

**A**

**Project report**

**On**

Computer Graphics & Gaming Lab

**By**

**Student :Alfiya Javed Sayyed**

**SRN No : 202202232**

**Roll no:58**

**Div : F**

**Batch:F3**

**Second Year Engineering**

**Department of Computer Engineering**

**Science & Technology**

**Vishwakarma University**

Kondhwa Budruk, Pune - 411 048. Maharashtra, India.

**Academic Year 2023-24**

Lab Cie Project

**Project Report:**

**Making a Ludo game using graphics.h.**

**Project Logic :**

The C++ code presents a rendition of the classic Ludo board game, leveraging the graphics.h library to render a visually engaging interface. It establishes structures to manage game pieces and players, facilitating the tracking of positions, statuses, and colors. Through functions like `path\_init()` and `play()`, the game initializes the board layout, allocates paths for game pieces, and orchestrates the core gameplay. The `play()` function handles player turns, dice rolls, and piece movements, while also overseeing victory conditions and updating the game state accordingly. Further enhancing the visual experience, the code incorporates utility functions for various graphical operations, such as drawing shapes and executing flood fills. In essence, the implementation seamlessly blends game mechanics with graphical rendering to deliver an immersive and enjoyable rendition of Ludo in the C++ environment.

**Explanation of Algorithms :**

1. Movement Algorithm: The movement() function manages the movement of game pieces on the board. It calculates the new position of a piece based on the dice roll and updates the graphical representation of the board accordingly. This algorithm ensures that pieces move along the designated paths while accounting for special rules such as capturing opponent pieces and reaching home safely.
2. Winning Condition Check: Within the play() function, there's an algorithm to check for winning conditions after each move. It verifies whether a player has successfully moved all their pieces to the home area, indicating victory. This algorithm continuously monitors the game state to determine if a player has achieved the winning objective, thus concluding the game when necessary.
3. Path Initialization: The path\_init() function initializes the paths for game pieces on the board. This algorithm sets up the predefined routes that pieces will follow during the game, ensuring that they traverse the board correctly and encounter obstacles or special areas as intended.
4. Home Assignment: In the home\_assgn() function, an algorithm assigns home positions for each player's game pieces. This process involves determining the starting positions within the home area for each player's pieces, ensuring fairness and consistency throughout the game.
5. Dice Roll Generation: While not explicitly outlined in the provided code, the game likely employs an algorithm to generate random dice rolls. This algorithm ensures unpredictability in gameplay by producing random numbers within the desired range (typically 1 to 6) to simulate the rolling of a die.

**Code Explanation :**

This C code aims to create a graphical representation of the Charminar monument using the BGI (Borland Graphics Interface) library. Let's break down the code and explain its components:

1. Structures:
   * The code defines two structures: goti and user.
   * The goti structure represents a game piece, storing information such as its position on the board, status (active or inactive), color, and sum.
   * The user structure represents a player, containing an array of goti structures for the player's game pieces, along with other player-related information such as default position, color, and win count.
2. Initialization and Graphics:
   * The layout() function initializes the graphical representation of the game board using the graphics.h library.
   * It draws the board layout, including the main board area, player starting areas, home areas, and special zones like stars and circles.
3. Game Logic:
   * Path Initialization: The path\_init() function sets up the paths for game pieces on the board, ensuring that pieces move along predefined routes.
   * Home Assignment: The home\_assgn() function assigns home positions for each player's game pieces, determining their starting positions within the home area.
   * Gameplay: The play() function manages the main game logic, including player turns, dice rolls, and piece movements.
   * Winning Condition Check: After each move, the code checks for winning conditions to determine if a player has won the game by moving all their pieces to the home area.
4. Utility Functions:
   * Various utility functions handle graphical operations, such as drawing stars, circles, and flood fills.
   * These functions enhance the visual representation of the game board and help in updating the graphical display to reflect changes in the game state.
5. Other Algorithms:
   * The code likely includes algorithms for generating random dice rolls, although it's not explicitly shown.
   * These algorithms ensure unpredictability in gameplay and contribute to the dynamic nature of the game.

In summary, the code implements the game mechanics of Ludo in C++ by combining graphical rendering with algorithmic logic for player turns, piece movements, and victory conditions. It provides a comprehensive framework for playing the game and offers a visually appealing interface for an enjoyable gaming experience.

**Source Code:-**

#include<graphics.h>

#include<iostream>

#include<cstdio>

using namespace std;

struct goti

{

int pos;

int status;

char cllr;

int sum;

};

struct user

{

struct goti g[4];

int def;

int color;

int win;

};

int color\_g[4]={4,2,14,1};

int ary[2][52]={0};

int diamond(int);

int home(int,int,int);

int red\_hm[2][6];

int grn\_hm[2][6];

int blu\_hm[2][6];

int ylw\_hm[2][6];

void home\_assgn(void);

void layout(void);

int movement(int,int ,int );

void play(void);

void star(int x,int y)

{

line(x-7,y-7,x,y+10);

line(x,y+10,x+7,y-7);

line(x+7,y-7,x-7,y-7);

line(x-7,y+7,x+7,y+7);

line(x+7,y+7,x,y-10);

line(x,y-10,x-7,y+7);

}

void flood\_fill(int x,int y,int newColor,int oldColor)

{

int c;

c=getpixel(x,y);

if(c==oldColor)

{

setcolor(newColor);

putpixel (x,y,newColor);

flood\_fill(x+1,y,newColor,oldColor);

flood\_fill(x,y+1,newColor,oldColor);

flood\_fill(x-1,y,newColor,oldColor);

flood\_fill(x,y-1,newColor,oldColor);

}

}

void hor(int x,int y,int c)

{

int i=0;

for(i=0;i<5;i++)

{

flood\_fill(x+i\*30,y,c,0);

}

}

void ver(int x,int y,int c)

{

int i=0;

for(i=0;i<5;i++)

{

flood\_fill(x,y+i\*30,c,0);

}

}

void cir(int x,int y,int c)

{

int r=13;

circle(x+28,y+28,r);

circle(x+28,y+28+55,r);

circle(x+28+55,y+28,r);

circle(x+28+55,y+28+55,r);

flood\_fill(x+28,y+28,c,0);

flood\_fill(x+28,y+28+55,c,0);int ary[2][15]={0};

flood\_fill(x+28+55,y+28,c,0);

flood\_fill(x+28+55,y+28+55,c,0);

}

void path\_init()

{

int i=0,j,k=0;

for (j =0;j<53;j++)

ary[2][j]=0;

for( j=0;j<=15;j++)

{

if(j==6)

{

int m=0,p=0;

for(int l=6;l<19;l++)

{

if(l<=11)

{

ary[i][l]=215;

ary[i+1][l]=185-m\*30;

ary[i][l+26]=275;

ary[i+1][l+26]=305+m\*30;

m++;

}

else if(l==12)

{

ary[i][l]=245;

ary[i+1][l]=35;

ary[i][38]=245;

ary[i+1][38]=455;

ary[i][51]=35;

ary[i+1][51]=245;

}

else

{

ary[i][l]=275;

ary[i+1][l]=35+p\*30;

ary[i][l+26]=215;

ary[i+1][l+26]=455-p\*30;

p++;

}

}

j+=2;

k+=13;

}

else

{

if(j==15)

{

ary[i][j+10]=455;

ary[i+1][j+10]=245;

}

else

{

ary[i][k]=35+j\*30;

ary[i+1][k] = 215;

ary[i][50-k]=35+j\*30;

ary[i+1][50-k]=275;

k++;

}

}

}

}

int main()

{

int gd=DETECT;

int gm;

struct user student[4];

initgraph(&gd,&gm,"");

layout();

path\_init();

home\_assgn();

play();

getch();

closegraph();

}

void layout(void)

{

int i=0,x,y,j;

setcolor(12);

rectangle(20,20,470,470);

line(200,20,200,470);

line(230,20,230,200);

line(230,290,230,470);

line(290,230,470,230);

line(260,20,260,200);

line(260,290,260,470);

line(290,20,290,470);

line(20,200,470,200);

line(20,230,200,230);

line(290,230,470,230);

line(20,260,200,260);

line(290,260,470,260);

line(20,290,470,290);

line(200,200,290,290);

line(200,290,290,200);

star(65,215);

star(215,95);

star(275,65);

star(395,215);

star(425,275);

star(275,395);

star(215,425);

star(95,275);

rectangle(50,50,170,170);

rectangle(55,55,165,165);

rectangle(320,50,440,170);

rectangle(325,55,435,165);

rectangle(50,320,170,440);

rectangle(55,325,165,435);

rectangle(320,320,440,440);

rectangle(325,325,435,435);

for(i=0;i<15;i++)

{

if(i>=6&&i<=9)

continue;

line(20+i\*30,200,20+i\*30,290);

}

for(i=0;i<15;i++)

{

if(i>=6&&i<=9)

continue;

line(200,20+i\*30,290,20+i\*30);

}

x=55,y=55;

cir(x,y,4);

cir(x+270,y,2);

cir(x,y+270,1);

cir(x+270,y+270,14);

flood\_fill(235,245,4,0);

flood\_fill(245,235,2,0);

flood\_fill(245,275,1,0);

flood\_fill(275,245,14,0);

flood\_fill(21,21,4,0);

flood\_fill(291,21,2,0);

flood\_fill(21,291,1,0);

flood\_fill(291,291,14,0);

flood\_fill(56,56,15,0);

flood\_fill(56,326,15,0);

flood\_fill(326,56,15,0);

flood\_fill(326,326,15,0);

flood\_fill(51,201,11,0);

flood\_fill(201,81,11,0);

flood\_fill(261,51,11,0);

flood\_fill(381,201,11,0);

flood\_fill(411,261,11,0);

flood\_fill(261,381,11,0);

flood\_fill(201,411,11,0);

flood\_fill(81,261,11,0);

hor(55,245,4);

hor(305,245,14);

ver(245,55,2);

ver(245,305,1);

setcolor(4);

}

int movement(int first,int last,int clr)

{

int num=6,temp=0,q,rx,ry,prev,prs=0,t1,t2;

//update play

// move the graph

// end of function

// new turn to player

for(q=first;q<=last;q++)

{

if(q==52)

{

last=last-52;

movement(0,last,clr);

q=0;

return 0;

}

prs=getpixel(ary[0][q],ary[1][q]);

setcolor(clr);

rx=ary[0][q];

ry=ary[1][q];

delay(200);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,0);

delay(200);

if(prs!=clr||q==first){

setcolor(0);

circle(rx,ry,12);

flood\_fill(rx,ry,0,clr);

}

}

setcolor(clr);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,0);

}

void play()

{

struct user \*student;

int n,k=1;

int num;

printf("\n HOW MANY PLAYERS : ");

scanf("%d",&n);

student = (struct user \*)malloc(n\*sizeof(struct user));

// intialise the users

for(int i =0;i< n;i++)

{

for(int j=0;j<4;j++){

student[i].g[j].pos=-1;

student[i].g[j].sum=0;

student[i].g[j].status=1;

}

student[i].win=0;

student[i].def=k;

k+=13;

}

//gameplaye and movement

int i=0;

printf ("GAME STARTED ");

while(1)

{

printf ("\nplayer-%d ::",i);

int num,run,first,getcl,safejone;

//generate random number and get it in pos

// and give again chance to same user

//num

// find which one user want to play in run

//give the extra conditionsssss

while(1)

{

num = (rand() % 6 ) +1;

printf("\tDICE : %d :",num);

printf("\n\tTO MOVE : (0,1,2,3) ::");

scanf("%d",&run); // which goti to move

if( student[i].g[run].status==0)

{

cout<<"it already reached home "<<endl;

break;

}

if(num == 6)

{

if(student[i].g[run].pos== -1)

{

student[i].g[run].pos = student[i].def;

// take out from home

student[i].g[run].sum=0;

}

else{

student[i].g[run].sum+=num;

first = student[i].g[run].pos;

getcl=getpixel(ary[0][first+num],ary[1][first+num]);

if(getcl != 0 && getcl != color\_g[i] && first+num != 1 && first+num != 9 && first+num != 14 && first+num != 22 && first+num != 27 && first+num != 35 && first+num != 40 && first+num !=48 )

{ // eat the other player

int l,m;

for( l=0;l<n;l++) //get the player

{

if(student[l].color == getcl)

break;

}

printf(" :: m %d :",m);

for( m=0 ; m<4;m++) // get the particular player

{ if( student[l].g[m].pos == first+num)

break;

}

student[l].g[m].pos =-1; // intial stage set

printf("goti number %d %d",m,l);

flood\_fill(ary[0][first+num],ary[1][first+num],0,getcl);

}

printf("sum :%d\n",student[i].g[run].sum);

if(student[i].g[run].sum > 50)

{

student[i].g[run].pos=home(color\_g[i],first,first+num);

if( student[i].g[run].pos == 6)

{cout<<"\t\t\tFINISHED"<<endl; student[i].g[run].status=0;

student[i].win+=1;

}

break;

}

movement(first,(first+num),color\_g[i]); //yaha temp changes %52.

student[i].g[run].pos = (student[i].g[run].pos+num)%52;

}

continue;

}

else{

if(student[i].g[run].pos==-1){

printf(" \n\t\t::CANNOT MOVE ::");

break;}

student[i].g[run].sum+=num;

first = student[i].g[run].pos;

getcl=getpixel(ary[0][first+num],ary[1][first+num]);

if(getcl != 0 && getcl != color\_g[i] && first+num != 1 && first+num != 9 && first+num != 14 && first+num != 22 && first+num != 27 && first+num != 35 && first+num != 40 && first+num !=48 )

{

int l=0,m=0;

for( l=0;l<n;l++)

{ if(color\_g[l] == getcl)

break;

}

for( m=0 ; m<4;m++)

{ if( student[l].g[m].pos == first+num)

break;

}

// printf("goti number %d %d",m,l);

student[l].g[m].pos =-1;

flood\_fill(ary[0][first+num],ary[1][first+num],0,getcl);

}

// printf("sum :%d\n",student[i].g[run].sum);

if( student[i].g[run].sum >50)

{ printf(" \t \t HOME TIME ");

student[i].g[run].pos= home(color\_g[i],first,first+num);

if( student[i].g[run].pos == 6)

{cout<<"\t\t\t FINISHED"<<endl; student[i].g[run].status=0;

student[i].win+=1;

if(student[i].win==4)

{ cout<<"\t\t\t\t\*\*\*\*\*\*\*WINNER IS :"<<i<<" \*\*\*\*\*\*"<<endl;

return;

}

}

break;

}

movement(first,(first+num),color\_g[i]);

student[i].g[run].pos=(student[i].g[run].pos+num)%52;

break;

}

}

i=(i+1)%n;

}

}

int diamond(int x)

{

int a;

if(x == 4)

a = 51;

else if( x == 2)

a = 12;

else if(x == 14)

a = 25;

else if (x == 1)

a == 38;

return a;

}

void home\_assgn()

{

for(int i=0;i<6;i++)

{

red\_hm[0][i]=65+30\*i;

red\_hm[1][i]=245;

}

for(int i=0;i<6;i++)

{

grn\_hm[0][i]=245;

grn\_hm[1][i]=65+30\*i;

}

for(int i=0;i<6;i++)

{

blu\_hm[0][i]=245;

blu\_hm[1][i]=425-30\*i;

}

for(int i=0;i<6;i++)

{

ylw\_hm[0][i]=425-30\*i;

ylw\_hm[1][i]=245;

}

}

int home(int clr,int first,int pos)

{

int ab,remn,tmp,rx,ry;

int bdb;

tmp=diamond(clr);

if(first > 5)

{movement(first,tmp,clr);

bdb=0;

remn=pos-tmp;

flood\_fill(ary[0][tmp],ary[1][tmp],0,clr);

}

else{

if(pos > 6)

{

return first;

}

bdb=first;

remn=pos;

}

//printf(" bdb %d remn %d",bdb,remn);

switch(clr)

{

case 1: if(bdb > 0)

{

setcolor(1);

rx=blu\_hm[0][bdb-1];

ry=blu\_hm[1][bdb-1];

circle(rx,ry,12);

flood\_fill(rx,ry,1,3);

}

for(ab=bdb;ab<remn;ab++)

{ clr=3;

setcolor(clr);

rx=blu\_hm[0][ab];

ry=blu\_hm[1][ab];

delay(200);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,1);

delay(200);

setcolor(1);

circle(rx,ry,12);

flood\_fill(rx,ry,1,clr);

}

setcolor(clr);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,1);

break;

case 2: if(bdb > 0)

{

setcolor(2);

rx=grn\_hm[0][bdb-1];

ry=grn\_hm[1][bdb-1];

circle(rx,ry,12);

flood\_fill(rx,ry,2,3);

}

for(ab=bdb;ab<remn;ab++)

{ clr=3;

setcolor(clr);

rx=grn\_hm[0][ab];

ry=grn\_hm[1][ab];

delay(200);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,2);

delay(200);

setcolor(2);

circle(rx,ry,12);

flood\_fill(rx,ry,2,clr);

}

setcolor(clr);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,2);

break;

case 4: if(bdb > 0)

{

setcolor(4);

rx=red\_hm[0][bdb-1];

ry=red\_hm[1][bdb-1];

circle(rx,ry,12);

flood\_fill(rx,ry,4,3);

}

for(ab=bdb;ab<remn;ab++)

{ clr=3;

setcolor(clr);

rx=red\_hm[0][ab];

ry=red\_hm[1][ab];

delay(200);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,4);

delay(200);

setcolor(4);

circle(rx,ry,12);

flood\_fill(rx,ry,4,clr);

}

setcolor(clr);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,4);

break;

case 14: if(bdb > 0)

{

setcolor(14);

rx=ylw\_hm[0][bdb-1];

ry=ylw\_hm[1][bdb-1];

circle(rx,ry,12);

flood\_fill(rx,ry,14,3);

}

for(ab=bdb;ab<remn;ab++)

{ clr=3;

setcolor(clr);

rx=ylw\_hm[0][ab];

ry=ylw\_hm[1][ab];

delay(200);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,14);

delay(200);

setcolor(14);

circle(rx,ry,12);

flood\_fill(rx,ry,14,clr);

}

setcolor(clr);

circle(rx,ry,12);

flood\_fill(rx,ry,clr,14);

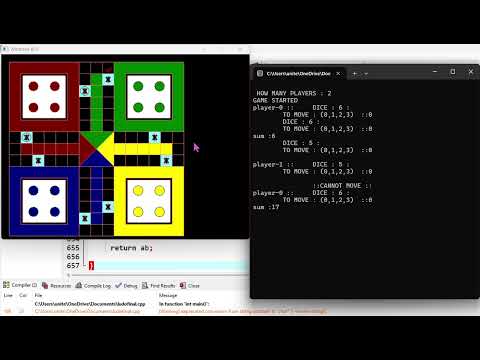
break;

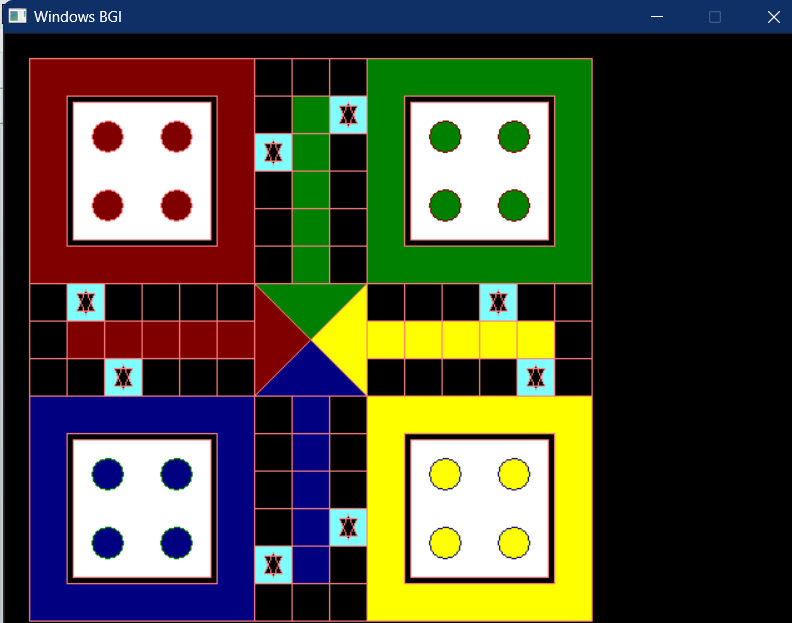
}

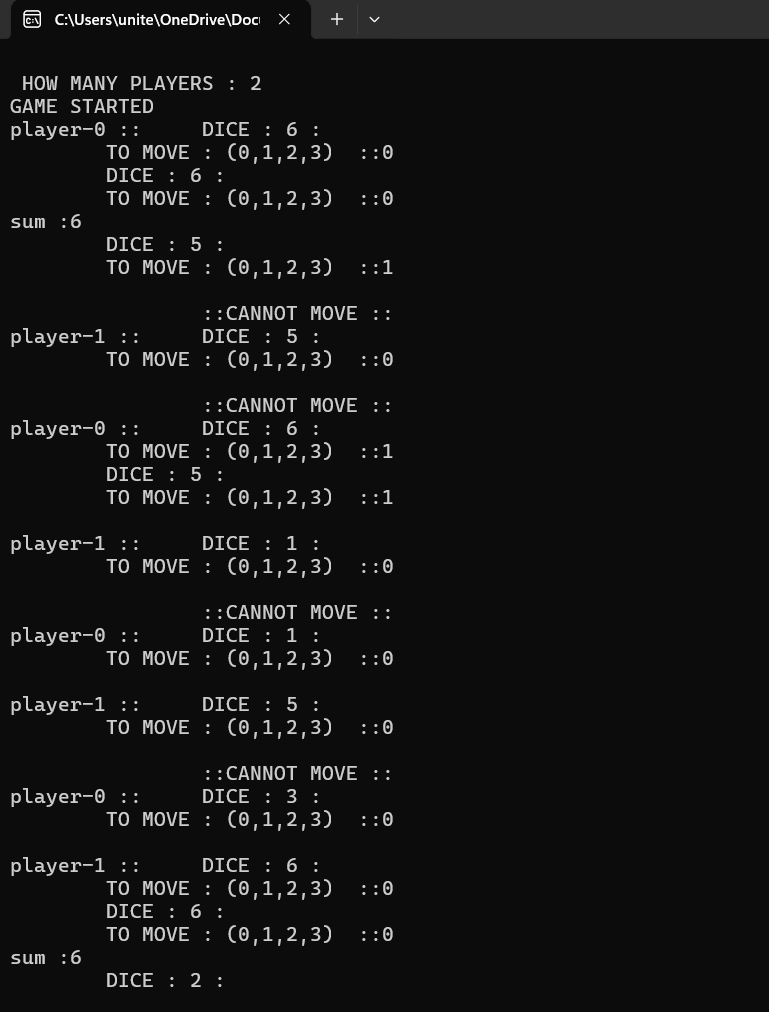
return ab;

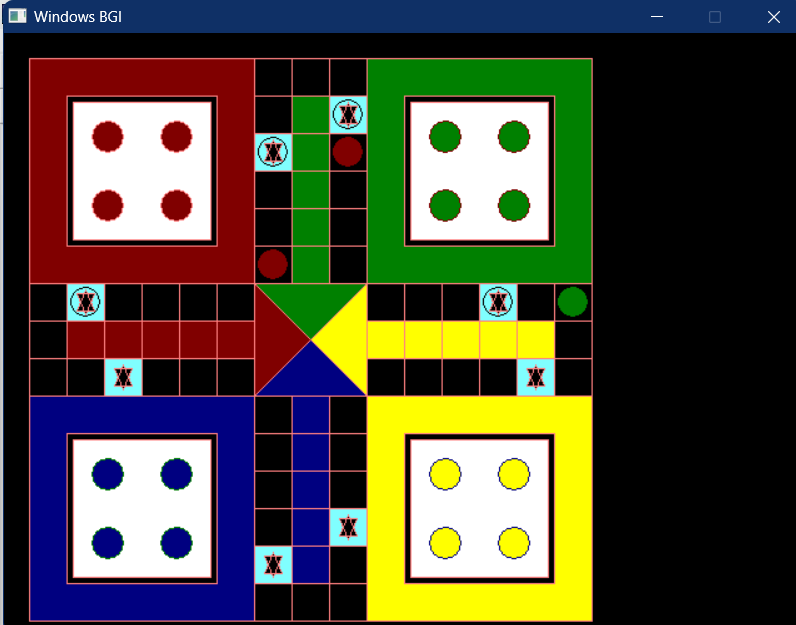
}

**Output:**

**[](https://www.youtube.com/embed/Dge9EdEs180?feature=oembed)**

****





**Conclusion :**

This project successfully offers a functional implementation of the classic Ludo board game, integrating graphical rendering with algorithmic logic to create an engaging gaming experience. Through structures, functions, and utility algorithms, the code manages player turns, piece movements, and victory conditions while maintaining an interactive graphical display of the game board. Despite not being accompanied by detailed comments or documentation, the code effectively encapsulates the essential elements of Ludo gameplay, including path initialization, home assignment, and gameplay mechanics. With further refinement, documentation, and perhaps optimization, this code could serve as a solid foundation for building more sophisticated Ludo game applications or as a learning resource for understanding game development concepts in C++.