RV COLLEGE OF ENGINEERING®
BENGALURU – 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

"Bricks Breaker Ball Game"

COMPUTER GRAPHICS LAB (16CS73)

OPEN ENDED EXPERIMENT REPORT

VII SEMESTER

2020-2021

Submitted by

Rohit Raj 1RV17CS125
Mohit Singh 1RV17CS193

Under the Guidance of

Dr. Mamatha T
Department of CSE, R.V.C.E.,
Bengaluru - 560059

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the **Open-Ended Experiment** titled "Bricks Breaker Ball Game" has been carried out by **Rohit Raj (1RV17CS125) & Mohit Singh (1RV17CS193),** bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73)** during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.

Dr. Hemavathy R.
Faculty In-charge,
Department of CSE,
R.V.C.E., Bengaluru –59

Dr. Ramakanth Kumar P
Head of Department,
Department of CSE,
R.V.C.E., Bengaluru–59

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **Rohit Raj (1RV17CS125) & Mohit Singh (1RV17CS193)** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **"Bricks Breaker Ball Game"** has been carried out by us and submitted in partial fulfillment for the Internal **Assessment of** Course**: COMPUTER GRAPHICS LAB (16CS73) - Open-Ended Experiment** during the year 2020-2021. We do declare that matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Place: Bengaluru

Rohit Raj (1RV17CS125)
Mohit Singh (1RV17CS193)

Date: 10-01-2021

ABSTRACT

A 2D graphics-based ball game "Bricks Breaker Ball Game" is a great start for a student who starts learning computer graphics & visualization. The development of the game has large scope to learn computer graphics from scratch. We used OpenGL utility toolkit to implement the algorithm, written it in C++ language.

There is still scope left in the development of project like, after "Game Over" a list should show top ten high scores. Another great improvement to the game would be if the game could be accessed and played by multiple players at the same time. Even the interface could look better and be more user friendly.

In the future we hope we would implement these modifications for a better experience while playing this game. Finally, we could say by developing the game we have learnt the basics of computer graphics and in the future, by improving it further, we shall learn more. It would be our pleasure if we could develop this game using a 3D graphics package.

ACKNOWLEDGEMENTS

While presenting our project on “Bricks Breaker Ball Game”, we feel that it is our duty to acknowledge the help rendered to us by various persons. We would like to convey our thanks to the Principal, the HoD (CSE) Dr. Ramakant Kumar P (RVCE, Bangalore), for being kind enough to provide us with an opportunity to do a project in this institution. We would greatly mention the enthusiastic influence provided by Dr. Hemavathy R, our project guide, for her ideas and co-operation showed on us during the venture and making this project a great success.

We are very much pleasured to express our sincere gratitude to the friendly cooperation showed by all the staff members of Computer Science Department, RVCE. We would also like to thank our friends for helping us with doubts and clarifications and supporting us in spirit to see this project through.

CONTENTS

1. INTRODUCTION

In the project “Bricks Breaker Ball Game”, the project is programmed using C++. Oops concepts are explored and the project involves the bricks and ball where the main objective is to break the bricks through balls and score as high as possible. We also move a rectangular slab through keyboard which supports and reflects the ball back. If the ball falls down, game is over. A total of 3 chances are available after which the game gets over. Ability to pause the game is also provided.

This project includes the multiple windows, menus and submenus using which color of the bat & ball, screen color, ball size will be changed. These all actions are assigned to keyboard and mouse.

User-interface is provided by means of both Keyboard and Mouse. By using arrow keys rectangle box can be moved. Mouse interaction is achieved by means of a menu which is operational only with the “right mouse button” through which ball moves.

.

1.1 Computer Graphics:

Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. 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.

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computers themselves. It has grown to include the creation, storage, and manipulation of models and images of objects. These models come from a diverse and expanding set of fields, and include physical, mathematical, engineering, architectural, and even conceptual structures, natural phenomena, and so on. Computer graphics today is largely interactive. The user controls the contents, structure, and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse, or touch-screen.

Due to close relationships between the input devices and the display, the handling of such devices is included in the study of computer graphics. The advantages of the interactive graphics are many in number. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D patter-recognition abilities allow us to perceive and process data rapidly and efficiently. In many design, implementation, and construction processes today, the information pictures can give is virtually indispensable. Scientific visualization became an important field in the 1980s when the scientists and engineers realized that they could not interpret the prodigious quantities of data produced in supercomputer runs without summarizing the data and highlighting trends and phenomena in various kinds of graphical representations.

1.2 OpenGL Interface:

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are:

1. **Main GL:** Library has names that begin with the letter gl and are stored in a library usually referred to as GL.
2. **OpenGL Utility Library (GLU):** This library uses only GL functions but contains code for creating common objects and simplifying viewing.
3. **OpenGL Utility Toolkit (GLUT):** This provides the minimum functionality that should be accepted in any modern windowing system.

1.3 OpenGL Overview:

- OpenGL(Open Graphics Library) is the interface between a graphic program and graphics hardware. ***It is streamlined.*** In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.

- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- ***It is system-independent.*** It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- ***It is a state machine.*** At any moment during the execution of a program there is a current model transformation
- ***It is a rendering pipeline.*** The rendering pipeline consists of the following steps:
    - Defines objects mathematically.
    - Arranges objects in space relative to a viewpoint.
    - Calculates the color of the objects.
    - Rasterizes the objects.

2. SYSTEM SPECIFICATION

2.1 HARDWARE REQUIREMENTS:

- Dual Core Processor
- 2GB RAM
- 40GB Hard disk
- Mouse and other pointing devices
- Keyboard

2.2 SOFTWARE REQUIREMENTS:

- Programming language – C++ using OpenGL
- Operating system – Windows/Linux operating system
- Compiler – C++ Compiler
- Graphics library – GL/glut.h
- OpenGL 2.0

2.3 FUNCTIONAL REQUIREMENTS:

OpenGL APIs:

If we want to have a control on the flow of program and if we want to interact with the window system then we use OpenGL API'S. Vertices are represented in the same manner internally, whether they are specified as two-dimensional or three-dimensional entities, everything that we do are here will be equally valid in three dimensions. Although OpenGL is easy to learn, compared with other APIs, it is nevertheless powerful. It supports the simple three dimensional programs and also supports the advanced rendering techniques.

GL/glut.h:

We use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.The application program uses only GLUT functions and can be recompiled with the GLUT library for other window system. OpenGL makes a heavy use of macros to increase code readability and avoid the use of magic numbers. In most implementation, one of the include lines

3. ABOUT THE PROJECT

3.1 Overview:

The project involves the bricks and ball where the main objective is to break the bricks through balls and score as high as possible. We also move a rectangular slab through keyboard which supports and reflects the ball back. If the ball falls down, game is over. A total of 3 chances are available after which the game gets over. Ability to pause the game is also provided.

3.2 User Interface:

The interface is mainly concentrated on use of mouse and keyboard. Clicking right button of mouse displays a menu which has various options which helps in changing color of bat, ball and background..

By pressing P and R we can pause and restart the game. By pressing N we can move from the current window to next window.

3.3 Purpose:

The aim of this project is to develop a graphics package which supports basic operations which include building a using Open GL. The package must also has a user-friendly interface. The objective of developing this model was to design and apply the skills we learnt in class.

3.4 Scope:

It provides most of the features that a graphics model should have. It is developed in C language. It has been implemented on LINUX platform. The graphics package designed here provides an interface for the users for playing 2D GAME using bat and ball.

4. IMPLEMENTATION

4.1 FUNCTIONS IN OPEN GL

- **void glClear(glEnum mode);**

Clears the buffers namely color buffer and depth buffer. mode refers to GL_COLOR_BUFFER_BIT or DEPTH_BUFFER_BIT.

- **void glTranslate[fd](TYPE x, TYPE y, TYPE z);**

Alters the current matrix by displacement of (x, y, z), TYPE is either GLfloat or
GLdouble.

- **void glutSwapBuffers();**

Swaps the front and back buffers.

- **void glMatrixMode(GLenum mode);**

Specifies which matrix will be affected by subsequent transformations, Mode can be GL_MODELVIEW or GL_PROJECTION.

- **void glLoadIdentity( );**

Sets the current transformation matrix to identity matrix.

- **void glEnable(GLenum feature);**

Enables an OpenGL feature. Feature can be GL_DEPTH_TEST (enables depth test for hidden surface removal), GL_LIGHTING (enables for lighting calculations), GL_LIGHTi (used to turn on the light for number i), etc.

- **void glPushMatrix();**
  void glPopMatrix();

Pushes to and pops from the matrix stack corresponding to the current matrix mode.

- **void glutBitmapCharacter(void *font, int character);**

Without using any display lists, `glutBitmapCharacter` renders the `character` in the named bitmap `font`. The fonts used are GLUT_BITMAP_TIMES_ ROMAN_24, GLUT_BITMAP_ HELVETICA_18.

- **void glRasterPos[234][ifd](GLfloat x, GLfloat y, GLfloat z);**

This position, called the raster position, is used to position pixel and bitmap write operations.

- **void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);**

Defines an orthographic viewing volume with all the parameters measured from the centre of the projection plane.

- **void glutInit(int *argc, char **argv);**

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 the mode; the value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **void glutCreateWindow(char *title);**

Creates a window on display; the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

- **void glutMainLoop();**

Causes the program to enter an event-processing loop.

- **void glutDisplayFunc(void (*func)(void))**

Registers the display function func that is executed when the window needs to be redrawn.

- **void glutMouseFunc(void *f(int button, int state, int x, int y)**

Registers the mouse callback function f. The callback function returns the button (GLUT_LEFT_BUTTON,etc., the state of the button after the event (GLUT_DOWN), and the position of the mouse relative to the top-left corner of the window.

- **void glutKeyboardFunc(void *f(char key, int width, int height))**

Registers the keyboard callback function f. The callback function returns the ASCII code of the key pressed and the position of the mouse.

- **void glClearColor(GLclampf r, GLclampf g, GLclamp b, Glclamp a)**

Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

- **void glViewport(int x ,int y, GLsizei width, GLsizei height)**

Specifies the width*height viewport in pixels whose lower left corner is at (x,y) measured from the origin of the window.

- **void glColor3[b I f d ub us ui](TYPE r, TYPE g, TYPE b)**

Sets the present RGB colors. Valid types are bytes(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui). The maximum and minimum value for floating point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are those of the type, for eg, 255 and 0 for unsigned bytes.

- **void glutInitWindowSize(int width, int height);**

Specifies the initial height and width of the window in pixels.

- **void glutReshapeFunc(void *f(int width, int height));**

Registers the reshape callback function f. the callback function returns the height and width of the new window. The reshape callback invokes the display callback.

- **void createMenu(void);**

This function is used to create menus which are used as options in program.

4.2 USER DEFINED FUNCTIONS:

- **void reshape ( ) function:**

It is used for defining the viewport volume using glViewport(int x ,int y,GLsizei width,Glsizei height).

- **int main ( ) function:**

The main function is used for creating the window for display of the model of the atom. Here, we create menu for ease of use for the user. The callback functions, i.e., mouse callback, keyboard callback, display callback, idle callback, are written in main. The callback functions registered in main ( ) are,

```
glutKeyboardFunc (mykeys);
glutDisplayFunc (display);
glutReshapeFunc (reshape);
glutMouseFunc (mouse);
```

- **void createBricks()**

This function is used to create the bricks in the game.

- **void print(int a)**

This function is used to print the score in the game and also print the previous score.

- **void completeMessage(bool a)**

This function is used to show the final score in the game.

- **bool checkCollision(float aX, float aY, float aW, float aH, float bX, float bY, float bW, float bH)**

This function is used to check the collision in the game.

.

5. TESTING

Testing process started with the testing of individual program units such as functions or objects. These were then integrated into sub-systems and systems, and interactions of these units were tested.

Testing involves verification and validation. Validation: “Are we building right product?” Verification: “Are we building the product right?”

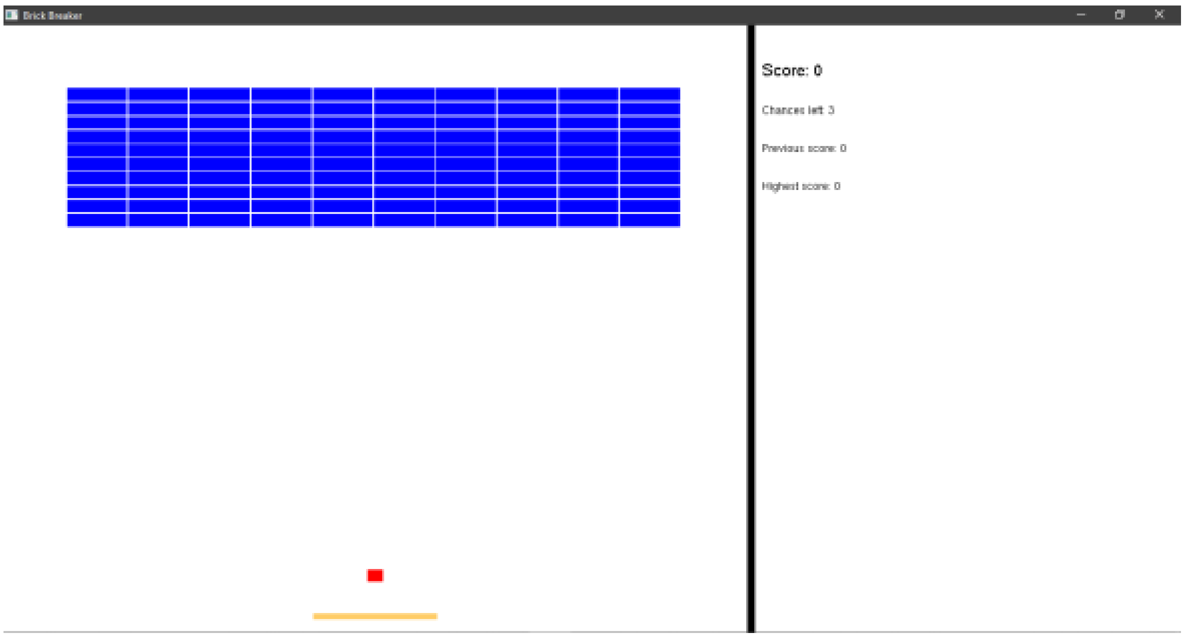
The ultimate goal of the verification and validation process is to establish confidence that the software system is ‘fit for purpose’. The level of required confidence depends on the system’s purpose, the expectations of the system users and the current marketing environment for the system.

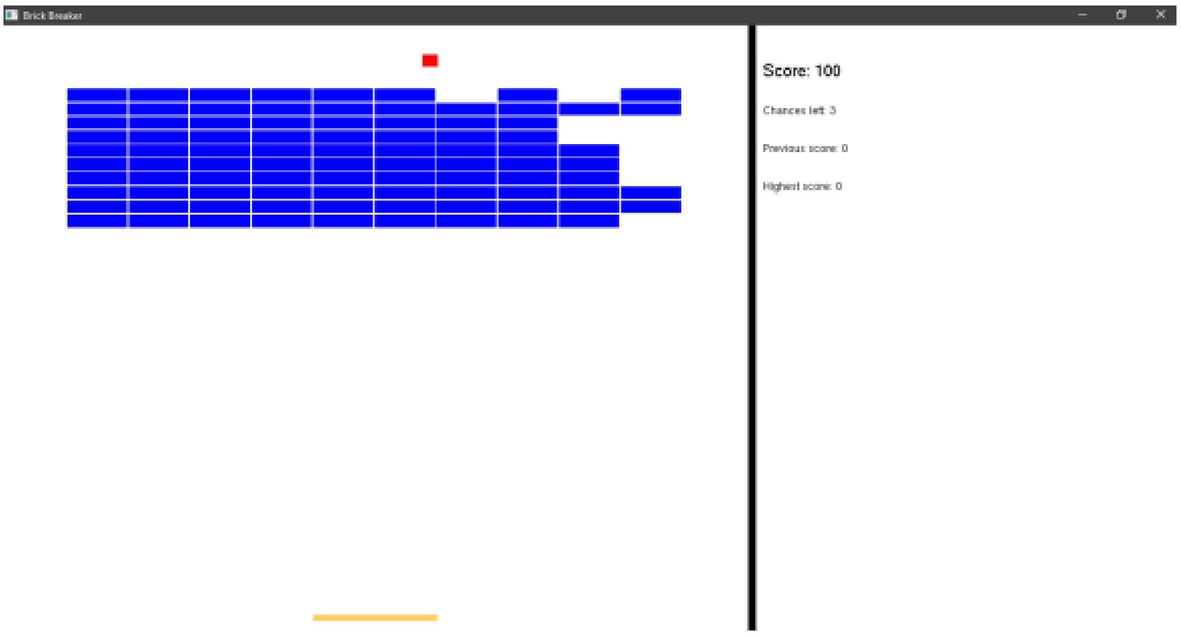
With the verification and validation process, there are two complementary approaches to the system checking and analysis:

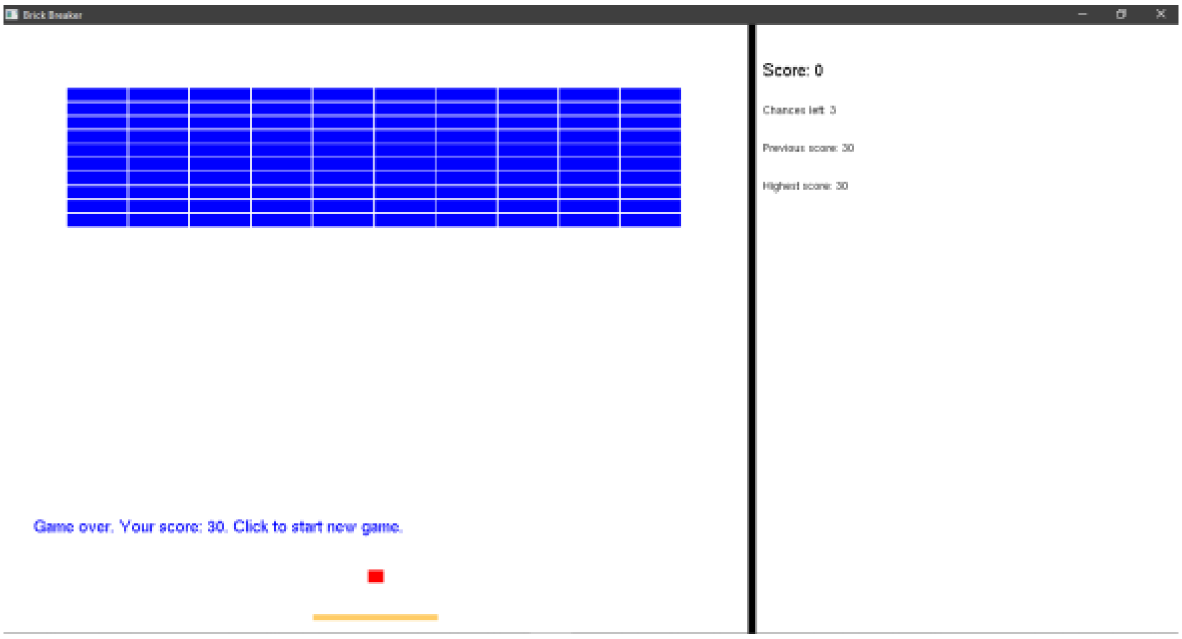
Software inspections or peer reviews analyses and check system representations such as the requirements document, design diagrams, and the program source code.

Software testing involves running an implementation of the software with test data.

6. SNAPSHOTS


Brick Breaker
Score: 0
Chances left: 3
Previous score: 0
Highest score: 0


Brick Breaker
Score: 100
Chances left: 3
Previous score: 0
Highest score: 0


Brick Breaker

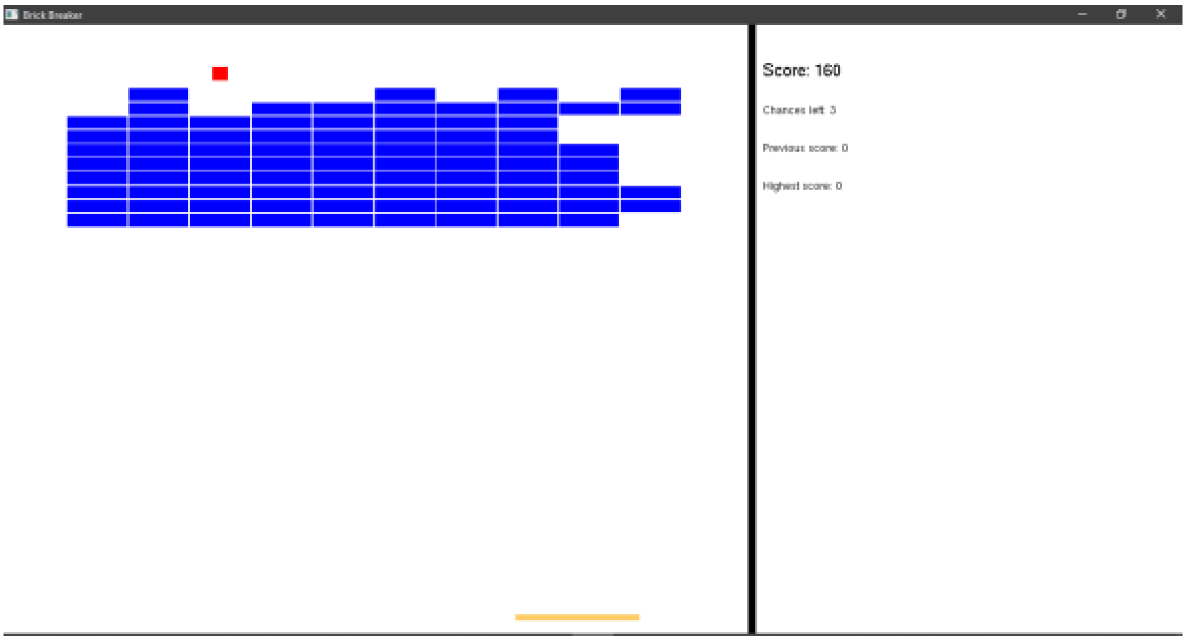
Game over. Your score: 30. Click to start new game.

Score: 0

Chances left: 3

Previous score: 30

Highest score: 30


Brick Breaker

Score: 160

Chances left: 3

Previous score: 0

Highest score: 0

7. CONCLUSION AND ENHACEMENTS

The project was started with modest aim with no prior experience in any programming projects as this, but ended up in learning many things, fine tuning the programming skills and getting into the real world of software development with an exposure to corporate environment. During the development of any software of significant utility, we are forced with the tradeoff between speed of execution and amount of memory consumed. This is simple interactive application. It is extremely user friendly and has the features, which makes simple graphics project. It has an open source and no security features has been included. The user is free to alter the code for feature enhancement. Checking and verification of all possible types of the functions are taken care. Care was taken to avoid bugs. Bugs may be reported to creator as the need.

Further this project can be enhanced by adding few more options i,e menus in game. Using this we can design a 3D game which contains cube instead of single window and multiple number of balls which are randomly moving and all faces of cube is considered as wall.

8. BIBLIOGRAPHY

Reference Books and E-books

1. Donald D. Hearn, M. Pauline Baker, Warren Carithers , Computer Graphics with OpenGL, 4th Edition, Publisher Pearson Education, November 2010, ISBN-13: 978- 0136053583

2. Edward Angel, Interactive Computer Graphics: A Top-Down Approach Using OpenGL, 5th Edition, Publisher Pearson Education, 2010, ISBN: 978131725306

3. Zhigang Xiang and Roy Plastock ,Computer Graphics 2nd Edition, ASIN: 0070601658, Publisher Tata McGraw-Hill, 2007, ISBN-13: 978-0070601659

4. www.opengl.org/resources/code/samples/redbook.

APPENDIX A – SOURCE CODE

```
#include<windows.h>
#include<stdio.h>
#include<iostream>
#include<GL/glut.h>
#include<string>
#include<sstream>

//folow this copied code


using namespace std;

float barX = 200, barY = 465, barWidth = 80, barheight = 5;
float ballX = 235, ballY = 430, ballWH = 10, ballVelX = 0.3, ballVelY = 0.3;
const int brickAmount = 100;
int score = 0, chances = 3, previousScore = 0, highestScore = 0;
bool flag = true, flag2 = true;

struct bricks {
    float x;
    float y;
    float width;
    float height;
    bool isAlive = true;
};
bricks bricksArray[brickAmount];

void createBricks() {
    float brickX = 41, brickY = 50;
    for (int i = 0; i < brickAmount; i++) {
        if (brickX > 400) {
            brickX = 41;
            brickY += 11;
        }
        bricksArray[i].x = brickX;
        bricksArray[i].y = brickY;
        bricksArray[i].width = 38.66;
        bricksArray[i].height = 10;
        brickX += 39.66;
    }
    glColor3ub(0, 0, 255);
    glBegin(GL_QUADS);
    for (int i = 0; i < brickAmount; i++) {
        if (bricksArray[i].isAlive == true) {
            glVertex2f(bricksArray[i].x, bricksArray[i].y);
            glVertex2f(bricksArray[i].x + bricksArray[i].width, bricksArray[i].y);
            glVertex2f(bricksArray[i].x + bricksArray[i].width, bricksArray[i].y + bricksArray[i].height);
            glVertex2f(bricksArray[i].x, bricksArray[i].y + bricksArray[i].height);
        }
    }
    glEnd();
}

void print(int a) {
    glRasterPos2f(490, 40);
    stringstream ss;
    ss << a;
    string s = "Score: " + ss.str();
    int len = s.length();
    for (int i = 0; i < len; i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, s[i]);
    }
    glRasterPos2f(490, 70);
    stringstream ss2;
    ss2 << chances;
    string chance = "Chances left: " + ss2.str();
    for (int i = 0; i < chance.length(); i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, chance[i]);
    }
    glRasterPos2f(490, 100);
    stringstream ss3;
    ss3 << previousScore;
    string prevScore = "Previous score: " + ss3.str();
    for (int i = 0; i < prevScore.length(); i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, prevScore[i]);
    }
    glRasterPos2f(490, 130);
    stringstream ss4;
    ss4 << highestScore;
    string highScore = "Highest score: " + ss4.str();
    for (int i = 0; i < highScore.length(); i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, highScore[i]);
    }
    /*    string
        for(int i = 0; i < highScore.length(); i++){
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, highScore[i]);
        }*/
}

void message(bool a) {
    if (a == false) {
        glRasterPos2f(20, 400);
```

```
 stringstream ss;
        ss << previousScore;
        string s = "Game over. Your score: " + ss.str() + ". Click to start new game.";
        int len = s.length();
        for (int i = 0; i < len; i++) {
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, s[i]);
        }
    }
}

void completeMessage(bool a) {
    if (a == false) {
        glRasterPos2f(20, 400);
        stringstream ss;
        ss << score;
        string s = "STAGE COMPLETE. Your score: " + ss.str() + ". Click to start new game.";
        int len = s.length();
        for (int i = 0; i < len; i++) {
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, s[i]);
        }
    }
}

void myDisplay(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    //Bar
    glBegin(GL_QUADS);
    glColor3ub(255, 204, 102);
    glVertex2f(barX, barY);
    glVertex2f(barX + barWidth, barY);
    glVertex2f(barX + barWidth, barY + barheight);
    glVertex2f(barX, barY + barheight);
    glEnd();
    //Ball
    glBegin(GL_QUADS);
    glColor3ub(255, 0, 0);
    glVertex2f(ballX, ballY);
    glVertex2f(ballX + ballWH, ballY);
    glVertex2f(ballX + ballWH, ballY + ballWH);
    glVertex2f(ballX, ballY + ballWH);
    glEnd();

    //sidebar
    glBegin(GL_QUADS);
    glColor3ub(0, 0, 0);
    glVertex2f(480, 0);
 glVertex2f(480, 480);
    glVertex2f(485, 480);
    glVertex2f(485, 0);
    glEnd();

    print(score);
    createBricks();
    message(flag);
    completeMessage(flag2);

    glutSwapBuffers();
}

void myInit(void) {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glViewport(0, 0, 760, 480);
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 760.0, 480.0, 0.0);
}

bool checkCollision(float aX, float aY, float aW, float aH, float bX, float bY, float bW, float bH) {
    if (aY + aH < bY)
        return false;
    else if (aY > bY + bH)
        return false;
    else if (aX + aW < bX)
        return false;
    else if (aX > bX + bW)
        return false;
    else
        return true;
}

void moveBall() {
    if (score >= 300) {
        ballVelX = 0.5;
        ballVelY = 0.5;
    }
    if (score >= 1000) {
        barX = 200;
        barY = 465;
        ballX = 235;
        ballY = 430;
        ballVelX = 0;
 ballVelY = 0;
        float brickX = 2, brickY = 2;
        for (int i = 0; i < brickAmount; i++) {
            if (brickX > 450) {
                brickX = 2;
                brickY += 11;
            }
            bricksArray[i].x = brickX;
            bricksArray[i].y = brickY;
            bricksArray[i].width = 38.66;
            bricksArray[i].height = 10;
            bricksArray[i].isAlive = true;
            brickX += 39.66;
        }
        previousScore = score;
        if (highestScore < score) {
            highestScore = score;
        }
        chances = 3;
        score = 0;
        flag2 = false;
        Sleep(3000);
        completeMessage(flag2);
    }
    else {
        ballX += ballVelX;
        for (int i = 0; i < brickAmount; i++) {
            if (bricksArray[i].isAlive == true) {
                if (checkCollision(ballX, ballY, ballWH, ballWH, bricksArray[i].x, bricksArray[i].y, bricksArray[i].width,
bricksArray[i].height) == true) {
                    ballVelX = -ballVelX;
                    bricksArray[i].isAlive = false;
                    score += 10;
                    break;
                }
            }
        }
        ballY -= ballVelY;
        for (int i = 0; i < brickAmount; i++) {
            if (bricksArray[i].isAlive == true) {
                if (checkCollision(ballX, ballY, ballWH, ballWH, bricksArray[i].x, bricksArray[i].y, bricksArray[i].width,
bricksArray[i].height) == true) {
                    ballVelY = -ballVelY;
                    bricksArray[i].isAlive = false;
                    score += 10;
                    break;
                }
            }
```

```
}
if (ballX < 0) {
    ballVelX = -ballVelX;
}
else if (ballX + ballWH > 480) {
    ballVelX = -ballVelX;
}
if (ballY < 0) {
    ballVelY = -ballVelY;
}
else if (ballY + ballWH > 480) {
    if (chances <= 1) {
        barX = 200;
        barY = 465;
        ballX = 235;
        ballY = 430;
        ballVelX = 0;
        ballVelY = 0;
        float brickX = 2, brickY = 2;
        for (int i = 0; i < brickAmount; i++) {
            if (brickX > 450) {
                brickX = 2;
                brickY += 11;
            }
            bricksArray[i].x = brickX;
            bricksArray[i].y = brickY;
            bricksArray[i].width = 38.66;
            bricksArray[i].height = 10;
            bricksArray[i].isAlive = true;
            brickX += 39.66;
        }
        previousScore = score;
        if (highestScore < score) {
            highestScore = score;
        }
        chances = 3;
        score = 0;
        flag = false;
        Sleep(3000);
        message(flag);
    }
    else {
        chances--;
        ballX = 235;
        ballY = 430;
        if (ballVelY < 0) {
            ballVelY = -ballVelY;
        }
```

```
                Sleep(3000);
            }
        }
        if (checkCollision(ballX, ballY, ballWH, ballWH, barX, barY, barWidth, barheight) == true) {
            ballVelY = -ballVelY;
        }
    }
    glutPostRedisplay();
}

void keyboard(int key, int x, int y) {
    switch (key) {
    case GLUT_KEY_LEFT:
        barX -= 10;
        if (barX < 0) {
            barX = 0;
        }
        glutPostRedisplay();
        break;
    case GLUT_KEY_RIGHT:
        barX += 10;
        if (barX + barWidth > 480) {
            barX = 480 - barWidth;
        }
        glutPostRedisplay();
        break;
    default:
        break;
    }
}

void mouse(int button, int state, int x, int y) {
    switch (button) {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN) {
            flag = true;
            if (ballVelX <= 0 && ballVelY <= 0) {
                ballVelX = 0.3;
                ballVelY = 0.3;
            }
            glutIdleFunc(moveBall);
        }
        break;
    default:
        break;
    }
}
```

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(760, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("Brick Breaker");
    glutDisplayFunc(myDisplay);
    glutSpecialFunc(keyboard);
    glutMouseFunc(mouse);
    myInit();
    glutMainLoop();
}
```