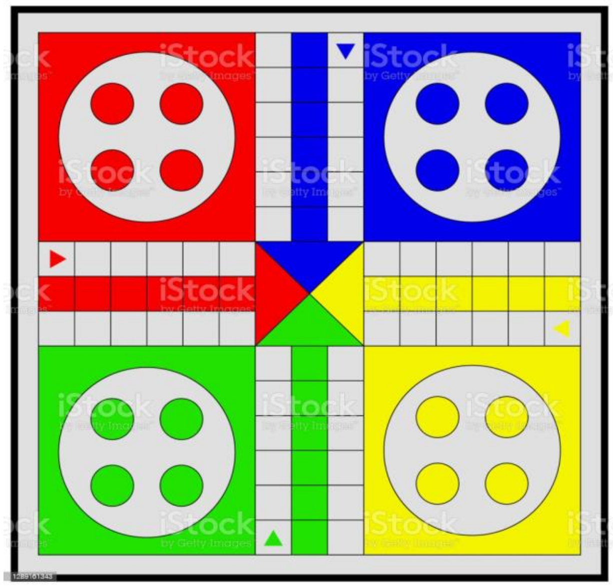
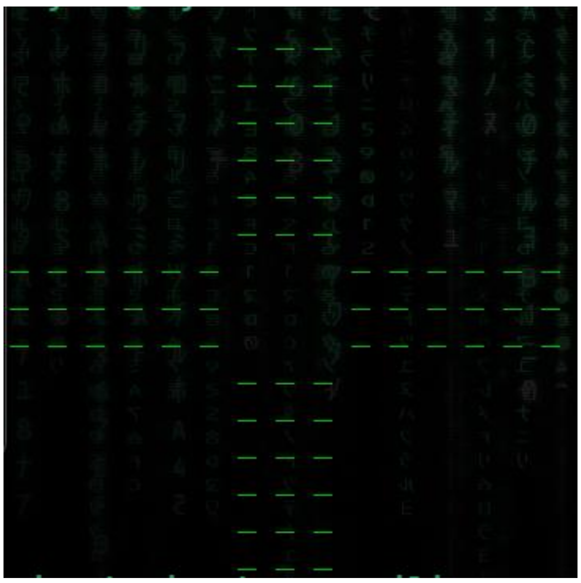
LUDO PROJECT

(Terminal – Based)





TEAM

Riyank Singh (202251127)
Nipur Patel (202251092)

Table of Content

Introduction

Ludo is four player board game. Each player is assigned a colour and has four tokens in their colour. Special areas of the Ludo board are typically coloured bright yellow, green, red, and blue. The board is normally square with a cross-shaped play space, with each arm of the cross having three columns of squares, usually six per column. The middle columns usually have five squares coloured; these represent a player's home column. A sixth coloured square not on the home column is a player's starting square. At the centre of the board is a large finishing square. A player wins if all his/her four tokens reach to home.

Challenges

Challenge #1 – Display of Board.

Solution: In Display function

Challenge #2 – Handling of Token position and updating them with the dice value.

Solution: In area_finder and area functions.

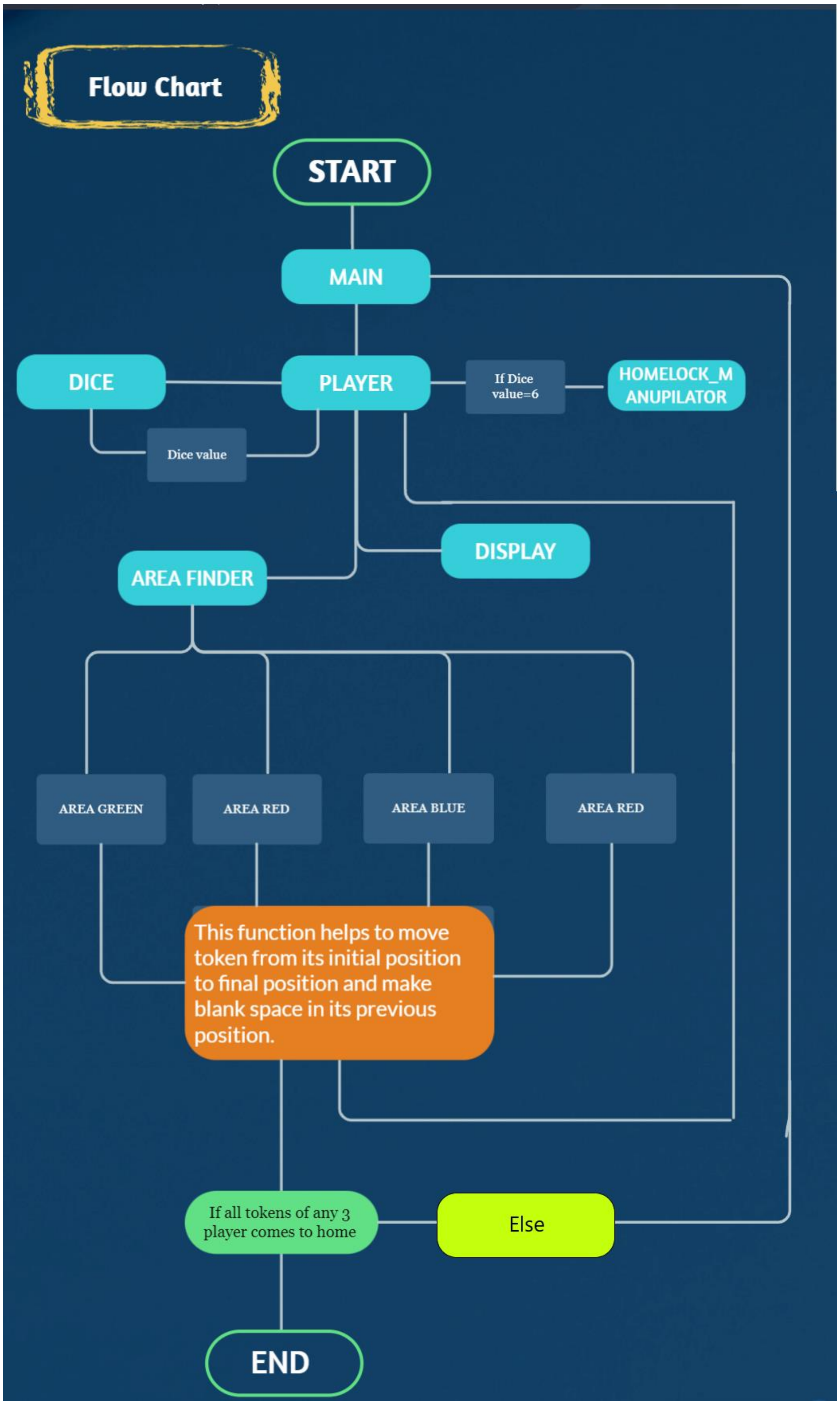
Challenge #3 – Making the game Single player with multi-tokens

Solution: In Player Functions.

Challenge #4 – Making the game Multiplayer.

Solution: In Player Functions.

Flowchart of the functions of code.


Flow Chart
START
MAIN
DICE
PLAYER
If Dice value=6
HOMELOCK_MANUPILATOR
Dice value
DISPLAY
AREA FINDER
AREA GREEN
AREA RED
AREA BLUE
AREA RED
This function helps to move token from its initial position to final position and make blank space in its previous position.
If all tokens of any 3 player comes to home
Else
END

Explanation and Code for the Game

The int main Function & Global Declarations

The Main Function loads the game by assigning the colour area arrays the default value='_', assigns the data to the storage arrays. The main function is preceded by a lot of global variable and function declarations.

int tokens[16][5] (Global)

It contains the data for all 16 tokens. The tabulated form of this array is given in table 2.

	0	1	2	3	4
Unique Code	Sequence	Area (1-4)	Location (1-18)	Colour	Symbol
0	1			1	*
1	2			1	*
2	3			1	*
3	4			1	*
4	1			2	+
5	2			2	+
6	3			2	+
7	4			2	+
8	1			3	/
9	2			3	/
10	3			3	/
11	4			3	/
12	1			4	?
13	2			4	?
14	3			4	?
15	4			4	?

Table 2

LUDO (TERMINAL BASED)

int homecount[4] (Global)

This array consists the count of number of tokens of a particular colour who have completed their round in the board and now are in home.

int choices[4] (Global)

This array consists the number of choices of tokens a particular player has.

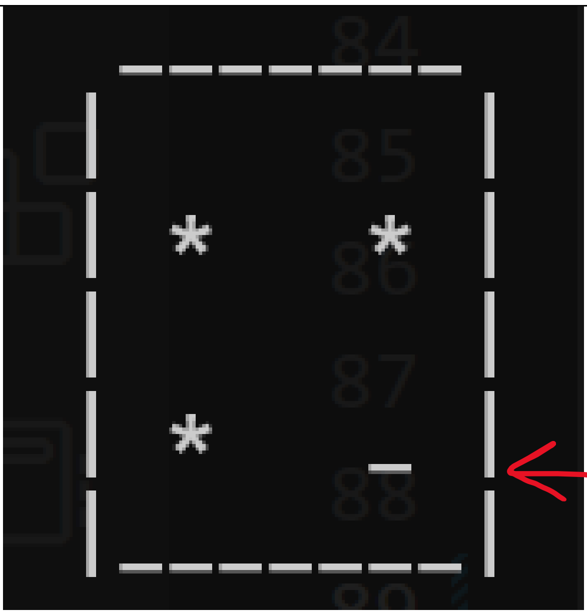
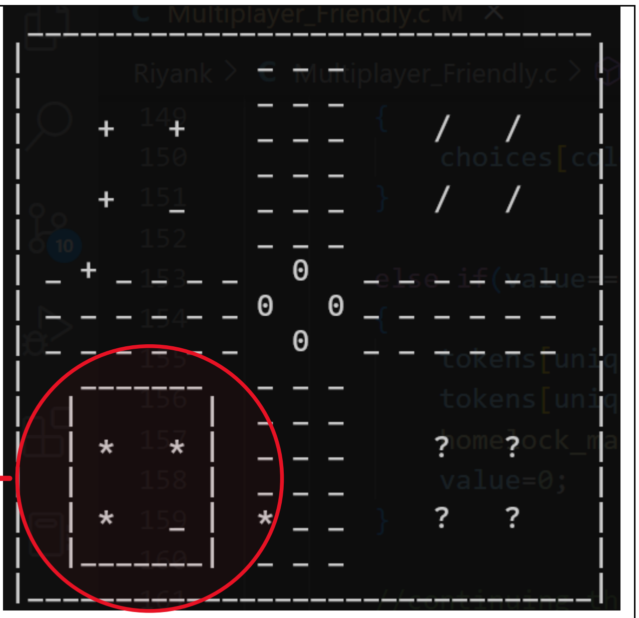
char rea arrays (local to main)

1) green_area[6][3] (global pointer name “gr_ptr”)
2) blue_area[6][3] (global pointer name “bl_ptr”)
3) red_area[3][6] (global pointer name “re_ptr”)
4) yellow_area[3][6] (global pointer name “ye_ptr”)

These arrays store the game progress graphically in the form of 2- D array. They all have initial value ‘_’ representing blank spaces. To represent a token, we replace this value with respective colour symbol given in table 2.

char homelock [4][4] (local to main)

This array is made for displaying home lock tokens. This array is accessed by “homelock_mainpulator” function. It stores the colour symbol

 Enlarged view of home lock area of green.	 The Board

For making this array we have used global array pointer namely “homelock_ptr”.

Working of main function

1) Firstly, it assigns the arrays the initial values which are required for the game. In simple words it loads the game.
2) Then it one by one pass 1,2,3,4 as values of ‘colourcode’ to function ‘player’. This simulates player turns, until the game terminates.
3) Terminating Condition: The game gets end when any three player’s home count value is equals to 4.

Code for main function:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int i,j;
void display(int); //for displaying (giving input the four char arrays)

void area_finder(int, int,char); //value, initial kingdomn(kingdom number and pos-
tion)

void area_g(int, char, int); //navigation function in green kingdom
void area_r(int, char, int); //navigation function in red kingdom
void area_b(int, char, int); //navigation function in blue kingdom
void area_y(int, char, int); //navigation function in yellow kingdom

void homelock_manipulator(int); //for manipulating the homelock strings
void player(int); //game console

int dice(void); //dice

int tokens[16][5]; //coords of tokens (sequence/area/location/colourcode/coor sym-
bol/coords x/ coord y)
int homecount[4]={0,0,0,0}; //declaring count of homecount tokens (won tokens)
(g,r,b,y)
int choices[4]={0,0,0,0}; //showing no. of choices (g,r,b,y)

char *gr_ptr, *bl_ptr, *re_ptr, *ye_ptr;//pointers pointing to the area arrays
char *homelock_ptr; //pointer pointing to the arrays storing data for homelock ptr.

int main(void)
{
    //defining areas
    char green_area[6][3],blue_area[6][3],red_area[3][6], yellow_area[3][6];
    gr_ptr=&green_area[0][0];bl_ptr=&blue_area[0][0];re_ptr=&red_area[0][0];ye_ptr=
&yellow_area[0][0];

    for(i=0;i<18;i++)
    {
        *(gr_ptr+i)=*(bl_ptr+i)=*(re_ptr+i)=*(ye_ptr+i)='_';//assigning the blank 
spaces.
    }

    for(i=0;i<16;i++) //feeding sequence data
    {
        tokens[i][0]=i+1;
    }

    for(i=0;i<4;i++) //feeding colourcode
    {
        tokens[i][3]=1;
        tokens[i+4][3]=2;
        tokens[i+8][3]=3;
        tokens[i+12][3]=4;
    }

    for(i=0;i<4;i++) //feeding coloursymbol
    {
        tokens[i][4]=(int)'*';
        tokens[i+4][4]=(int)'+';
        tokens[i+8][4]=(int)'/';
        tokens[i+12][4]=(int)'?';
    }

    char homelock [4][4]={ //assigning sybmols
                            {"****"},
                            {"++++"},
                            {"////"},
                            {"????"}
                         };
    homelock_ptr=&homelock[0][0]; //mapping pointer

    system("cls"); //for windows users replease clear to cls

    int k=0;

    while((homecount[0]!=4 && homecount[1]!=4 && homecount[2]!=4) || (home-
count[1]!=4 && homecount[2]!=4 && homecount[3]!=4)
    || (homecount[2]!=4 && homecount[3]!=4 && homecount[0]!=4) || (homecount[3]!=4 
&& homecount[0]!=4 && homecount[1]!=4))
    {
        player(k%4+1);
        k++;
    }
    system("cls");
    display(0);
}
```

The "player" function

This functions handles the events of this games that are to take user input, validate them, process the data by calling "area_finder" function and finally displaying the changes by calling the function "homelock_manipulator" and "display".

The input of this is function is a number which indicates the whose turn it is (Player green, red, blue or yellow." E.g. if input is 1, then it is the turn of green.

Working of player function

1) Firstly, its validate whether the 'home count' of that player is less than 4 or not. If it is greater than 4 then it simply goes to end by the control statement "goto: end".
2) Then it gets the dice value by function "dice".
3) Then it displays the board by calling the function "display"
4) If the dice value is equal to 6, then it increases the choice value by 1 of that user.
5) If the choice is greater than 1, then is also asks the user for choice number.
6) For a new token in the board, it call the "homelock_manipulator" for clearing the past location of the newly deployed token. Then, it assigns the initial coordinate of ("colourcode," 14) (see area function for explaination) to the new token.
7) At last, it calls "area_finder" function to the do the changes in the area array.

Code for "player" function:

```
void player(int colourcode) //game event handler
{
    bool flag=1;

    printf("Player %d's turn\n", colourcode);
    colourcode--;
    char dummy; //for handling enter of getchar
    int user_choice=1; //default choice of user
    int value; //dice value

    int unique_id=4*(colourcode) + (user_choice-1); //storing unique value

    if (homecount[colourcode]==4) //if the player has won
    {
        goto end;
    }

    value=dice(); //getting random dice value

    printf("\nLUDO GAME (TERMINAL BASED)\n");

    display(colourcode+1); //displaying board
    printf("PRESS ENTER TO CONTINUE.");
    dummy=getchar();

    if(value==6 && choices[colourcode]+homecount[colourcode]<5) //terminating con-
diton for inreasing choice
    {
        choices[colourcode]++; //max choices =4
    }

    for(int i=0;i<16;i++)
    {
        printf("\nThe Coords for %d are %d %d", tokens[i][0], tokens[i][1], to-
kens[i][2]); //for debuging process
    }

    printf("\nThe Dice Value is %d\n", value);

    if(choices[colourcode]>0)
    {
        if(choices[colourcode]>1||homecount[colourcode]>1)
        {
            begin:
            printf("Enter the choice: ");
            scanf("%d", &user_choice);
            flag=0;

            if(user_choice<=homecount[colourcode])
            {
                printf("\nThis token is already in the home.\n");
                goto begin;
            }

            else if(user_choice>choices[colourcode])
            {
                printf("\nThis token is in homelock.\n");
                goto begin;
            }

            unique_id=4*(colourcode) + (user_choice-1);
        }

/*if the player gets dice value to 6 and she/he wishes to not to put a token out of
homelock
then decreasing the choice.*/

        if(value==6 && user_choice!=homecount[colourcode]+choices[colourcode])
        {
            choices[colourcode]--;
        }
        else if(value==6 && choices[colourcode]!=5) //bringing the token out of
homelock
        {
            tokens[unique_id][1]=colourcode+1; //assigning the spawn area to new
token
            tokens[unique_id][2]=14; //assigning the spawn location to new token
            homelock_manipulator(tokens[unique_id][3]-1);
            value=0;
        }

        //continuing the game
        area_finder(value, unique_id, tokens[unique_id][4]); //value, sequence,
colour symbol
    }

        if(choices[colourcode]<2 && flag) //for only one choice
        {
        printf("\nPRESS ENTER TO CONTINUE.");
        dummy=getchar();
        }

         end: //if the player had won //multiplayer

        system("clear");
}
```

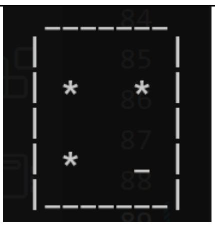
The "homelock_manipulator" function

This function access the char array homelock[4][4] with the help of the char pointer ptr_homelock. The function of this to modify the homelock array by assigning a blank space.

Working of "homelock_manipulator" function

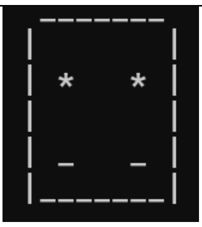
This function based upon colour code and number of choices, assign the '_' to the homelock array.

So let's take an example, there is one token out of homelock of player green and in the turn of green the dice value is 6, he give the choice 2. So, the 2nd token will be deployed on the board at (1,14) (area, location). Before calling this function, the array was "***_" which modifies to "**__".



```
 -------
|       |
| *   * |
|       |
| *   _ |
|_______|
```

Before



```
 -------
|       |
| *   * |
|       |
| _   _ |
|_______|
```

After

Code of “homelock_manipulator” function

```
//based upon no of tokens out of hoemlock, changing the array of homelock
void homelock_manipulator(int colourcode)
{
    switch (4-homecount[colourcode]-choices[colourcode]) // no of homelock_tokens
colorcode(0,1,2,3)
    {
    case 0:
        *(homelock_ptr+4*colourcode)='_';
        break;

    case 1:
        *(homelock_ptr+4*colourcode+1)='_';
        break;

    case 2:
        *(homelock_ptr+4*colourcode+2)='_';
        break;

    case 3:
        *(homelock_ptr+4*colourcode+3)='_';
        break;

    default:
        break;
    }
}
```

The “area_finder” function

This function by pass the value, token unique number and the colours symbol to the area(area_g, area_r, area_b, area_y) functions based upon the area presently the token is it.

Input (Initial Area)	Function call
1	area_g
2	area_r
3	area_b
4	area_y

Code for “area_finder” function

```
//identifying location(area) of token from its area and calling the respective area
function
void area_finder(int value, int user_choice,char c)
{
    switch(tokens[user_choice][1])
    {
        case 1:
        area_g(value,c,user_choice);
        break;

        case 2:
        area_r(value,c,user_choice);
        break;

        case 3:
        area_b(value,c,user_choice);
        break;

        case 4:
        area_y(value,c,user_choice);
        break;
    }
}
```

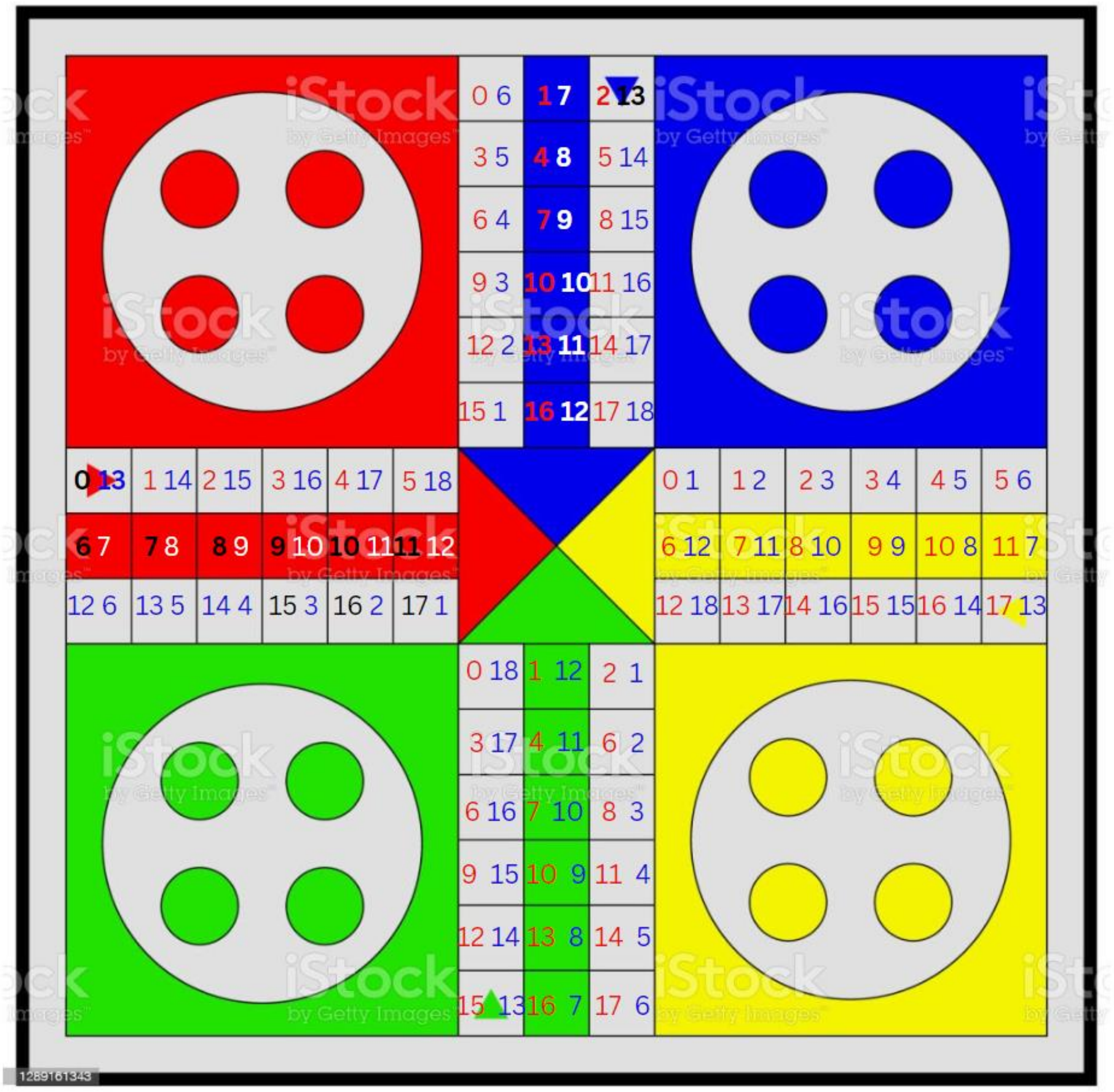
The area_c functions (c={g,r,b,y})

Input: colour code (for showing whose turn it is)
Output: null

The area functions are responsible for moving the tokens into their respective areas. These functions update the coordinates of the tokens based upon the dice value they get. They also update the character arrays such that graphically it looks that the pieces are moving from their initial place to the final place.

Coordinate System for navigation in the board

As the game is a 2-d board, we need a coordinate system for making coding simple and efficient for moving a token.



First number = Array Index
Second number = value of location parameter

Mapping of Array Index with Location Parameter

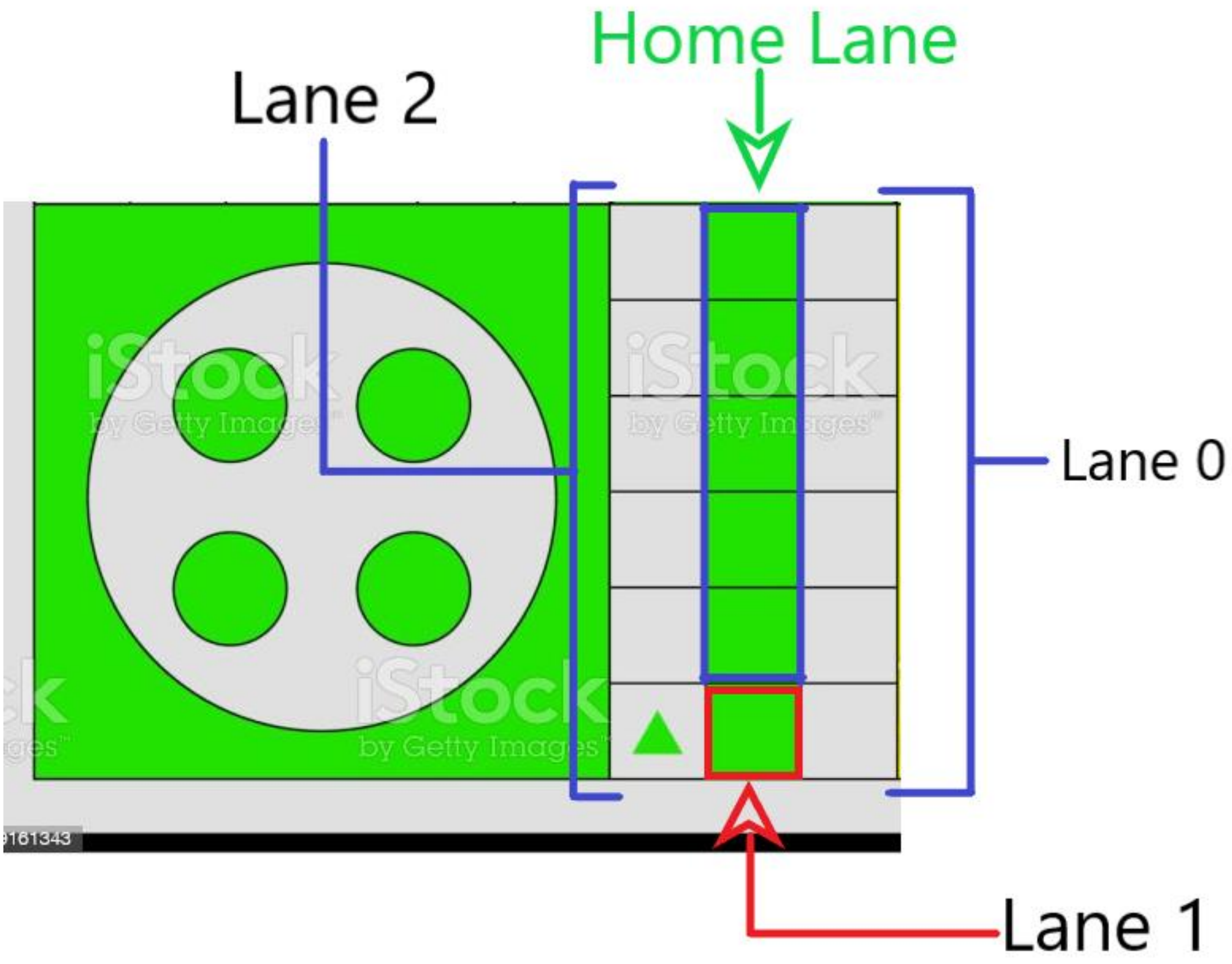
As we have four characters array and four area functions for modifying them. So, let's the first parameter of coordinate system as area. Area parameter value of a token ranges from 1 to 4 denoting the four areas.

As each area array has 18 elements, we can define another coordinate parameter namely 'location'. The value of location parameter can vary from 1 to 18.

Now each area is divided into four lanes to make it easy to map the array indexes with the value of location parameter. The lanes are:

1) lane 0 (location parameter between 1 to 6)
2) lane 1 (location parameter = 7)

3) home lane (location parameter between 8 to 12)
4) lane 2 (location parameter between 13 to 18)



Home lane is accessible only to the token which is of the same colour as of the area.

Working of "area" function

1) First of all, the area function keeps a backup of the initial location of the token in a temporary variable 'temp' and then computes the location where the chosen token will be after the player's turn.

2) If the value exceeds 18 (indicating the token is going to change the area), it will increase the parameter area value by 1 (for area=4, it will modify it to 1), then computes where the token would be in the next area. After doing this it directly goes to the last of the function.
3) If the value is below 18 or equal to 18, then it checks in which lane the token will land. Upon finding the value of final location parameter, it modifies that the array element corresponding to that location to the colour symbol.

4) We also need to clear the previous location of the array element i.e., to assign '_' value to the array element corresponding to old location. The corresponding array element is found out in a similar fashion as the new location in step 3.

5) It also takes care of the tokens in the home lane. It dumbs the increment in the location parameter if the final location parameter exceeds 13.

6) If the token location parameter value is equal to 13 and the colour of the token is same as the area, then the function increases the homecount of that colour by 1.

This functionality is similar to in all the area functions.

Code of area_g function

```c
void area_g(int value, char c, int user_choice)
{
    int temp=tokens[user_choice][2]; //temp for clearing past place
    tokens[user_choice][2]+=value; //calculating the final position
    int x=tokens[user_choice][2]; //taking x as a replica of token[user_choice][2]

    if(x>18) //checking the value eligible for next area
    {
        *(gr_ptr+(18-temp)*3)='_';//clearing past record

        value=x-18; //finding the coordinates to move in next kingdom

        tokens[user_choice][2]=value; //assigning the coords of the next kingdom

        value=0; //as coordinates are already assigned, putting value = 0

        tokens[user_choice][1]++;

        area_r(value, c, user_choice);
        goto end;
        //to move to next kingdom
    }

    //moving pice to new location
    if(x<=6) //for lane 0
    {
        *(gr_ptr+2+3*(x-1))=c;
    }

    else if(x==7) //for lane 1
    {
        *(gr_ptr+1+3*5)=c;
    }
```

```
else if(tokens[user_choice][3]==1 && (x<13 || temp<13)) //for home lane
{
    if(x==13)  //condition for reaching homecount
    {
        homecount[0]++; //increasing count by 1
        tokens[user_choice][1]=0;
        tokens[user_choice][2]=0;
    }

    else if(x>13) //if value is larger then ignoring the increament
    {
        x=temp;
        value=0;
    }

    else //for moving in home lane
    {
    *(gr_ptr+(12-x)*3+1)=c;
    }
}

else //for lane 2
{
    if(x>7 && x<13)
    {
        x+=5;
    }

    else if(x==13 && temp==7)
    {
        x=18;
    }
    *(gr_ptr+(18-x)*3)=c;
}

tokens[user_choice][2]=x;

end:
//clearing previous location
//for finding temp coordinate, we have used the same approcah as done to find x

if(value!=0)
{
if(temp<=6) //for lane 1
{
    *(gr_ptr+2+3*(temp-1))='_';
}

else if(temp==7) //for value is equal to 7
```

```
    {
        *(gr_ptr+1+3*5)='_';
    }

    else if(tokens[user_choice][3]==1 && temp<13) //for home lane
    {
        *(gr_ptr+(12-temp)*3+1)='_';
    }

    else //for values between 13 to 18
    {
        if(temp>7 && temp<13)
        {
        temp+=5;
        }

        *(gr_ptr+(18-temp)*3)='_';
    }
    }
}
```

Code for area_r function

```
void area_r(int value, char c, int user_choice)
{
    int temp=tokens[user_choice][2];
    tokens[user_choice][2]+=value;
    int x=tokens[user_choice][2];

    if(x>18)
    {
        *(re_ptr+(temp-13))='_';//clearing past record

        value=x-18;

        tokens[user_choice][2]=value; //assigning the coords of the next kingdom;

        value=0;
        tokens[user_choice][1]+=1;
        area_b(value,c,user_choice);
        goto end;
        //to move to next kingdom
    }

    //moving pice to new location
    if(x<=6)
    {
        *(re_ptr+18-x)=c;
    }
    else if(x==7)
    {
        *(re_ptr+6)=c;
    }

    else if(tokens[user_choice][3]==2 && (x<13 || temp<13)) //for home lane
    {
        if(x==13)  //condition for reaching homecount
        {
            homecount[1]++;; //increasing count by 1
            tokens[user_choice][1]=0;
            tokens[user_choice][2]=0;
        }

        else if(tokens[user_choice][2]>13) //if value is larger then dumping the
increament
        {
            x=temp;
            value=0;
        }

        else
        {
            *(re_ptr+x-1)=c;
        }
    }

    else
    {
        if(x>7 && x<13)
        {
            x+=5;
        }

        else if(x==13 && temp==7)
        {
            x=18;
        }

        *(re_ptr+(x-13))=c;
    }

    tokens[user_choice][2]=x;

    //clearing previous location

    end:
```

```
    if(value!=0)
    {
    if(temp<=6)
    {
        *(re_ptr+18-temp)='_';
    }

    else if(temp==7)
    {
        *(re_ptr+6)='_';
    }

    else if(tokens[user_choice][3]==2 && temp<13)
    {
        *(re_ptr+temp-1)='_';
    }

    else
    {
        if(temp>7 && temp<13)
        {
            temp+=5;
        }
        *(re_ptr+temp-13)='_';
    }
    }
}
```

Code for area_b function

```
void area_b(int value, char c, int user_choice)

{
    int temp=tokens[user_choice][2];
    tokens[user_choice][2]+=value;
    int x=tokens[user_choice][2];

    if(x>18)
    {
        *(bl_ptr+3*temp-37)='_';//clearing past record

        value=x-18;
        tokens[user_choice][2]=value;
        value=0;
        tokens[user_choice][1]++;
        area_y(value, c, user_choice);
        goto end;
        //to move to next kingdom
    }
```

```
if(x<=6) //moving pice to new location
{
    *(bl_ptr+18-(3*(x)))=c;
}

else if(x==7)
{
    *(bl_ptr+1)=c;
}

else if(tokens[user_choice][3]==3 && (x<13 || temp<13))
{
    if(x==13)  //condition for reaching homecount
    {
        homecount[2]++;; //increasing count by 1
        tokens[user_choice][1]=0;
        tokens[user_choice][2]=0;
    }

    else if(x>13) //if value is larger then dumping the increament
    {
        x=temp;
        value=0;
    }

    else
    {
    *(bl_ptr+1+(x-7)*3)=c;
    }
}

else
{
    if(x>7 && x<13)
    {
    x+=5;
    }

    else if(x==13 && temp==7)
    {
        x=18;
    }

    *(bl_ptr+3*(x)-37)=c;
}

tokens[user_choice][2]=x;
```

```
    end: //for moving to next kingdom

    //clearing previous location

    if(value!=0)
    {

    if(temp <= 6) //moving pice to new location
    {
        *(bl_ptr+18-(3*temp))='_';
    }

    else if(temp == 7)
    {
        *(bl_ptr+1)='_';
    }

    else if(tokens[user_choice][3]==3 && temp<13)
    {
        *(bl_ptr+1+(temp-7)*3)='_';
    }

    else
    {
        if(temp>7 && temp<13)
        {
        temp+=5;
        }
        *(bl_ptr+3*temp-37)='_';
    }
    }
}
```

Code for area_y function

```
void area_y(int value, char c, int user_choice)
{
    int temp=tokens[user_choice][2];
    tokens[user_choice][2]+=value;
    int x=tokens[user_choice][2];

    if(x>18)
    {
        *(ye_ptr+30-temp)='_';

        value=x-18;
        tokens[user_choice][2]=value;
        value=0;
        tokens[user_choice][1]=1;
        area_g(value,c,user_choice);
        goto end;
        //to move to next kingdom
    }

    if(x<=6) //moving pice to new location
    {
        *(ye_ptr+x-1)=c;
    }

    else if(x==7)
    {
        *(ye_ptr+11)=c;
    }

    else if(tokens[user_choice][3]==4 && (x<13 || temp<13))
    {
        if(x==13)  //condition for reaching homecount
        {
            homecount[3]++;; //increasing count by 1
            tokens[user_choice][1]=0;
            tokens[user_choice][2]=0;
        }

        else if(x>13) //if value is larger then dumping the increament
        {
            x=temp;
            value=0;
        }

        else
        {
        *(ye_ptr + 18 - x)=c;
        }
    }

    else
    {
        if(x>7 && x<13)
        {
        x+=5;
        }

       else if(x==13 && temp==7)
        {
            x=18;
        }

        *(ye_ptr+30-x)=c;
    }

    tokens[user_choice][2]=x;

    end:

    //clearing previous location

    if(value!=0)
    {
        if(temp<=6) //moving pice to new location
        {
            *(ye_ptr+temp-1)='_';
        }

        else if(temp==7)
        {
            *(ye_ptr+11)='_';
        }

        else if(tokens[user_choice][3]==4 && temp<13)
        {
            *(ye_ptr + 18 - temp)='_';
        }

        else
        {
            if(temp>=7 && temp<13)
            {
            temp+=5;
            }
            *(ye_ptr+30-temp)='_';
        }
    }
}
```

The Dice Function

This function uses the rand() function from the stdlib.h library. It gives a random number.

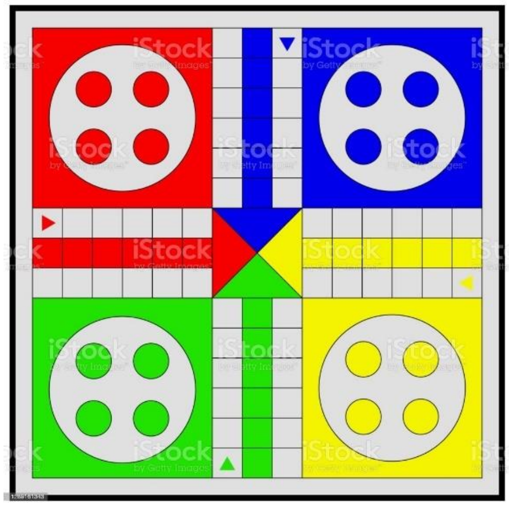
We perform the modulus operator with respect to 6. This returns value ranging from 0 to 5. By increasing the value by 1, it will return the values ranging from 1 to 6.

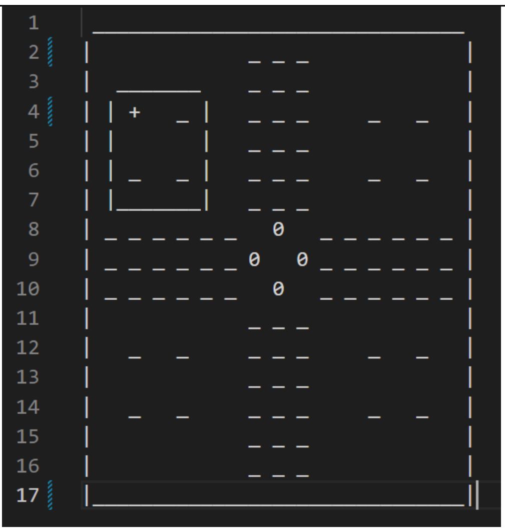
Code for dice function

```
int dice(void)
{
    float f;
    unsigned short int r;

    r = rand();
    //As modules of 6 results in (0,1,2,3,4,5). So it gives almost equal probabil-
ity distribution of probability
    r = 1 + (r % 6); //increasing by 1 to get (1,2,3,4,5,6)

    return (r);
}
```

The display Function

It basically displays the board with all the tokens by printing all the area arrays, home lock array, home count array. We are able to make it look good by doing pattern printing. We followed a map while developing this function.


Actual Ludo board



```
 1    ________________________________
 2   |             _ _ _               |
 3   |   _______   _ _ _               |
 4   | | +    _ |  _ _ _    _   _      |
 5   | |        |  _ _ _               |
 6   | | _    _ |  _ _ _    _   _      |
 7   | |________|  _ _ _               |
 8   | _ _ _ _ _ _   0   _ _ _ _ _ _   |
 9   | _ _ _ _ _ _ _0   0_ _ _ _ _ _   |
10   | _ _ _ _ _ _   0   _ _ _ _ _ _   |
11   |             _ _ _               |
12   |   _    _    _ _ _    _   _      |
13   |             _ _ _               |
14   |   _    _    _ _ _    _   _      |
15   |             _ _ _               |
16   |             _ _ _               |
17   |________________________________||
```

Ludo Terminal based

The numbers (now zeroes) at the centre represents the home count (number of tokens who reached the home).

Working of "area" function

For understanding the functionality, we can divide the map into three sections i.e.,

(i) From line 1 to line7
(ii) From line 8 to line 10
(iii) From line 11 to line 17

(i) From line 1 to line 7

1) We adjusted the spaces and design for highlighting the home lock area
2) Then we printing the blue area by using 2 nested loops.

3) Further we added spaces and the same design as in step 1.

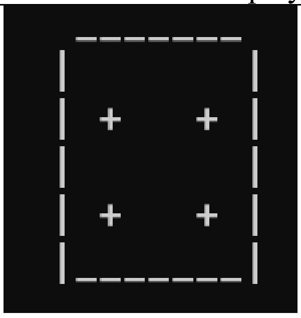
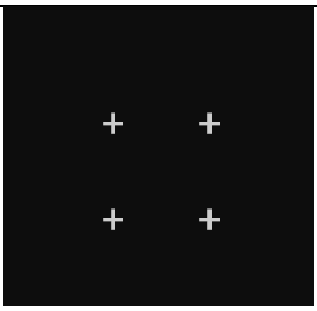
(ii) From line 8 to line 10

1) We first printed the red area by using 2 nested loops.
2) Then we printed the home count (hard-coded)
3) Then we printed the yellow area by using 2 nested loops.

(iii) From line 11 to 17

It is same as done in from line 1 to 7 except that in this section we are printing the green area and displaying the home lock area of green and yellow.

For making the game more interactive for players, we added a square around the home lock region. It does not appears, until it is the turn of that particular player. For doing this, we are sending the colour code of the player.

 Player's turn	 Without player's turn

Code for display function

```
    void display(int turn)
{
    printf(" ");
    for(i=0;i<32;i++)
    {
        printf("_"); //top line
    }
    printf(" \n");
    for(i=0;i<6;i++) //blue area
    {
        printf("| "); //for each line

        if(turn==2)
        {
            if(i==1 || i==3)
            {
                printf(" |       |  ");
            }

            if(i==0)
            {
                printf("  _______   ");
            }

            if(i==2) //homelock of red
            {
                printf(" | %c   %c |  ", *(homelock_ptr+4),*(homelock_ptr+4+1));
            }
            else if(i==4) //homelock of red
            {
                printf(" | %c   %c |  ", *(homelock_ptr+4+2),*(homelock_ptr+4+3));
            }

            else if(i==5)
            {
                printf(" |_______|  ");
            }
        }

        else
        {
            for(j=0;j<12 && i!=2 && i!=4;j++)//spaces
            {
                printf(" ");
            }

            if(i==2) //homelock of red
            {
                printf("   %c   %c    ", *(homelock_ptr+4),*(homelock_ptr+4+1));
            }
            else if(i==4) //homelock of red
            {
                printf("   %c   %c    ", *(homelock_ptr+4+2),*(homelock_ptr+4+3));
            }
        }

        for(j=0;j<3;j++) //displaying blue area
        {
            printf("%c ", *(bl_ptr+j+i*3));
        }

        if(turn==3)
        {
            if(i==1 || i==3)
            {
                printf("  |       | ");
            }

            if(i==0)
            {
                printf("   _______  ");
            }

            if(i==2) //homelock of red
            {
                printf("  | %c   %c | ", *(homelock_ptr+8),*(homelock_ptr+8+1));
            }
            else if(i==4) //homelock of red
            {
                printf("  | %c   %c | ", *(homelock_ptr+8+2),*(homelock_ptr+8+3));
            }

            else if(i==5)
            {
                printf("  |_______| ");
            }
        }
        else
        {
            for(j=0;j<12 && i!=2 && i!=4;j++)
            {
                printf(" "); //spaces
            }

            if(i==2) //homlock of blue
            {
                printf("    %c   %c   ", *(homelock_ptr+8),*(homelock_ptr+8+1));
            }
            else if(i==4) //homlock of blue
            {
                printf("    %c   %c   ", *(homelock_ptr+8+2),*(homelock_ptr+8+3));
            }
        }

        printf(" |");

        printf("\n");
    }

    for(i=0;i<3;i++) //red and blue area
    {
        printf("| ");
        for(j=0;j<6;j++) //red area
        {
            printf("%c ", *(re_ptr+j+i*6));
        }

        if(i==0) //homecount (won token) count of blue
        {
            printf("  %d   ", homecount[2]);
        }

        else if(i==1) //homecount (won token) count of red and yellow
        {
            printf("%d   %d ", homecount[1], homecount[3]);
        }

        else //homecount (won token) count of green
        {
            printf("  %d   ", homecount[0]);
        }

        for(j=0;j<6;j++) //yellow area
        {
            printf("%c ", *(ye_ptr+j+i*6));
        }
        printf(" |");
        printf("\n");
    }

    for(i=0;i<6;i++) //green
    {
        printf("| ");

        if(turn==1)
        {
            if(i==1 || i==3)
            {
                printf(" |       |  ");
            }

            if(i==0)
            {
                printf("  _______   ");
            }

            if(i==2) //homelock of red
            {
                printf(" | %c   %c |  ", *(homelock_ptr),*(homelock_ptr+1));
            }
            else if(i==4) //homelock of red
            {
                printf(" | %c   %c |  ", *(homelock_ptr+2),*(homelock_ptr+3));
            }

            else if(i==5)
            {
                printf(" |_______|  ");
            }
        }

        else
        {
            for(j=0;j<12 && i!=2 && i!=4;j++) //spaces
            {
                printf(" ");
            }

            if(i==2) //green homelock area
            {
                printf("   %c   %c    ", *(homelock_ptr),*(homelock_ptr+1));
//green home
            }
            else if(i==4) //green homelock area
            {
                printf("   %c   %c    ", *(homelock_ptr+2),*(homelock_ptr+3));
            }
        }

        for(j=0;j<3;j++) //green area
        {
            printf("%c ", *(gr_ptr+j+i*3));
        }

        if(turn==4)
        {
            if(i==1 || i==3)
            {
                printf("  |       | ");
            }

            if(i==0)
            {
                printf("   _______  ");
            }

            if(i==2) //homelock of red
            {
                printf("  | %c   %c | ", *(homelock_ptr+12),*(homelock_ptr+12+1));
            }
            else if(i==4) //homelock of red
            {
                printf("  | %c   %c | ", *(homelock_ptr+12+2),*(home-
lock_ptr+12+3));
            }

            else if(i==5)
            {
                printf("  |_______| ");
            }
        }
        else
        {
            for(j=0;j<12 && i!=2 && i!=4;j++)
            {
                printf(" "); //spaces
            }

            if(i==2) //homlock of blue
            {
                printf("    %c    %c    ", *(homelock_ptr+12),*(homelock_ptr+12+1));
            }
            else if(i==4) //homlock of blue
            {
                printf("    %c    %c    ", *(homelock_ptr+12+2),*(home-
lock_ptr+12+3));
            }
        }

        printf(" |");
        printf("\n");
    }

    printf("|");
    for(i=0;i<32;i++)
    {
        printf("_");
    }
    printf("|");
    printf(" \n");
}
```

Future Improvements and Enhancement

- Multiplayer with kills on.
- Option for number of players (2,3,4)
- Sleep function so that the movement of token is notable to the user.
- Play against CPU

Credits & References

- Credits for Ludo Board map, istockphoto – " https://www.istockphoto.com/vector/vector-image-with-ludo-board-game-gm1289161343-384918308"
- Bool Data Type: "Use of bool in C - GeeksforGeeks"
- Random Function: "I need to generate random numbers in C - Stack Overflow"