
int count = 1
blackWhite[y][x+(count*dir
False
True
!= color)
moves[y][x+(count*dir*)] = 3*
canMove = 0
else
moves[y][x+(count*dir*)] = 1*
canMove = 0
else
1
True
False
dir+=2
x+(count*dir
True
= 7 && x+(count*dir*)*
= 0)
False
False
True
board[y][x+(count*dir

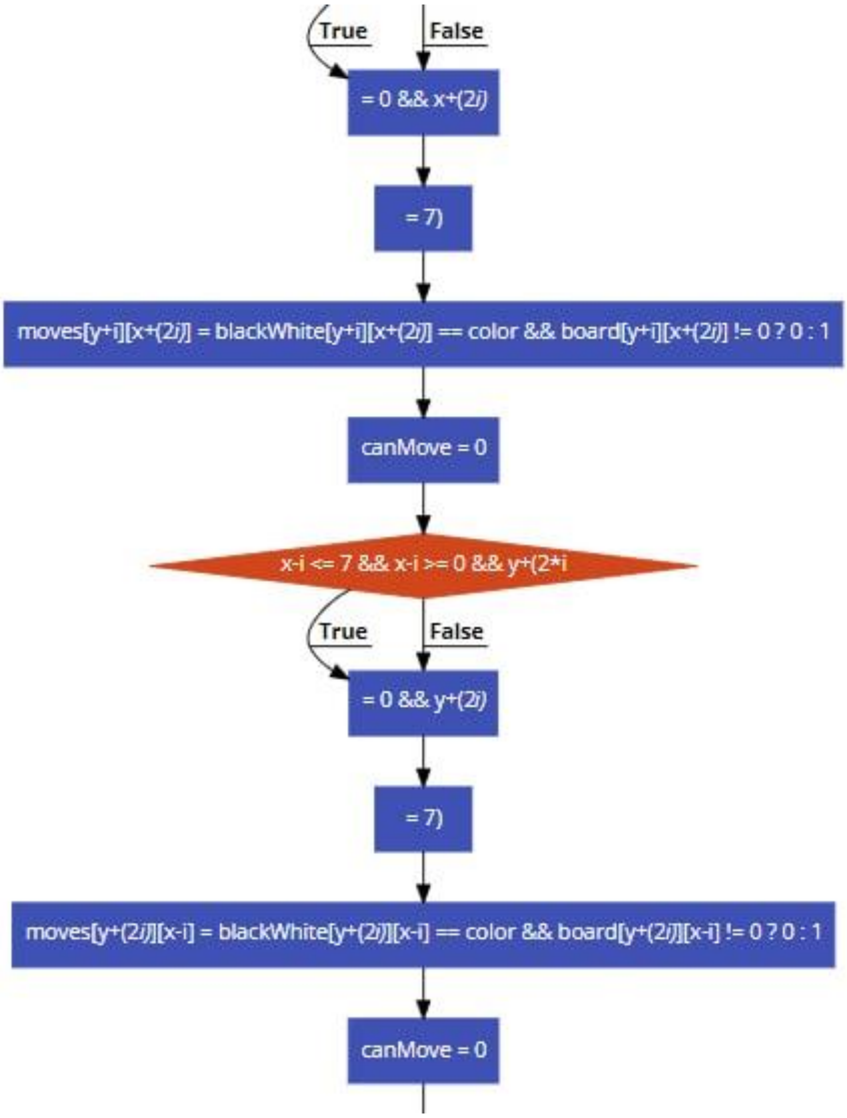

else
count++
count = 1
1
True
False
y+(count*dir
True
~ 7 && y+(countdir)
= 0)
False
True
False
board[y+(count*dir
[x] != 0)
blackWhite[y+(count*dir
False
True
[x] != color)
moves[y+(countdir)][x] = 3
canMove = 0
else
moves[y+(countdir)][x] = 1
canMove = 0

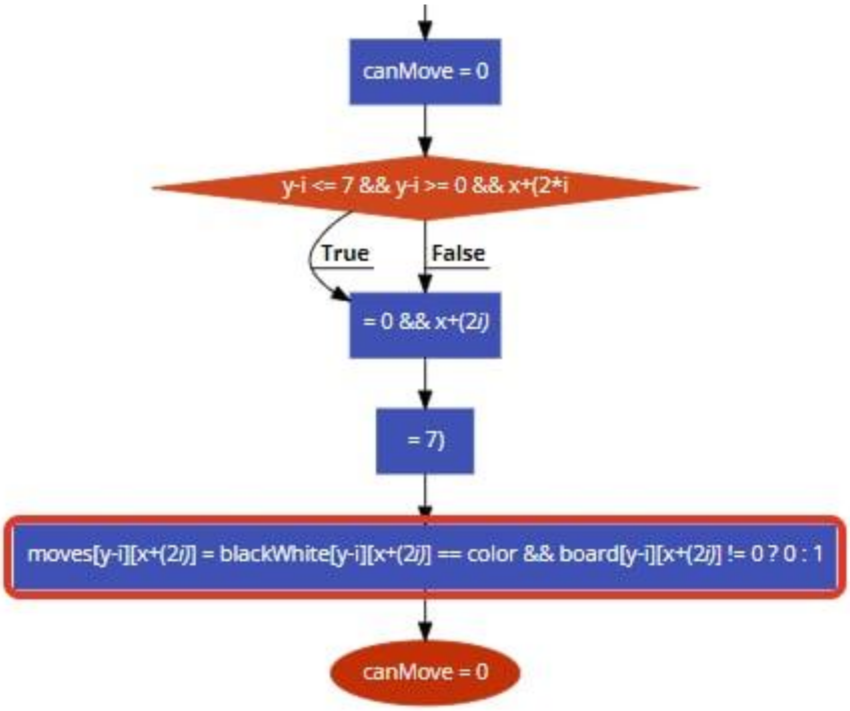

else
moves[y+(countdir)][x] = 1
canMove = 0
else
count++

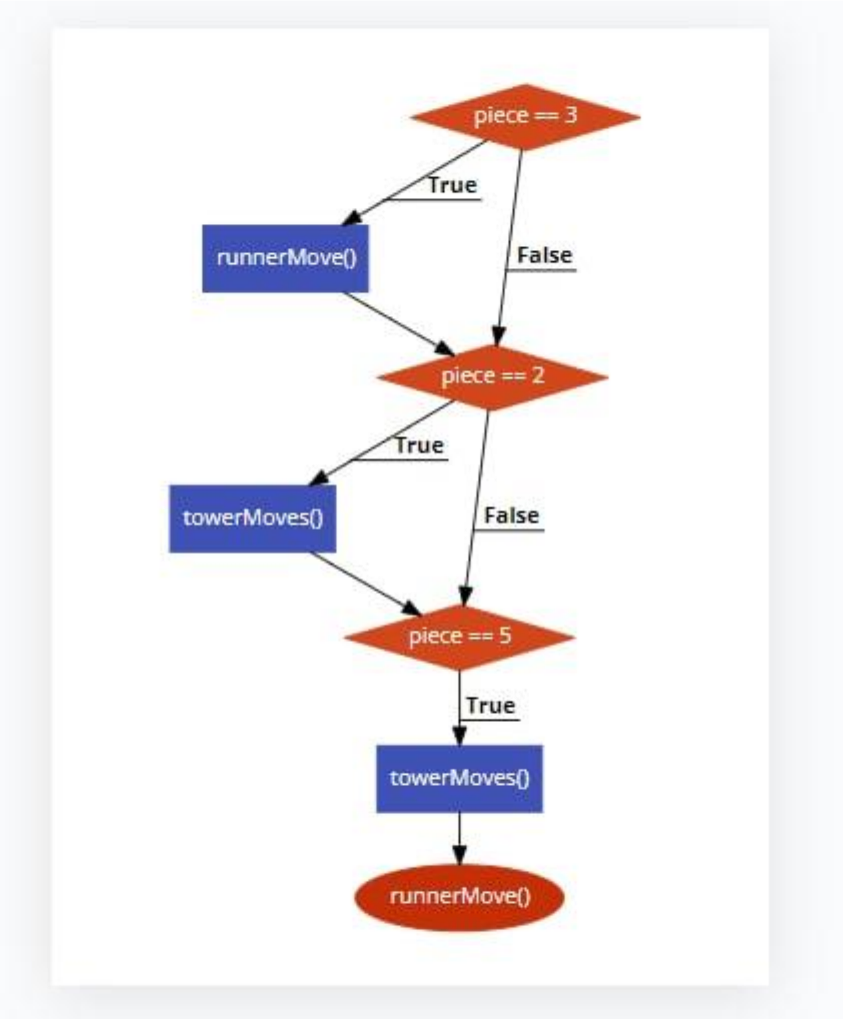

void runnerMove()
int xDir = -1
xDir < 2
False
True
[x+(countxDir)] != 0)
int yDir = -1
blackWhite[y+(count*yDir
True
False
[x+(countxDir)] != color)
yDir < 2
True
False
int count = 1
xDir+=2
moves[y+(countyDir)][x+(countxDir)] = 3
canMove = 0
else
1
False
True
yDir+=2
x+(count*xDir
moves[y+(countyDir)][x+(countxDir)] = 1
False
True
True
= 7 && x+(countxDir)
False
= 0 && y+(countyDir)
canMove = 0
= 7 && y+(countyDir)
else
= 0)
count++
board[y+(count*yDir

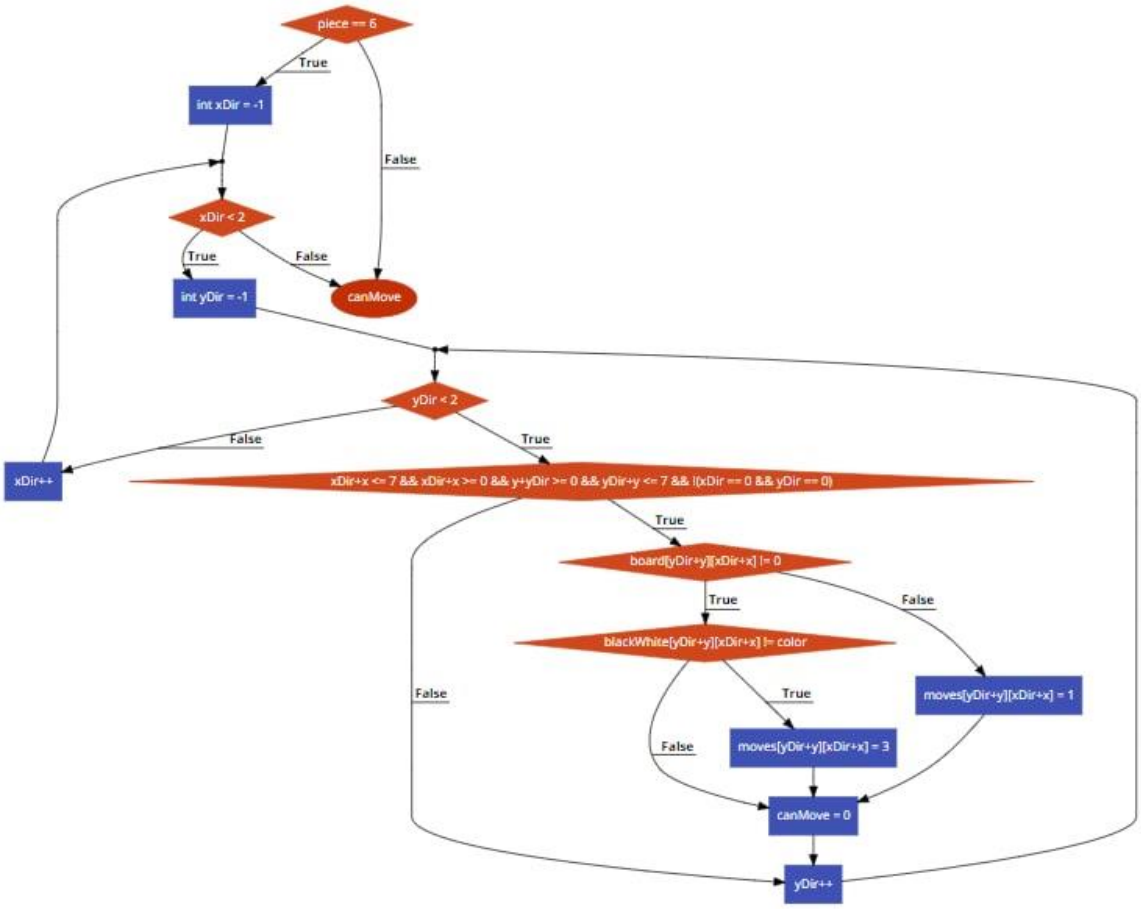

piece == 1
True
color == 1
True
False
y == 6 && board[y-2][x] == 0
y == 1 && board[y+2][x] == 0
True
moves[y-2][x] = 1
False
True
moves[y+2][x] = 1
False
canMove = 0
canMove = 0
y-1 >= 0
y+1 <= 7
True
True
board[y-1][x] == 0
board[y+1][x] == 0
True
True
moves[y-1][x] = 1
moves[y+1][x] = 1
False
False
canMove = 0
canMove = 0
x + 1 <= 7 && blackWhite[y-1][x+1] == 2 && board[y-1][x+1] != 0
x + 1 <= 7 && blackWhite[y+1][x+1] == 1 && board[y+1][x+1] != 0
True
True
moves[y-1][x+1] = 3
moves[y+1][x+1] = 3
False
False
canMove = 0
canMove = 0
x - 1 >= 0 && blackWhite[y-1][x-1] == 2 && board[y-1][x-1] != 0
x - 1 >= 0 && blackWhite[y+1][x-1] == 1 && board[y+1][x-1] != 0
True
True
moves[y-1][x-1] = 3
moves[y+1][x-1] = 3
canMove = 0

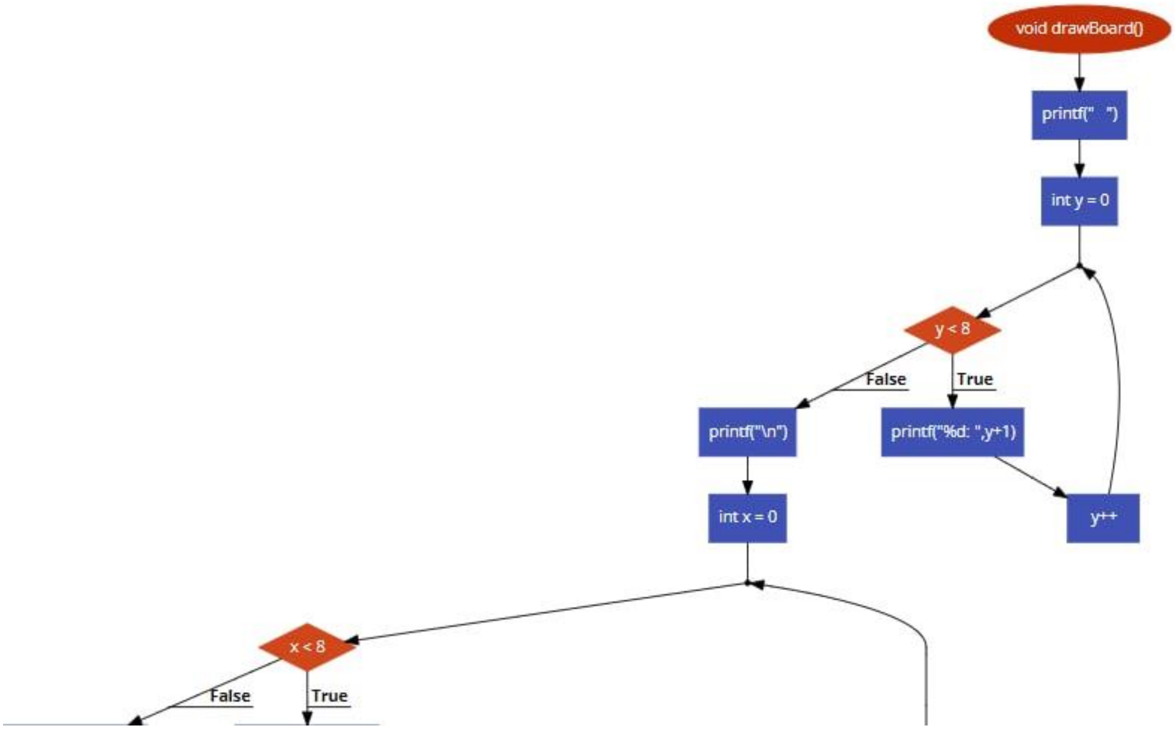

piece == 4
True
False
int i = -1
i < 2
TODO CLEAN THIS UP
False
True
= 0 && y+(2i)
x+i <= 7 && x+i >= 0 && y+(2*i
True
False
i+=2
= 7)
moves[y+(2i)][x+i] = blackWhite[y+(2i)][x+i] == color && board[y+(2i)][x+i] != 0 ? 0 : 1
canMove = 0
y+i <= 7 && y+i >= 0 && x+(2*i
True
False
= 0 && x+(2i)

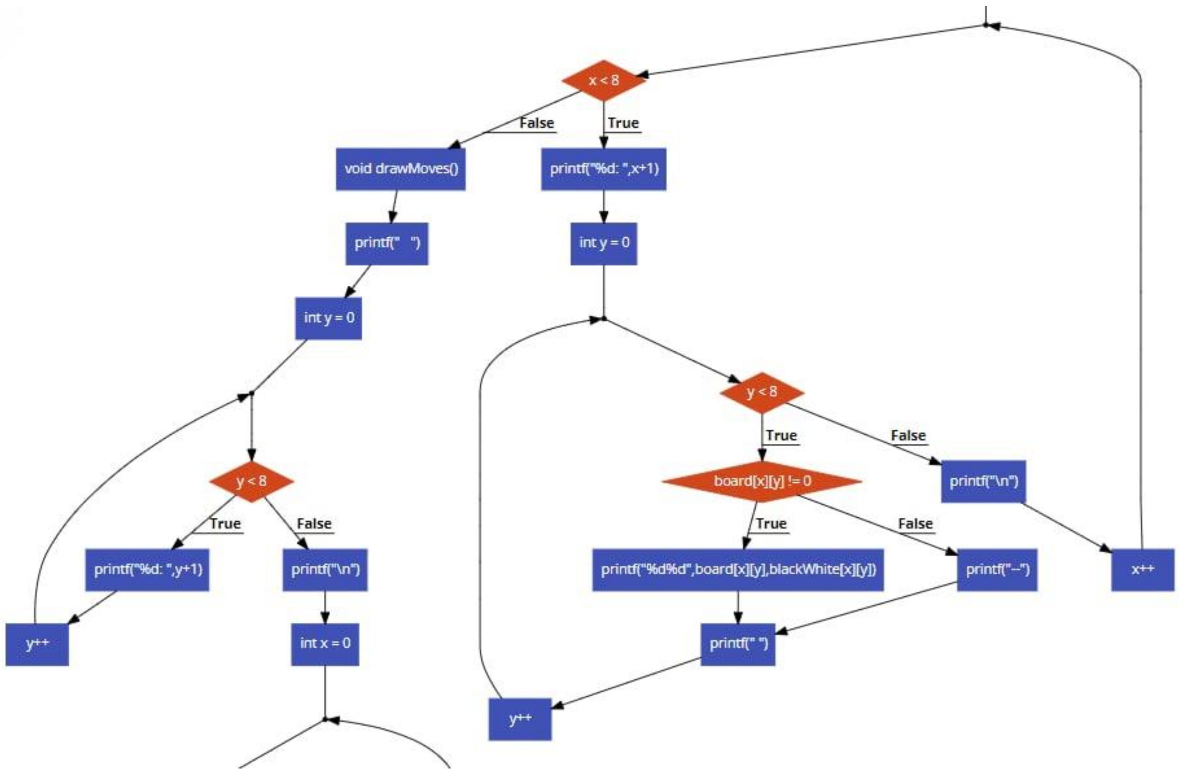

True
False
= 0 && x+(2i)
= 7)
moves[y+i][x+(2i)] = blackWhite[y+i][x+(2i)] == color && board[y+i][x+(2i)] != 0 ? 0 : 1
canMove = 0
x-i <= 7 && x-i >= 0 && y+(2*i
True
False
= 0 && y+(2i)
= 7)
moves[y+(2i)][x-i] = blackWhite[y+(2i)][x-i] == color && board[y+(2i)][x-i] != 0 ? 0 : 1
canMove = 0

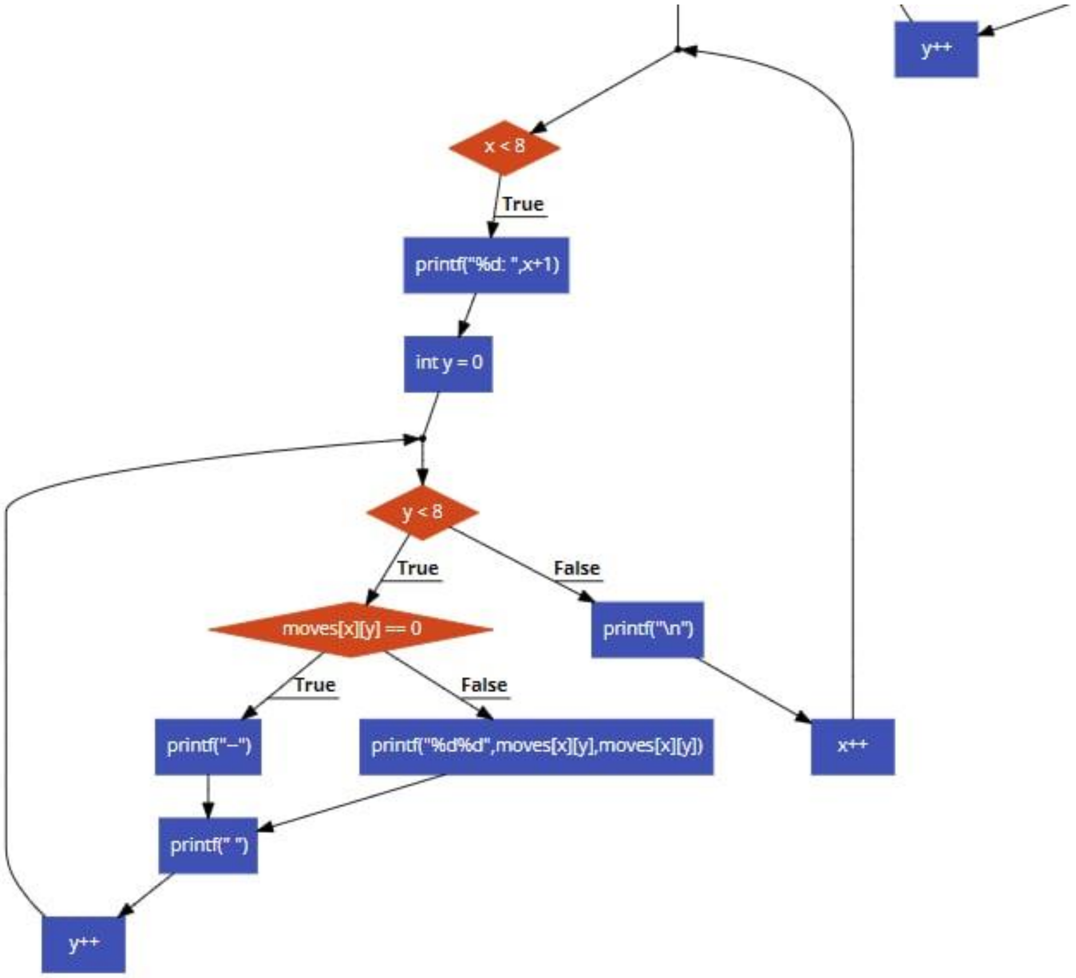

canMove = 0
y-i <= 7 && y-i >= 0 && x+(2*i
True
False
= 0 && x+(2*i*)
= 7)
moves[y-i][x+(2*i*)] = blackWhite[y-i][x+(2*i*)] == color && board[y-i][x+(2*i*)] != 0 ? 0 : 1
canMove = 0

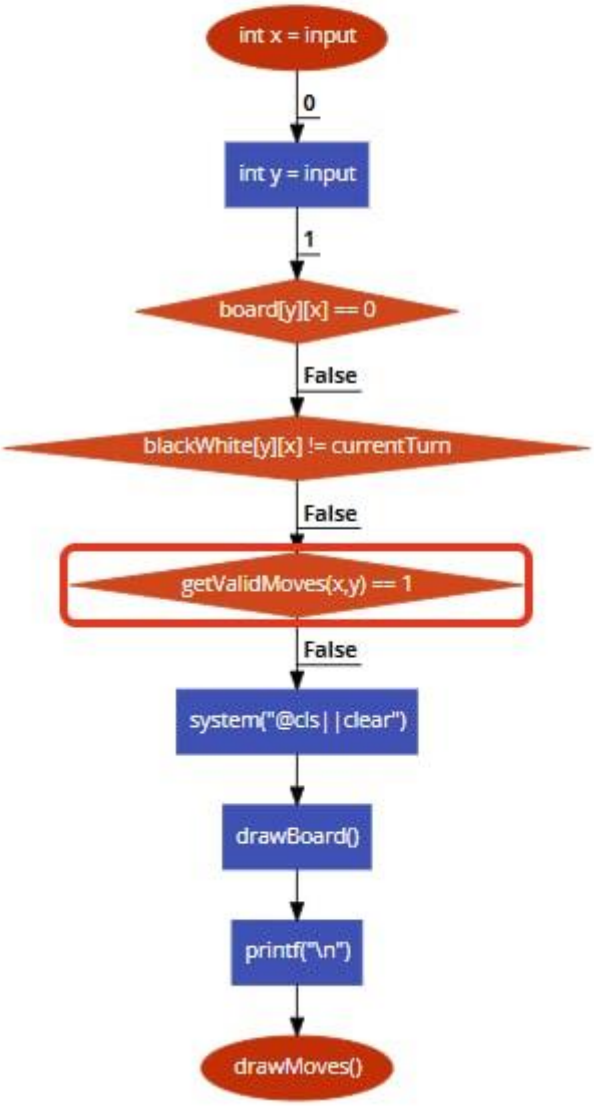

piece == 3
True
False
runnerMove()
piece == 2
True
False
towerMoves()
piece == 5
True
towerMoves()
runnerMove()


piece == 6
True
int xDir = -1
False
xDir < 2
True
False
int yDir = -1
canMove
yDir < 2
False
True
xDir++
xDir+x <= 7 && xDir+x >= 0 && y+yDir >= 0 && yDir+y <= 7 && !(xDir == 0 && yDir == 0)
True
board[yDir+y][xDir+x] != 0
True
False
blackWhite[yDir+y][xDir+x] != color
False
True
moves[yDir+y][xDir+x] = 1
False
moves[yDir+y][xDir+x] = 3
canMove = 0
yDir++

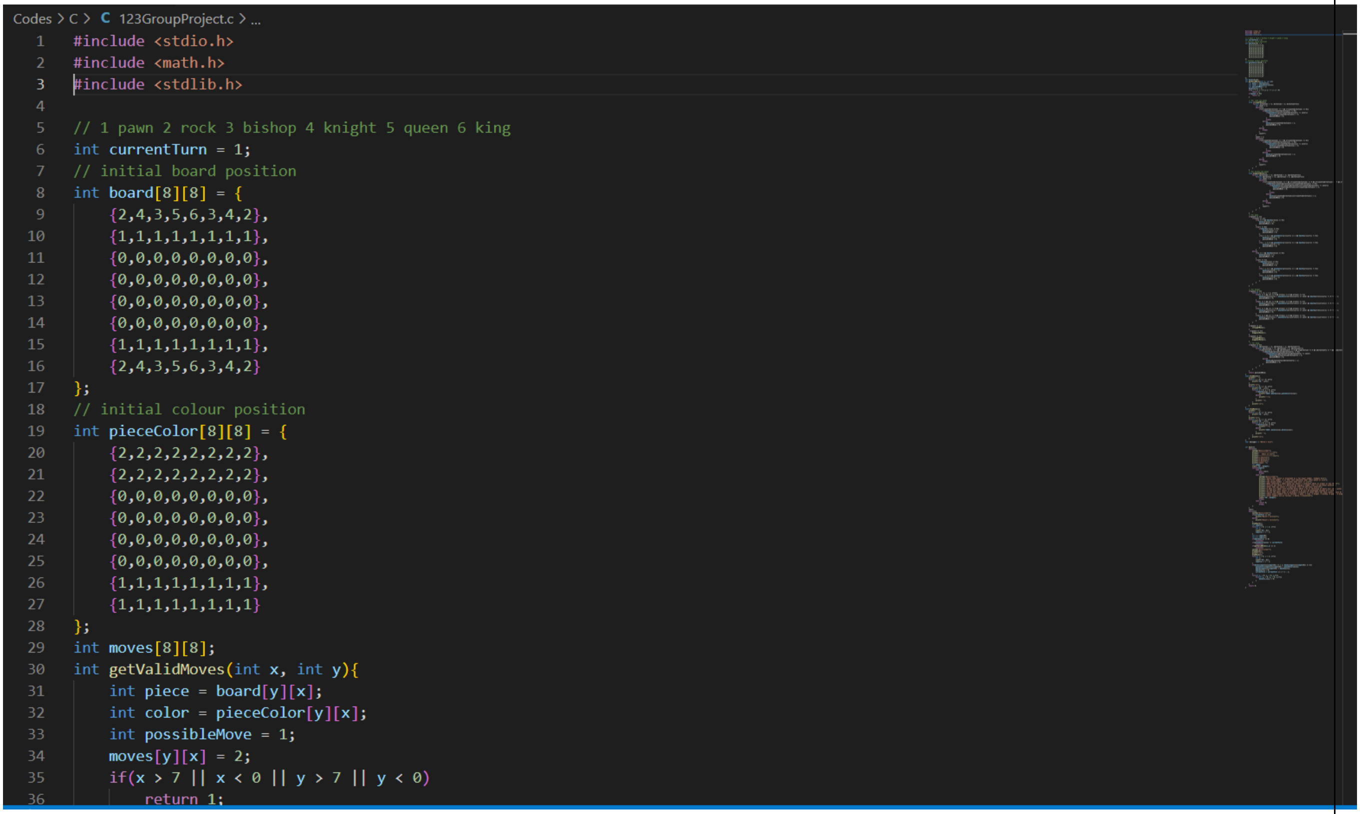

void drawBoard()
printf("   ")
int y = 0
y < 8
False
True
printf("\n")
printf("%d: ",y+1)
int x = 0
y++
x < 8
False
True


x < 8
False
True
void drawMoves()
printf("%d: ",x+1)
printf("   ")
int y = 0
int y = 0
y < 8
True
False
y < 8
True
False
board[x][y] != 0
printf("\n")
True
False
printf("%d: ",y+1)
printf("\n")
printf("%d%d",board[x][y],blackWhite[x][y])
printf("--")
x++
y++
int x = 0
printf(" ")
y++


y++
x < 8
True
printf("%d: ",x+1)
int y = 0
y < 8
True
False
moves[x][y] == 0
printf("\n")
True
False
printf("--")
printf("%d%d",moves[x][y],moves[x][y])
x++
printf(" ")
y++

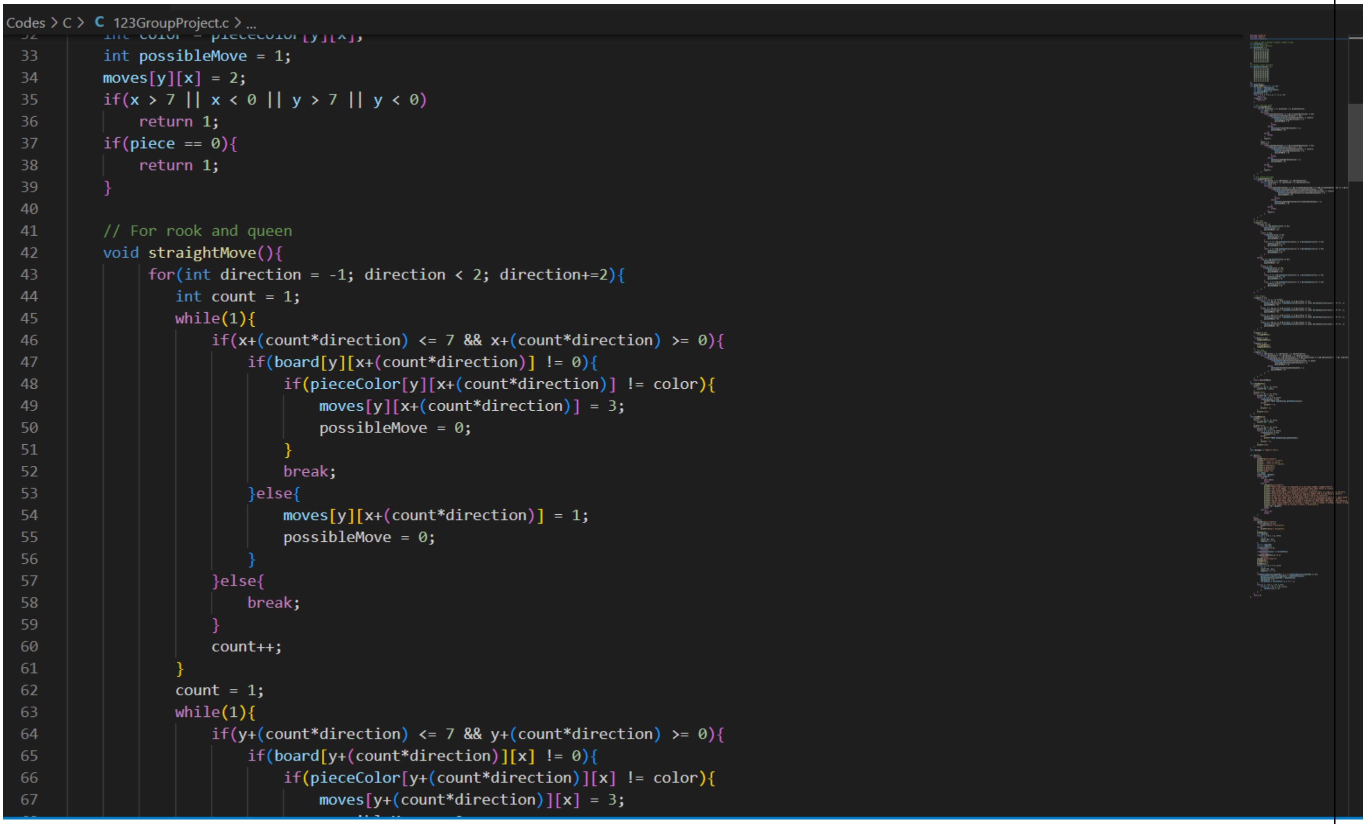

int x = input
0
int y = input
1
board[y][x] == 0
False
blackWhite[y][x] != currentTurn
False
getValidMoves(x,y) == 1
False
system("@cls||clear")
drawBoard()
printf("\n")
drawMoves()

Code



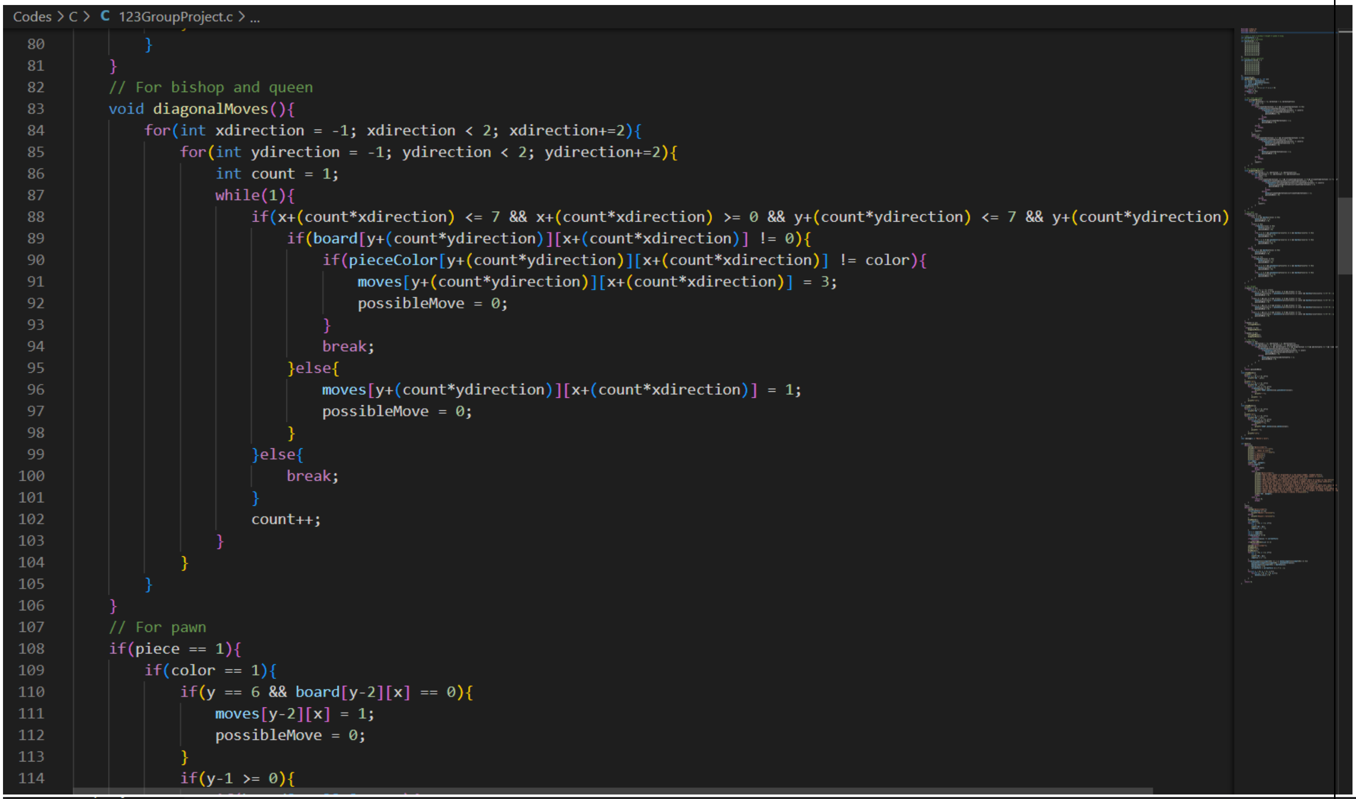
```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

// 1 pawn 2 rock 3 bishop 4 knight 5 queen 6 king
int currentTurn = 1;
// initial board position
int board[8][8] = {
    {2,4,3,5,6,3,4,2},
    {1,1,1,1,1,1,1,1},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {1,1,1,1,1,1,1,1},
    {2,4,3,5,6,3,4,2}
};
// initial colour position
int pieceColor[8][8] = {
    {2,2,2,2,2,2,2,2},
    {2,2,2,2,2,2,2,2},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1}
};
int moves[8][8];
int getValidMoves(int x, int y){
    int piece = board[y][x];
    int color = pieceColor[y][x];
    int possibleMove = 1;
    moves[y][x] = 2;
    if(x > 7 || x < 0 || y > 7 || y < 0)
        return 1;
```



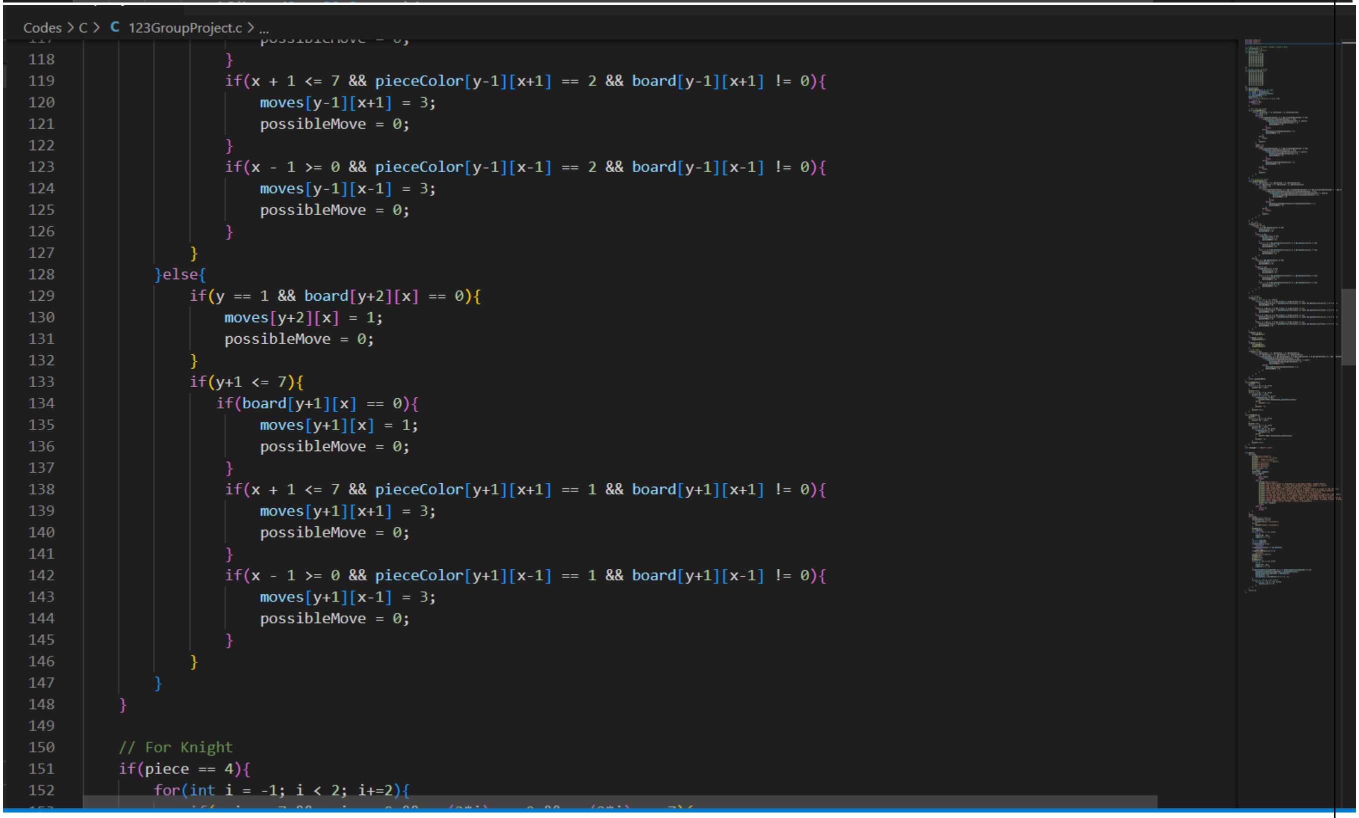
```c
    int possibleMove = 1;
    moves[y][x] = 2;
    if(x > 7 || x < 0 || y > 7 || y < 0)
        return 1;
    if(piece == 0){
        return 1;
    }

    // For rook and queen
    void straightMove(){
        for(int direction = -1; direction < 2; direction+=2){
            int count = 1;
            while(1){
                if(x+(count*direction) <= 7 && x+(count*direction) >= 0){
                    if(board[y][x+(count*direction)] != 0){
                        if(pieceColor[y][x+(count*direction)] != color){
                            moves[y][x+(count*direction)] = 3;
                            possibleMove = 0;
                        }
                        break;
                    }else{
                        moves[y][x+(count*direction)] = 1;
                        possibleMove = 0;
                    }
                }else{
                    break;
                }
                count++;
            }
            count = 1;
            while(1){
                if(y+(count*direction) <= 7 && y+(count*direction) >= 0){
                    if(board[y+(count*direction)][x] != 0){
                        if(pieceColor[y+(count*direction)][x] != color){
                            moves[y+(count*direction)][x] = 3;
```



Codes > C > C 123GroupProject.c > ...

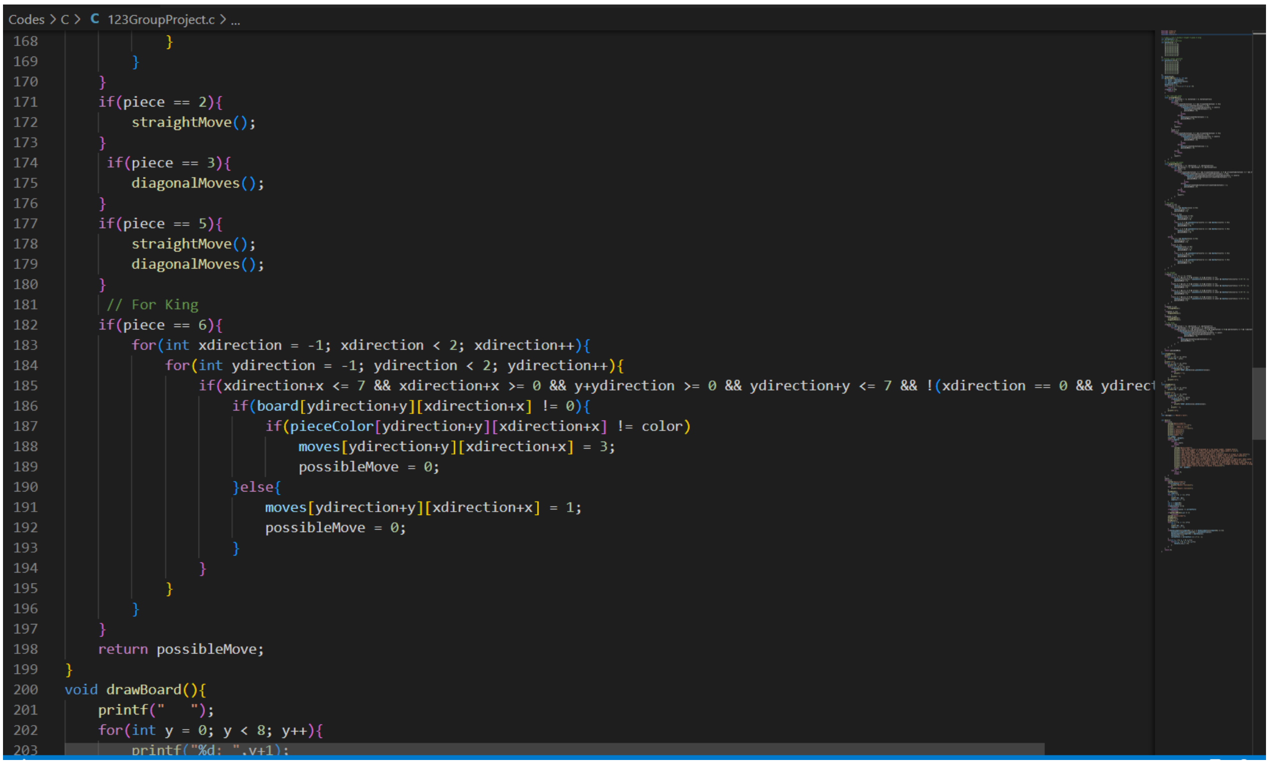
```c
        }
    }
    // For bishop and queen
    void diagonalMoves(){
        for(int xdirection = -1; xdirection < 2; xdirection+=2){
            for(int ydirection = -1; ydirection < 2; ydirection+=2){
                int count = 1;
                while(1){
                    if(x+(count*xdirection) <= 7 && x+(count*xdirection) >= 0 && y+(count*ydirection) <= 7 && y+(count*ydirection)
                        if(board[y+(count*ydirection)][x+(count*xdirection)] != 0){
                            if(pieceColor[y+(count*ydirection)][x+(count*xdirection)] != color){
                                moves[y+(count*ydirection)][x+(count*xdirection)] = 3;
                                possibleMove = 0;
                            }
                            break;
                        }else{
                            moves[y+(count*ydirection)][x+(count*xdirection)] = 1;
                            possibleMove = 0;
                        }
                    }else{
                        break;
                    }
                    count++;
                }
            }
        }
    }
    // For pawn
    if(piece == 1){
        if(color == 1){
            if(y == 6 && board[y-2][x] == 0){
                moves[y-2][x] = 1;
                possibleMove = 0;
            }
            if(y-1 >= 0){
```



Codes > C > C 123GroupProject.c > ...

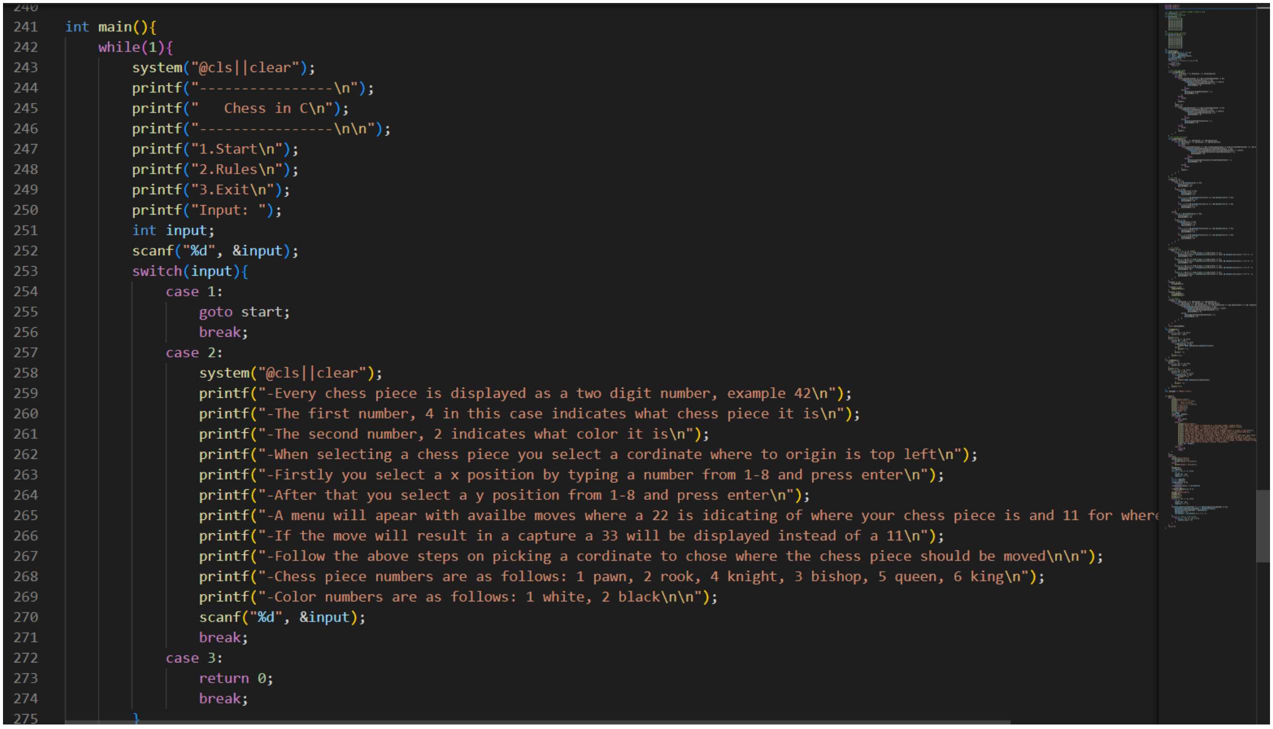
```c
                    possibleMove = 0;
                }
                if(x + 1 <= 7 && pieceColor[y-1][x+1] == 2 && board[y-1][x+1] != 0){
                    moves[y-1][x+1] = 3;
                    possibleMove = 0;
                }
                if(x - 1 >= 0 && pieceColor[y-1][x-1] == 2 && board[y-1][x-1] != 0){
                    moves[y-1][x-1] = 3;
                    possibleMove = 0;
                }
            }
        }else{
            if(y == 1 && board[y+2][x] == 0){
                moves[y+2][x] = 1;
                possibleMove = 0;
            }
            if(y+1 <= 7){
               if(board[y+1][x] == 0){
                    moves[y+1][x] = 1;
                    possibleMove = 0;
                }
                if(x + 1 <= 7 && pieceColor[y+1][x+1] == 1 && board[y+1][x+1] != 0){
                    moves[y+1][x+1] = 3;
                    possibleMove = 0;
                }
                if(x - 1 >= 0 && pieceColor[y+1][x-1] == 1 && board[y+1][x-1] != 0){
                    moves[y+1][x-1] = 3;
                    possibleMove = 0;
                }
            }
        }
    }

    // For Knight
    if(piece == 4){
        for(int i = -1; i < 2; i+=2){
```



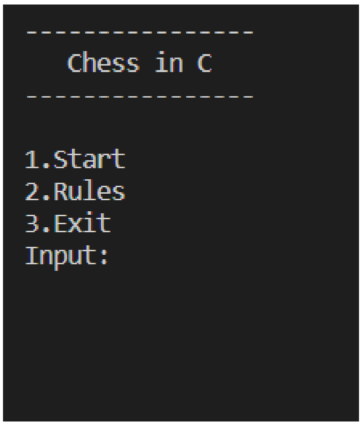
Codes > C > C 123GroupProject.c > ...

```c
            }
        }
    }
    if(piece == 2){
        straightMove();
    }
     if(piece == 3){
        diagonalMoves();
    }
    if(piece == 5){
        straightMove();
        diagonalMoves();
    }
     // For King
    if(piece == 6){
        for(int xdirection = -1; xdirection < 2; xdirection++){
            for(int ydirection = -1; ydirection < 2; ydirection++){
                if(xdirection+x <= 7 && xdirection+x >= 0 && y+ydirection >= 0 && ydirection+y <= 7 && !(xdirection == 0 && ydirect
                    if(board[ydirection+y][xdirection+x] != 0){
                        if(pieceColor[ydirection+y][xdirection+x] != color)
                            moves[ydirection+y][xdirection+x] = 3;
                            possibleMove = 0;
                    }else{
                        moves[ydirection+y][xdirection+x] = 1;
                        possibleMove = 0;
                    }
                }
            }
        }
    }
    return possibleMove;
}
void drawBoard(){
    printf("   ");
    for(int y = 0; y < 8; y++){
        printf("%d: ",y+1);
```



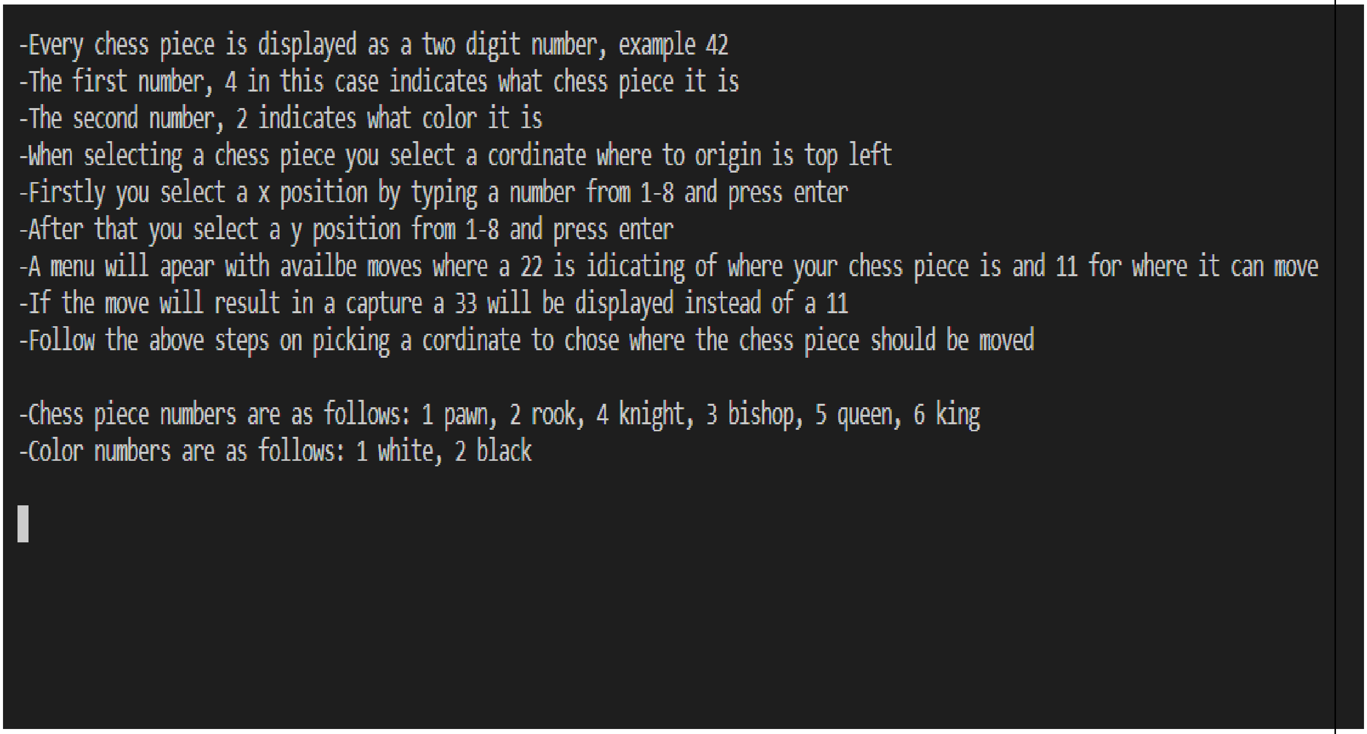
```c
int main(){
    while(1){
        system("@cls||clear");
        printf("----------------\n");
        printf("   Chess in C\n");
        printf("----------------\n\n");
        printf("1.Start\n");
        printf("2.Rules\n");
        printf("3.Exit\n");
        printf("Input: ");
        int input;
        scanf("%d", &input);
        switch(input){
            case 1:
                goto start;
                break;
            case 2:
                system("@cls||clear");
                printf("-Every chess piece is displayed as a two digit number, example 42\n");
                printf("-The first number, 4 in this case indicates what chess piece it is\n");
                printf("-The second number, 2 indicates what color it is\n");
                printf("-When selecting a chess piece you select a cordinate where to origin is top left\n");
                printf("-Firstly you select a x position by typing a number from 1-8 and press enter\n");
                printf("-After that you select a y position from 1-8 and press enter\n");
                printf("-A menu will apear with availbe moves where a 22 is indicating of where your chess piece is and 11 for where
                printf("-If the move will result in a capture a 33 will be displayed instead of a 11\n");
                printf("-Follow the above steps on picking a cordinate to chose where the chess piece should be moved\n\n");
                printf("-Chess piece numbers are as follows: 1 pawn, 2 rook, 4 knight, 3 bishop, 5 queen, 6 king\n");
                printf("-Color numbers are as follows: 1 white, 2 black\n\n");
                scanf("%d", &input);
                break;
            case 3:
                return 0;
                break;
        }
```

Output



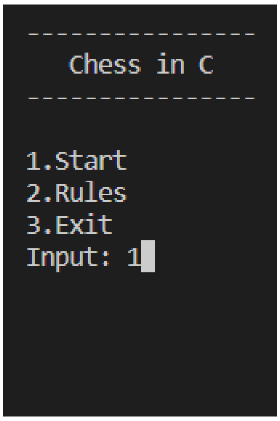
```
----------------
   Chess in C
----------------

1.Start
2.Rules
3.Exit
Input:
```



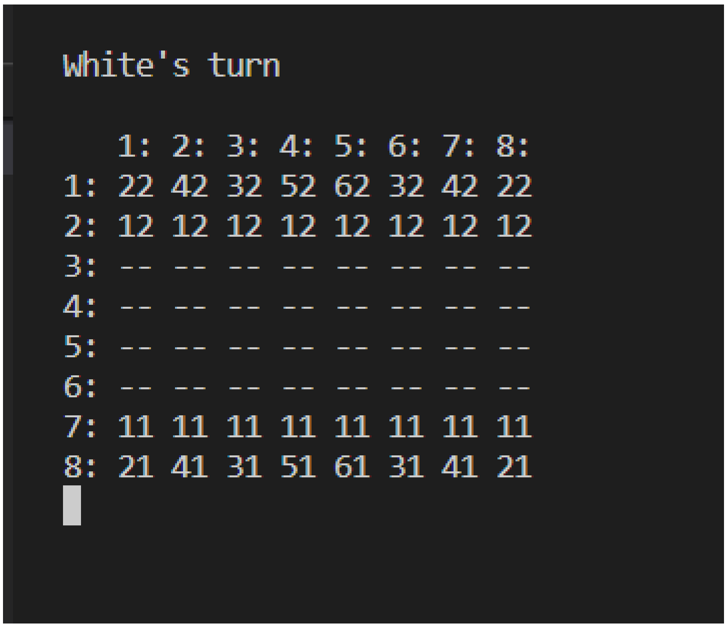
```
-Every chess piece is displayed as a two digit number, example 42
-The first number, 4 in this case indicates what chess piece it is
-The second number, 2 indicates what color it is
-When selecting a chess piece you select a cordinate where to origin is top left
-Firstly you select a x position by typing a number from 1-8 and press enter
-After that you select a y position from 1-8 and press enter
-A menu will apear with availbe moves where a 22 is idicating of where your chess piece is and 11 for where it can move
-If the move will result in a capture a 33 will be displayed instead of a 11
-Follow the above steps on picking a cordinate to chose where the chess piece should be moved

-Chess piece numbers are as follows: 1 pawn, 2 rook, 4 knight, 3 bishop, 5 queen, 6 king
-Color numbers are as follows: 1 white, 2 black
```



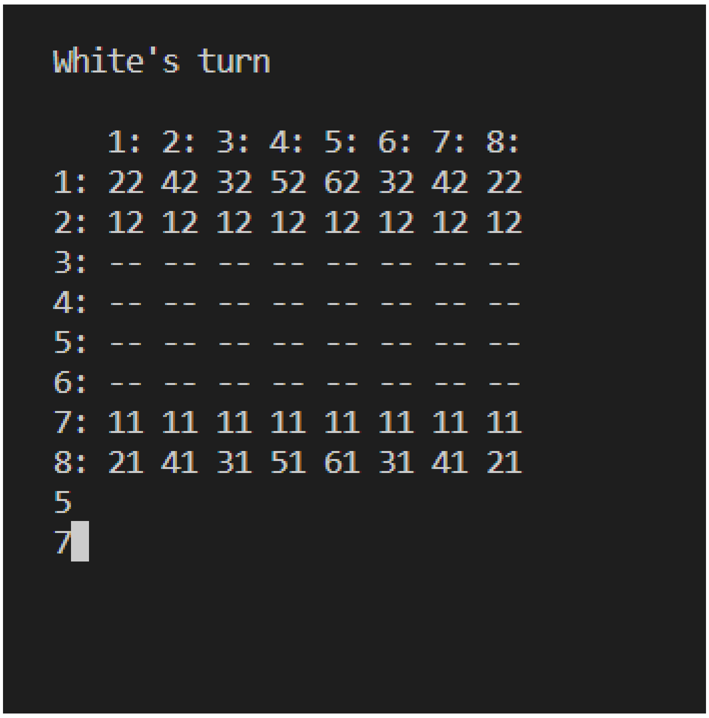
```
----------------
   Chess in C
----------------

1.Start
2.Rules
3.Exit
Input: 1
```



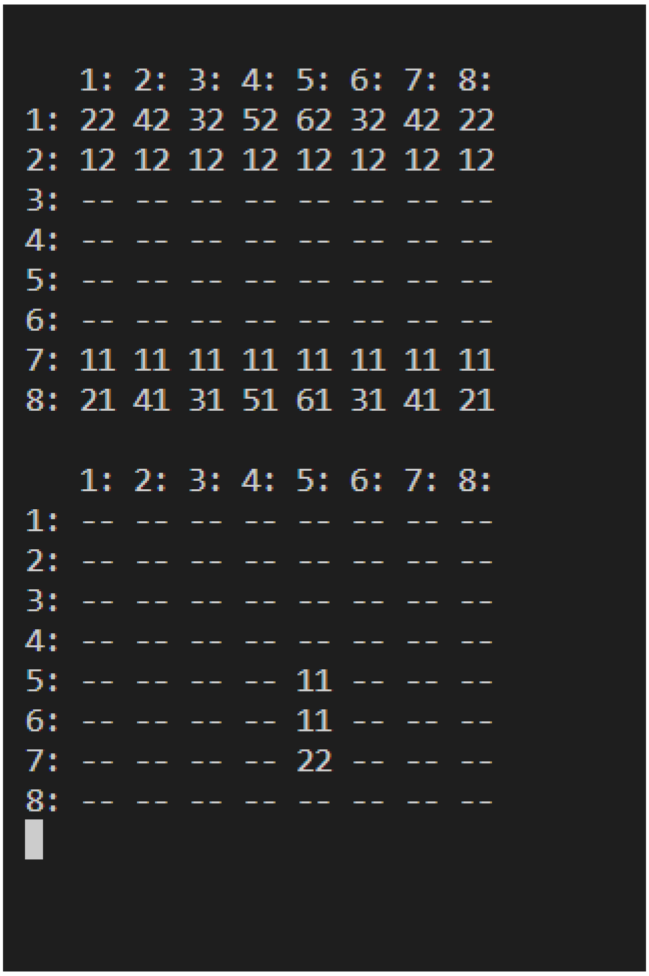
```
White's turn

   1: 2: 3: 4: 5: 6: 7: 8:
1: 22 42 32 52 62 32 42 22
2: 12 12 12 12 12 12 12 12
3: -- -- -- -- -- -- -- --
4: -- -- -- -- -- -- -- --
5: -- -- -- -- -- -- -- --
6: -- -- -- -- -- -- -- --
7: 11 11 11 11 11 11 11 11
8: 21 41 31 51 61 31 41 21
```



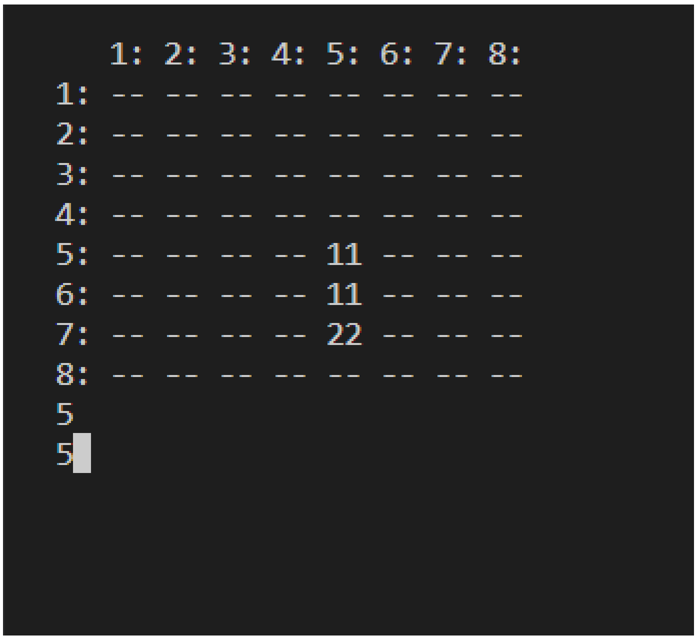
```
White's turn

   1: 2: 3: 4: 5: 6: 7: 8:
1: 22 42 32 52 62 32 42 22
2: 12 12 12 12 12 12 12 12
3: -- -- -- -- -- -- -- --
4: -- -- -- -- -- -- -- --
5: -- -- -- -- -- -- -- --
6: -- -- -- -- -- -- -- --
7: 11 11 11 11 11 11 11 11
8: 21 41 31 51 61 31 41 21
5
7
```

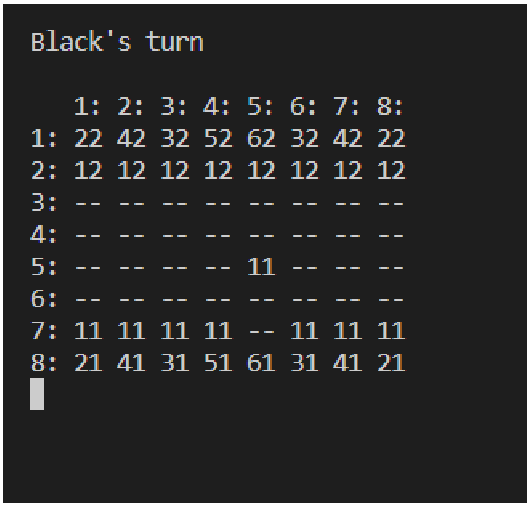


```
   1: 2: 3: 4: 5: 6: 7: 8:
1: 22 42 32 52 62 32 42 22
2: 12 12 12 12 12 12 12 12
3: -- -- -- -- -- -- -- --
4: -- -- -- -- -- -- -- --
5: -- -- -- -- -- -- -- --
6: -- -- -- -- -- -- -- --
7: 11 11 11 11 11 11 11 11
8: 21 41 31 51 61 31 41 21

   1: 2: 3: 4: 5: 6: 7: 8:
1: -- -- -- -- -- -- -- --
2: -- -- -- -- -- -- -- --
3: -- -- -- -- -- -- -- --
4: -- -- -- -- -- -- -- --
5: -- -- -- -- 11 -- -- --
6: -- -- -- -- 11 -- -- --
7: -- -- -- -- 22 -- -- --
8: -- -- -- -- -- -- -- --
```

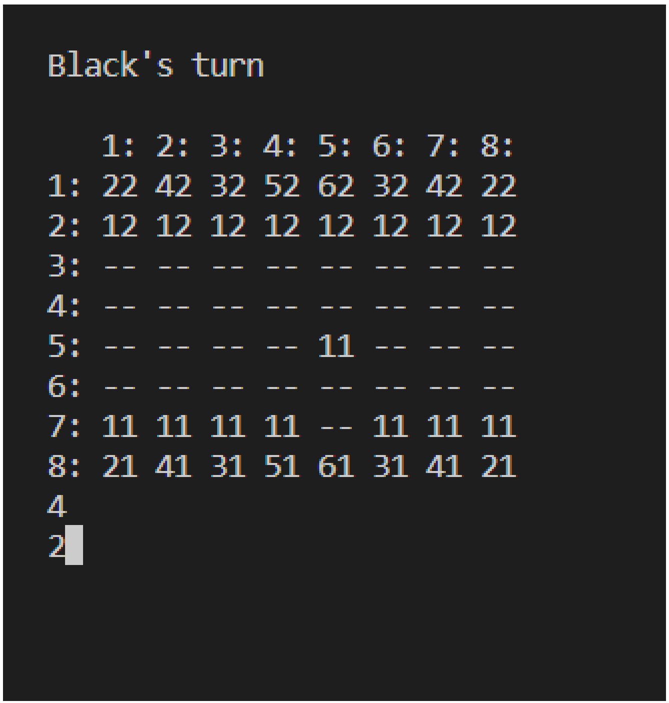


```
   1: 2: 3: 4: 5: 6: 7: 8:
1: -- -- -- -- -- -- -- --
2: -- -- -- -- -- -- -- --
3: -- -- -- -- -- -- -- --
4: -- -- -- -- -- -- -- --
5: -- -- -- -- 11 -- -- --
6: -- -- -- -- 11 -- -- --
7: -- -- -- -- 22 -- -- --
8: -- -- -- -- -- -- -- --
5
5
```



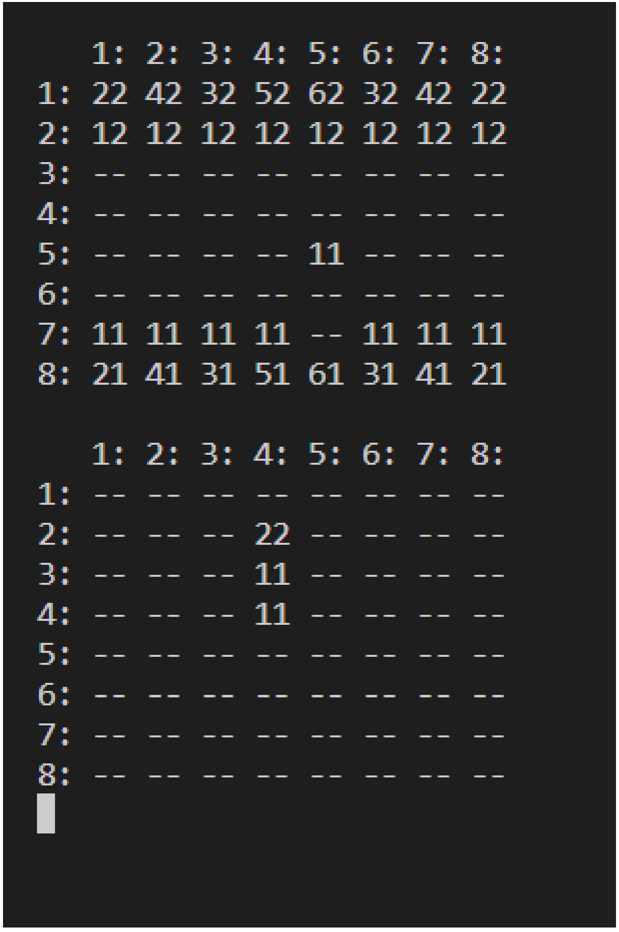
```
Black's turn

   1: 2: 3: 4: 5: 6: 7: 8:
1: 22 42 32 52 62 32 42 22
2: 12 12 12 12 12 12 12 12
3: -- -- -- -- -- -- -- --
4: -- -- -- -- -- -- -- --
5: -- -- -- -- 11 -- -- --
6: -- -- -- -- -- -- -- --
7: 11 11 11 11 -- 11 11 11
8: 21 41 31 51 61 31 41 21
```



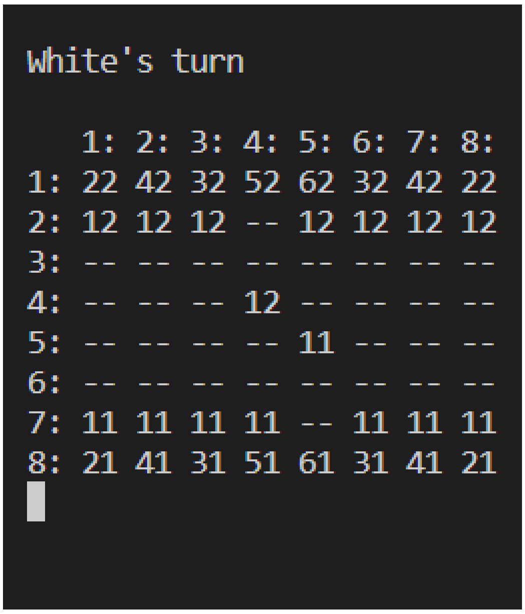
```
Black's turn

   1: 2: 3: 4: 5: 6: 7: 8:
1: 22 42 32 52 62 32 42 22
2: 12 12 12 12 12 12 12 12
3: -- -- -- -- -- -- -- --
4: -- -- -- -- -- -- -- --
5: -- -- -- -- 11 -- -- --
6: -- -- -- -- -- -- -- --
7: 11 11 11 11 -- 11 11 11
8: 21 41 31 51 61 31 41 21
4
2
```



```
   1: 2: 3: 4: 5: 6: 7: 8:
1: 22 42 32 52 62 32 42 22
2: 12 12 12 12 12 12 12 12
3: -- -- -- -- -- -- -- --
4: -- -- -- -- -- -- -- --
5: -- -- -- -- 11 -- -- --
6: -- -- -- -- -- -- -- --
7: 11 11 11 11 -- 11 11 11
8: 21 41 31 51 61 31 41 21

   1: 2: 3: 4: 5: 6: 7: 8:
1: -- -- -- -- -- -- -- --
2: -- -- -- 22 -- -- -- --
3: -- -- -- 11 -- -- -- --
4: -- -- -- 11 -- -- -- --
5: -- -- -- -- -- -- -- --
6: -- -- -- -- -- -- -- --
7: -- -- -- -- -- -- -- --
8: -- -- -- -- -- -- -- --
```



```
White's turn

   1: 2: 3: 4: 5: 6: 7: 8:
1: 22 42 32 52 62 32 42 22
2: 12 12 12 -- 12 12 12 12
3: -- -- -- -- -- -- -- --
4: -- -- -- 12 -- -- -- --
5: -- -- -- -- 11 -- -- --
6: -- -- -- -- -- -- -- --
7: 11 11 11 11 -- 11 11 11
8: 21 41 31 51 61 31 41 21
```

Future Scopes

This program helped us understand the fundamentals of c programming language and will help us in solving more complicated problems in the future. Changes could be made in the game so that players can even chat with each other while playing.

Conclusion

- This is a project made with the ‘C’ programming language. The program is designed for playing and having fun. This program contains functions, switch cases, and memory allocation and can be modified to do more things.

- Meanwhile, various interaction ways in terminal are provided to make our program easier to use.
- This is a project made with the ‘C’ programming language. The program is designed for playing and having fun. This program contains functions, switch cases, and memory allocation and can be modified to do more things.
- Thanks to all who helped us to complete this project, especially our PIC professor.