ABSTRACT

Brick Breaker (platformer) is a Breakout clone which the player must smash a wall of bricks by deflecting a bouncing ball with a paddle. The paddle may move horizontally and is controlled with the BlackBerry's trackwheel, the computer's mouse or the touch of a finger (in the case of touchscreen). The player gets 3 lives to start with; a life is lost if the ball hits the bottom of the screen. When all the bricks have been destroyed, the player advances to a new, harder level. There are 34 levels. Many levels have unbreakable silver bricks. If all lives are lost, the game is over. There are many versions of brick breaker, some in which you can shoot flaming fireballs or play with more than one ball if the player gets a power up.

ABIN B VINOD(1CE18CS003)

CONTENTS

INDEX **PAGE NO**

Chapter 1

INTRODUCTION

Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

Using this editor you can draw and paint using the mouse. It can also perform a host of other functions like drawing lines, circles, polygons and so on. Interactive picture construction techniques such as basic positioning methods, rubber-band methods, dragging and drawing are used. Block operations like cut, copy and paste are supported to edit large areas of the workspace simultaneously. It is user friendly and intuitive to use. OpenGL(open graphics library) is a standard specification defining a cross language cross platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives. OpenGL was developed by silicon graphics Inc.(SGI) in 1992 and is widely used in CAD ,virtual reality , scientific visualization , information visualization and flight simulation. It is also used in video games, where it competes with direct 3D on Microsoft Windows platforms.OpenGL is managed by the non-profit technology consortium,thekhronosgroup,Inc

OpenGL serves two main purpose :

- To hide the complexities of interfacing with different 3D accelerators, by presenting programmer with a single, uniform API
- To hide the differing capabilities of hardware platforms, by requiring that all Implementations support the full openGL, feature set.

OpenGL has historically been influential on the development of 3D accelerator, promoting a base level of functionality that is now common in consumer level hardware:

- Rasterized points, lines and polygons are basic primitives.
- A transform and lighting pipeline.
- Z buffering.
- Texture Mapping.
- Alpha
- Blending.

1.0 HISTORY & OPENGL

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube (CRT) screens soon after the introduction of computers.

Computer graphics today largely interactive, the user controls the contents, structure, and appearance of objects and of displayed images by using input devices, such as keyboard, mouse, or touch-sensitive panel on the screen. Graphics based user interfaces allow millions of new users to control simple, low-cost application programs, such as spreadsheets, word processors, and drawing programs.

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms (see Direct3D vs. OpenGL). OpenGL is managed by the non-profit technology consortium, the Khronos Group.

1.1 APPLICATION

Brick Breaker (platformer) is a Breakout clone which the player must smash a wall of bricks by deflecting a bouncing ball with a paddle. The paddle may move horizontally and is controlled with the BlackBerry's trackwheel, the computer's mouse or the touch of a finger (in the case of touchscreen). The player gets 3 lives to start with; a life is lost if the ball hits the bottom of the screen. When all the bricks have been destroyed, the player advances to a new, harder level. There are 34 levels. Many levels have unbreakable silver bricks. If all lives are lost, the game is over. There are many versions of brick breaker, some in which you can shoot flaming fireballs or play with more than one ball if the player gets a power up.

Chapter 2

SYSTEM REQUIREMENTS SPECIFICATION

2.1 HARDWARE REQUIREMENTS

Minimum hardware specification

- Microprocessor: **1.0 GHz** and above CPU based on either AMD or INTEL Microprocessor Architecture
- Main memory : **512 MB RAM**
- Hard disk speed in RPM:**5400 RPM**
- Keyboard: **QWERTY** Keyboard
- Mouse :**2 or 3** Button mouse
- Monitor : **1024 x 768** display resolution

2.2 SOFTWARE REQUIREMENTS

Minimum software specification

- Operating system : UBUNTU OR WINDOWS
- Tool Used : CodeBlocks
- OPENGL Library
- X86
- X64(WOW)
- Mouse Driver
- Graphics Driver
- C Language

Chapter 3

3 Low level design & flowchart

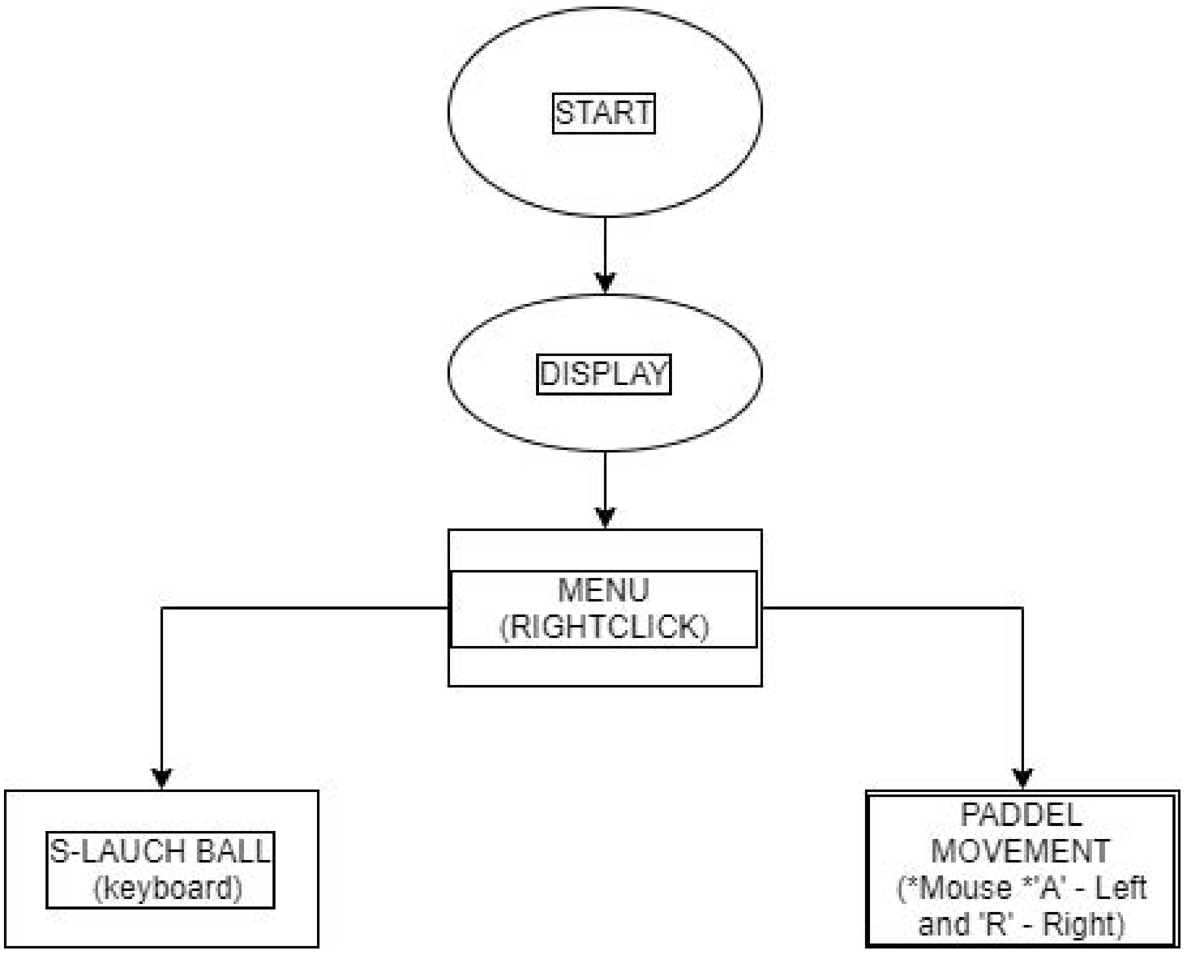


FIGURE 1.0

Chapter 4

IMPLEMENTATION

4. Functions

The **glColor3f (float, float, float) :-** This function will set the current drawing color

gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top):- which defines a two dimensional viewing rectangle in the plane z=0.

glClear( ):-Takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

glClearColor ():-Specifies the red, green, blue, and alpha values used by **glClear** to clear the color buffers.

GlLoadIdentity( ):-the current matrix with the identity matrix.

glMatrixMode(mode):-Sets the current matrix mode, mode can be GL_MODELVIEW, GL_PROJECTION or GL_TEXTURE.

Void glutInit (int *argc, charargv):-**Initializes GLUT, the arguments from main are passed in and can be used by the application.

Void glutInitDisplayMode (unsigned int mode):-Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model and buffering.

Function to draw the spherical ball
void draw_ball()

glViewport :specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let x nd y nd be normalized device coordinates

MAIN FUNCTION USED

//The main display function

```
void display (void)
{	glClearColor (0.0,0.0,0.0,1.0);
	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();
	gluLookAt(0,0,0,0,0,-25,0,1,0);
	glTranslatef(0,0,-25);
	draw_paddle();
	draw_bricks();
	draw_ball();
	text(score);
	glutSwapBuffers();}
```

// Function to draw the grid of bricks

void draw_bricks()

//Function to draw a single brick

void brick(GLfloat x,GLfloat y, GLfloat z)

// Function to draw the paddle

void draw_paddle()

//Function to draw the spherical ball

void draw_ball()

//Function to print the score on the screen

void text(int sc)

//mouse function

void mousemotion(int x,int y)

//handle brick color

void change_brick_color(int action)

//handle ball color

void change_ball_color(int action)

//handle menu

void handle_menu(int action)

//function to turn on lights

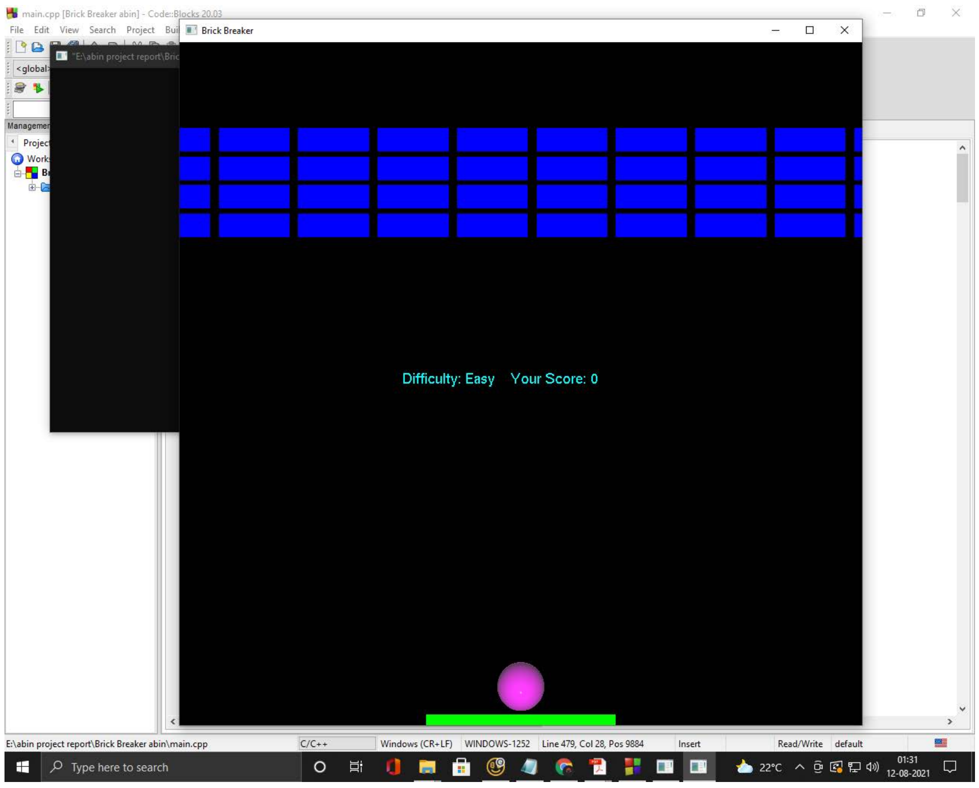
void lightsOn()

//function to take in keyboard entries

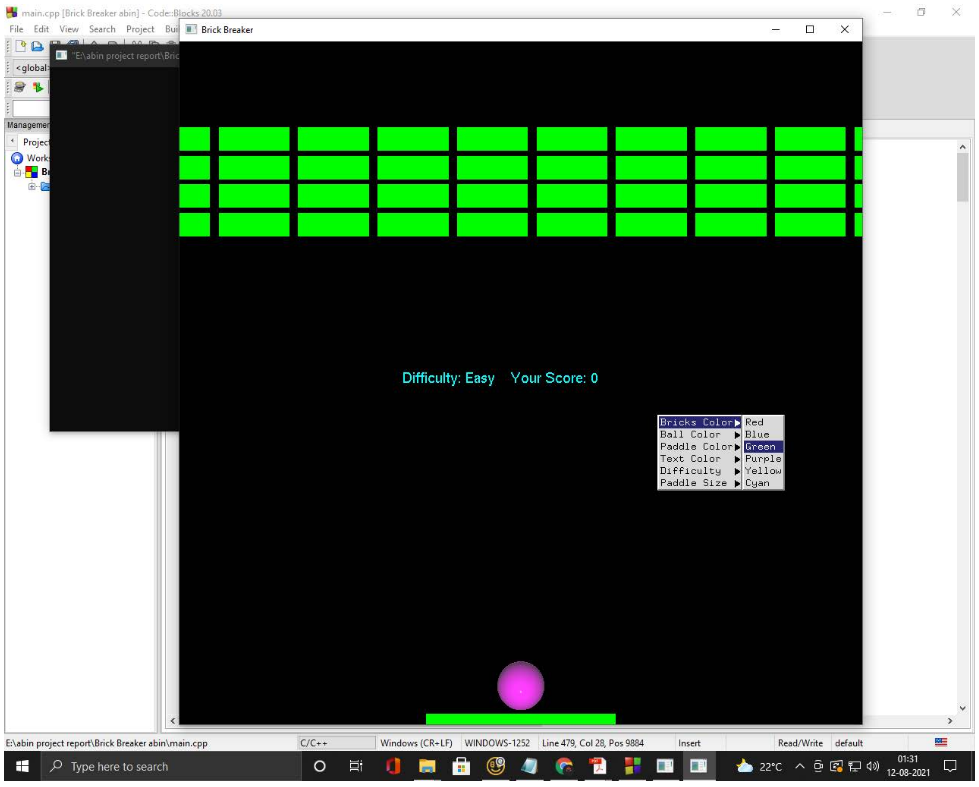
void keyboard (unsigned char key, int x, int y)

Chapter 5

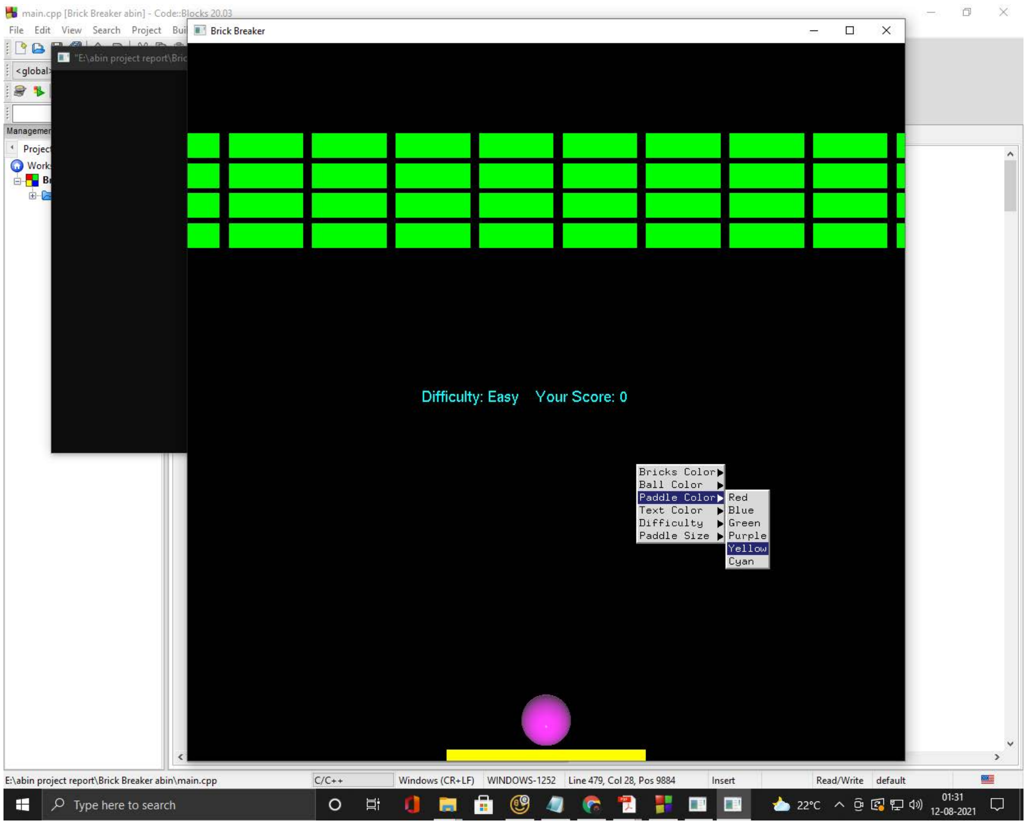
5.0 RESULTS & SNAPSHOTS



5.1: GAME START

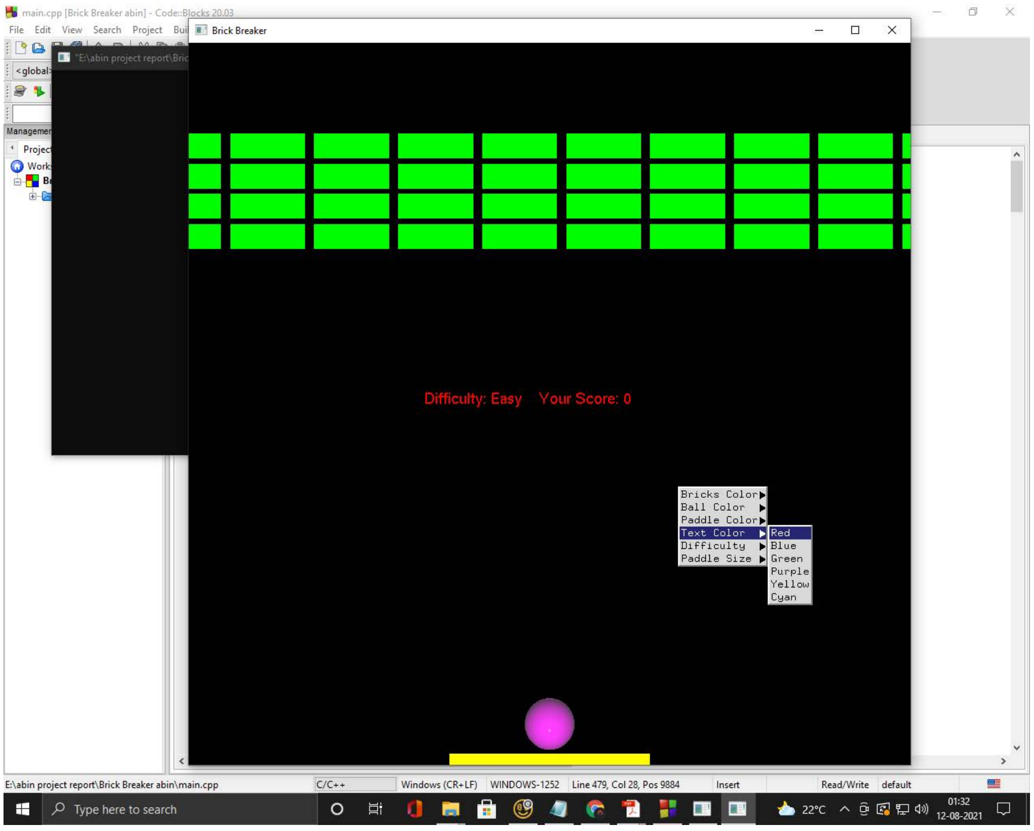


5.2: GAME MENU



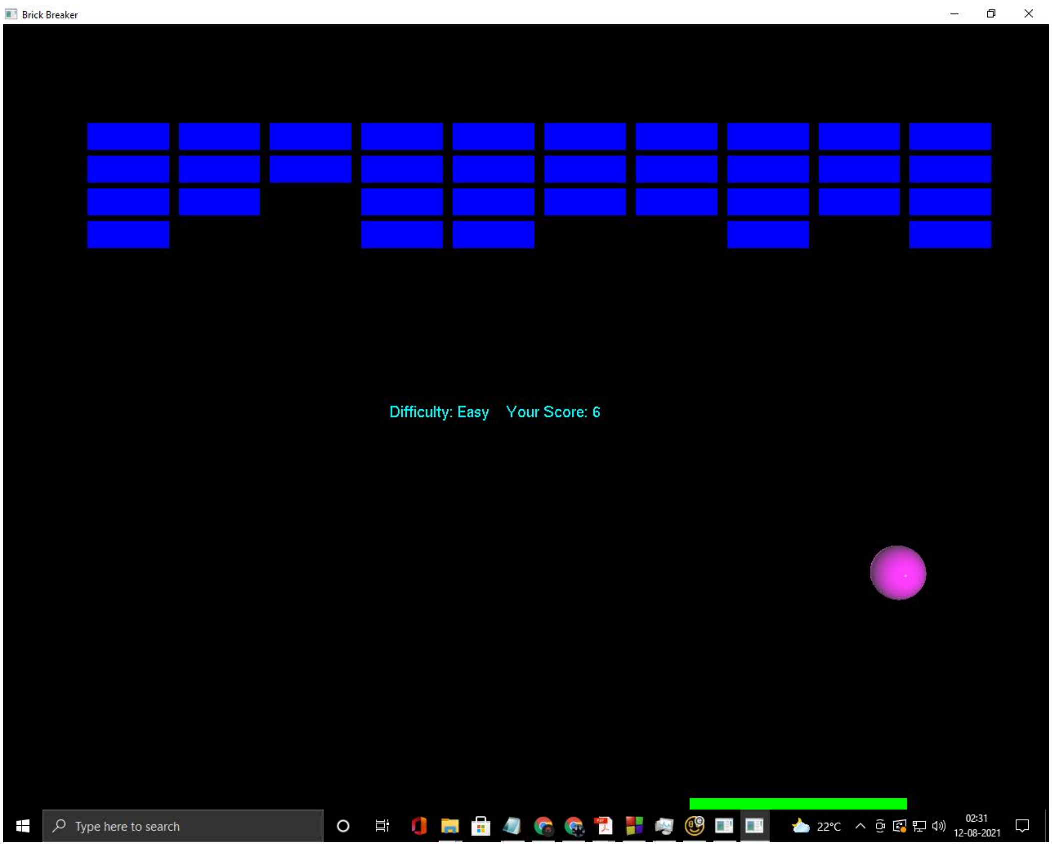
```
Difficulty: Easy   Your Score: 0
```

5.3: PADDLE COLOR CHANGE MENU

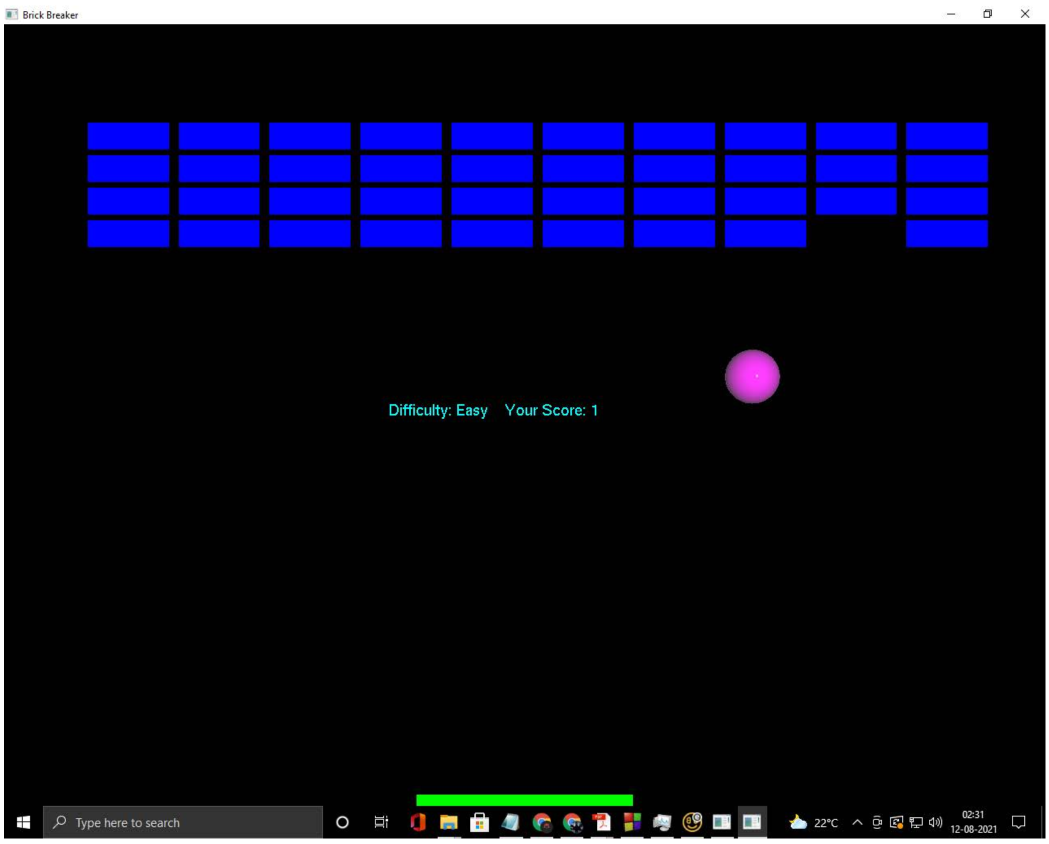


```
Difficulty: Easy   Your Score: 0
```

5.4:TEXT MENU



5.5: GAMEPLAY



5.6: GAMEPLAY

6.0 APPENDIX

```
#include <stdlib.h>
#include <GL/glut.h>
#include<stdio.h>
#include<string.h>
#include <math.h>
//#include<iostream>
//using namespace std;


int score = 0;
int brick_color = 1,ball_color = 3,level = 0,paddle_color = 2,text_color = 5,size = 1;;
GLfloat twoModel[]={GL_TRUE};
int game_level[] = {7,5,2};
float rate = game_level[level];

GLfloat brick_color_array[][3] = {{1,0,0},{0,0,1},{0,1,0},{1,0,1},{1,1,0},{0,1,1}};
GLfloat paddle_color_array[][3] = {{1,0,0},{0,0,1},{0,1,0},{1,0,1},{1,1,0},{0,1,1}};
GLfloat text_color_array[][4] = {{1,0,0,1},{0,0,1,1},{0,1,0,1},{1,0,1,1},{1,1,0,1},{0,1,1,1}};
GLfloat paddle_size[] = {2,4,6};
//The grid parameters for the bricks
int rows = 4;
int columns = 10;

// Structure to store the coordinates of each brick
struct brick_coords{

	GLfloat x;
	GLfloat y;
};

//Array to store the bricks
brick_coords brick_array[50][50];
GLfloat px,bx = 0,by = -12.8 ,speed = 0,dirx=0,diry=0,start = 0;



// Function to draw the paddle
void draw_paddle()
{
	glDisable(GL_LIGHTING);
	glColor3fv(paddle_color_array[paddle_color]);
	glBegin(GL_POLYGON);
	glVertex3f(-paddle_size[size]+px,0-15,0);
	glVertex3f(paddle_size[size]+px,0-15,0);
	glVertex3f(paddle_size[size]+px,1-15,0);
	glVertex3f(-paddle_size[size]+px,1-15,0);
	glEnd();
	glEnable(GL_LIGHTING);
}
```

```
//Function to draw a single brick
void brick(GLfloat x,GLfloat y, GLfloat z)
{

        glDisable(GL_LIGHTING);
        glColor3fv(brick_color_array[brick_color]);
        glBegin(GL_QUADS);
        glVertex3f(x,y,z);
        glVertex3f(x+3,y,z);
        glVertex3f(x+3,y+1,z);
        glVertex3f(x,y+1,z);
        glEnd();
        glEnable(GL_LIGHTING);
}


// Function to draw the grid of bricks
void draw_bricks()
{

        int i,j;
        if(start == 0)
        {
                for(i = 1;i<=rows;i++)
                {
                        for(j = 1;j<=columns;j++)
                        {

                                brick_array[i][j].x = (GLfloat)(j*4*0.84);
                                brick_array[i][j].y = (GLfloat)(i*2*0.6) ;
                        }
                }
        }


        glPushMatrix();
        glTranslatef(-19.5,5,0);

        for(i = 1;i<=rows;i+=1)
        {
                for(j = 1;j<=columns;j+=1)
                {

                        if(brick_array[i][j].x==0 || brick_array[i][j].y ==0)
                        {
                                continue;
                        }
                        brick(brick_array[i][j].x,brick_array[i][j].y,0);
                }
        }
        glPopMatrix();

}
```

```
//Function to draw the spherical ball
void draw_ball()
{
            GLfloat ambient1[] = {1,1,1};
            GLfloat diffuse1[] = {0.4,0.4,0.4};
            GLfloat specular1[] = {1,1,1};

            GLfloat position[] = {0,0,-50,1};
            GLfloat ambient2[] = {0,0,0};
            GLfloat diffuse2[] = {1,1,1};
            GLfloat specular2[] = {0,1,1};
float materialColours[][3]={{1,0,0},{0,0,1},{0,1,0},{1,0,1},{1,1,0},{0,1,1}};
            GLfloat matAmbient1[] = {1,1,1};
            GLfloat matDiffuse1[] = {1,1,1};
            GLfloat matSpecular1[] = {1,1,1};
            GLfloat shininess[] = {1000};

            glLightfv(GL_LIGHT0,GL_SPECULAR,specular1);
            glLightfv(GL_LIGHT0,GL_AMBIENT,ambient1);
            glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuse1);

            glLightfv(GL_LIGHT1,GL_POSITION,position);
            glLightfv(GL_LIGHT1,GL_SPECULAR,specular2);
            glLightfv(GL_LIGHT1,GL_AMBIENT,ambient2);
            glLightfv(GL_LIGHT1,GL_DIFFUSE,diffuse2);

            glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, shininess);
            glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,materialColours[ball_color
]);

           glPushMatrix();
            glTranslatef(bx,by,0);
            glScalef(1.0, 1.0, 0.5);
            //glScalef(size[i], size[], size[]);
            glutSolidSphere(1.0, 52, 52);

            glPopMatrix();}
//mouse function
void mousemotion(int x,int y)
{

  if(start == 1)
  {
    px=(x-glutGet(GLUT_WINDOW_WIDTH)/2)/20;
    if(px>15)
           {
                   px=15;
           }
           if(px<-15)
           {
                   px=-15;
           }
    }

    else glutSetCursor(GLUT_CURSOR_INHERIT);
}
```

```
//handle brick color
void change_brick_color(int action)
{

        brick_color=action-1;
}

//handle ball color
void change_ball_color(int action)
{

        ball_color=action-1;
}

//handle level
void change_difficulty(int action)
{

        level=action-1;
}

//handle menu
void handle_menu(int action)
{

}

//handle paddle color
void change_paddle_color(int action)
{
        paddle_color = action -1;
}

//handle paddle color
void change_text_color(int action)
{
        text_color = action -1;
}

//handle paddle size
void change_paddle_size(int action)
{
        size = action -1;
}
```

```
//add menu
void addMenu()
{


	int submenu1 = glutCreateMenu(change_brick_color);
	glutAddMenuEntry("Red",1);
	glutAddMenuEntry("Blue",2);
	glutAddMenuEntry("Green",3);
	glutAddMenuEntry("Purple",4);
	glutAddMenuEntry("Yellow",5);
	glutAddMenuEntry("Cyan",6);

	int submenu2 = glutCreateMenu(change_ball_color);
	glutAddMenuEntry("Red",1);
	glutAddMenuEntry("Blue",2);
	glutAddMenuEntry("Green",3);
	glutAddMenuEntry("Purple",4);
	glutAddMenuEntry("Yellow",5);
	glutAddMenuEntry("Cyan",6);

	int submenu4 = glutCreateMenu(change_paddle_color);
	glutAddMenuEntry("Red",1);
	glutAddMenuEntry("Blue",2);
	glutAddMenuEntry("Green",3);
	glutAddMenuEntry("Purple",4);
	glutAddMenuEntry("Yellow",5);
	glutAddMenuEntry("Cyan",6);

	int submenu3 = glutCreateMenu(change_difficulty);
	glutAddMenuEntry("Easy",1);
	glutAddMenuEntry("Medium",2);
	glutAddMenuEntry("Hard",3);

	int submenu5 = glutCreateMenu(change_text_color);
	glutAddMenuEntry("Red",1);
	glutAddMenuEntry("Blue",2);
	glutAddMenuEntry("Green",3);
	glutAddMenuEntry("Purple",4);
	glutAddMenuEntry("Yellow",5);
	glutAddMenuEntry("Cyan",6);

	int submenu6 = glutCreateMenu(change_paddle_size);
	glutAddMenuEntry("Small",1);
	glutAddMenuEntry("Medium",2);
	glutAddMenuEntry("Large",3);

	glutCreateMenu(handle_menu);
	glutAddSubMenu("Bricks Color",submenu1);
	glutAddSubMenu("Ball Color",submenu2);
	glutAddSubMenu("Paddle Color",submenu4);
	glutAddSubMenu("Text Color",submenu5);
	glutAddSubMenu("Difficulty",submenu3);
	glutAddSubMenu("Paddle Size",submenu6);
	glutAttachMenu(GLUT_RIGHT_BUTTON);

}
```

```
//Function to print the score on the screen
void text( int sc)
{
        glDisable(GL_LIGHTING);
        char text[40];
        char difficulty[10];
        if(level == 0)
        {
                sprintf(difficulty,"Easy");
        }

        if(level == 1)
        {
                sprintf(difficulty,"Medium");
        }

        if(level == 2)
        {
                sprintf(difficulty,"Hard");
        }
        if(sc <40)
        sprintf(text,"Difficulty: %s    Your Score: %d",difficulty, sc);
        else
        {
          sprintf(text,"You have won !!");
          start = 0;
          by = -12.8;
          bx = 0;
          dirx = 0;
          diry = 0;
          px = 0;

        }
        // The color
        glColor4fv(text_color_array[text_color]);
        // Position of the text to be printer
        glPushMatrix();
        glTranslatef(-1,0,0);
        glRasterPos3f(0, 0, 20);
        for(int i = 0; text[i] != '\0'; i++)
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,
text[i]);
        glEnable(GL_LIGHTING);
        glPopMatrix();
}

//The main display function
void display (void) {

        glClearColor (0.0,0.0,0.0,1.0);
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(0,0,0,0,0,-25,0,1,0);
        glTranslatef(0,0,-25);
        draw_paddle();
        draw_bricks();
        draw_ball();
        text(score);
        glutSwapBuffers();
}
```

```
//function to turn on lights
void lightsOn()
{
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_LIGHT1);
}

void reshape (int w, int h) {
        glViewport (0, 0, (GLsizei)w, (GLsizei)h);
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity ();
        gluPerspective (60, (GLfloat)w / (GLfloat)h, 1.0, 1000.0);
        glMatrixMode (GL_MODELVIEW);
}


//function to take in keyboard entries
void keyboard (unsigned char key, int x, int y)
{
        switch(key)
        {
                case 'd': px+=3; break;
                case 'a': px-=3; break;
                case 'q': exit(0); break;
                case 's':
                if(!start)
                {
                        dirx = diry= 1;
                        rate = game_level[level];
                        start = 1;
                        score = 0;
                        glutSetCursor(GLUT_CURSOR_NONE);

                }
                break;
        }
        if(px>15)
        {
                px=15;
        }
        if(px<-15)
        {
                px=-15;
        }
        if(start== 0)
        {
                px=0;
        }
        glutPostRedisplay();
}
```

```
//Function to handle the case when the ball strikes the bricks
void hit()
{
        int i,j;
        for(i = 1;i<=rows;i++)
                for(j=1;j<=columns;j++)
{
        if((bx>=brick_array[i][j].x-19.5-0.1 )&&( bx<=brick_array[i][j].x + 3-19.5+ 0.1))
        {
                if(by >=brick_array[i][j].y+5-0.1 && by <=brick_array[i][j].y+5 +1.2 +0.1)
        {
                                        brick_array[i][j].x = 0;
                                        brick_array[i][j].y = 0;
                                        diry= diry*-1;
                                        score++;
        }
        //cout<<bx<<" "<<by<<"\t"<<brick_array[i][j].x<<" "<<brick_array[i][j].y;
                        }
                else if(by >=brick_array[i][j].y+5 -0.1 && by <=brick_array[i][j].y+5 +1.2+0.1)
                        {
        if((bx>=brick_array[i][j].x-19.5 -0.1)&&( bx<=brick_array[i][j].x + 3-19.5 + 0.1 ))
                                {
                                        brick_array[i][j].x = 0;
                                        brick_array[i][j].y = 0;
                                        dirx= dirx*-1;
                                        score++;
                                }
                    //cout<<bx<<" "<<by<<"\t"<<brick_array[i][j].x<<" "<<brick_array[i][j].y;
                        }
                }
}



//The idle function. Handles the motion of the ball along with rebounding from various
surfaces
void idle()
{
        hit();
        if(bx<-16 || bx>16 && start == 1)
        {
                dirx = dirx*-1;
        }
        if(by<-15 || by>14 && start == 1)
        {
                diry = diry*-1;
        }
        bx+=dirx/(rate);
        by+=diry/(rate);
        rate-=0.001; // Rate at which the speed of ball increases

        float x = paddle_size[size];
//Make changes here for the different position of ball after rebounded by paddle
            if( by<=-12.8 && bx<(px+x*2/3) && bx>(px+x/3)&& start == 1 )
            {
                        dirx = 1;
                        diry = 1;
            }
            else if(by <=-12.8 && bx<(px-x/3) && bx>(px-x*2/3) && start == 1 )
            {
                        dirx = -1;
                        diry = 1;
            }
            else if( by<=-12.8 && bx<(px+x/3) &&bx>(px-x/3) && start == 1)
            {
                        dirx = dirx;
                        diry = 1;
            }
            else if(by <=-12.8 && bx<(px-(x*2/3)) && bx>(px-(x+0.3)) && start == 1 )
            {
                        dirx = -1.5;
                        diry = 0.8;
            }
            else if(by<=-12.8 && bx<(px+(x+0.3)) && bx>(px+x/3)&& start == 1 )
            {
                        dirx = 1.5;
                        diry = 0.8;
            }
            else if(by<-13)
            {
                        start = 0;
                        by = -12.8;
                        bx = 0;
                        dirx = 0;
                        diry = 0;
                        px = 0;
            }
            glutPostRedisplay();
}
int main (int argc,char **argv) {
            glutInit(&argc,argv);
            glutInitDisplayMode (GLUT_DOUBLE | GLUT_DEPTH);
            glutInitWindowSize (900, 900);
            glutInitWindowPosition (100, 100);
            glutCreateWindow ("Brick Breaker");
            glutDisplayFunc (display);
            glutReshapeFunc (reshape);
            glEnable(GL_DEPTH_TEST);
            glutIdleFunc (idle);
            glutPassiveMotionFunc(mousemotion);
            glutKeyboardFunc(keyboard);
            lightsOn();
            addMenu();

            glutMainLoop ();
            return 0;
}
```

CONCLUSION AND FUTURE SCOPE

CONCLUSION

The Brick Breaker has been tested under Windows 10 and has been found to provide ease of use and manipulation to the user. The Brick Breaker created for the Windows XP operating system can be used to draw lines, boxes, circles, ellipses, and polygons. It has a very simple and aesthetic user interface.

I found designing and developing this Brick Breaker as a very interesting and learning experience. It helped Me to learn about computer graphics, design of Graphical User Interfaces, interface to the user, user interaction handling and screen management. The graphics editor provides all and more than the features that have been detailed in the university syllabus.

FUTURE ENHANCEMENTS

These are the features that are planned to be supported in the future

* Support for multplayer

* Support for Vr mode

* Support for 3d transformations

* Support for 1v1 online

REFERENCE

[1] Edward Angel’s Interactive Computer Graphics Pearson Education 5th Edition

[2] Interactive computer Graphics --A top down approach using open GL--by Edward Angle

[3] Jackie .L. Neider, Mark Warhol, Tom.R.Davis, "OpenGL Red Book", Second Revised Edition, 2005.

[4] Donald D Hearn and M.Pauline Baker, "Computer Graphics with OpenGL", 3rd Edition.