Bachelor of Science in Computer Science & Engineering

Brick Breaker Blitz: A Ball-Bouncing Odyssey

by

Salman Farsi

ID: 1804102

Department of Computer Science & Engineering

Chittagong University of Engineering & Technology (CUET)

Chattogram-4349, Bangladesh.

13th August, 2023

Chittagong University of Engineering & Technology (CUET)
Department of Computer Science & Engineering
Chattogram-4349, Bangladesh.

Project Report

Computer Graphics Sessional (CSE - 458)

Student Name : Salman Farsi Session : 2021-2022
ID : 1804102

Submitted To : Dr. Muhammad Ibrahim Khan
Professor
Department of Computer Science & Engineering

Moumita Sen Sharma
Lecturer
Department of Computer Science & Engineering

Shuhena Salam Aonty
Lecturer
Department of Computer Science & Engineering

Department : Computer Science & Engineering
Program : B.Sc. Engineering

Tentative Title : Brick Breaker Blitz: A Ball-Bouncing Odyssey

Table of Contents

List of Figures

List of Code Snippets

1 Introduction

The Brick Breaker Ball Bouncing Game is an exciting arcade-style game developed using the OpenGL library and C++. Players control a bouncing ball and attempt to destroy bricks with it while keeping it from falling off the screen. In the game, a two-dimensional space displays a ball moving at a certain speed and direction. Players can control a horizontal bar or paddle at the bottom of the screen to guide the ball's movement. The objective is to make the ball bounce off walls, the ceiling, and the paddle to hit and break bricks located at the top of the screen. Each broken brick earns the player points.

The game implements the OpenGL library to render graphics and handle user interactions. The game loop, the core structure, updates the game's state and renders visuals in each iteration. Key components include rendering using OpenGL, physics to handle ball movement and bouncing, user input for paddle control, and game logic with scoring and level design.

The game starts with the ball at the center, moving randomly. Players use keyboard inputs to move the paddle left or right. The ball bounces off walls, ceiling, and paddle. The objective is to break bricks at the top, earning points. The game continues until the ball falls off the screen.

In this project, such type of a game is developed that ensures the game's simplicity, coupled with challenging gameplay, making it enjoyable and engaging for players. Players can attempt to beat their high scores and improve their skills in subsequent rounds.

2 Specific Objectives

The objectives of this game are as follows:

1. To create a game where players control a bouncing ball to destroy bricks while preventing it from falling off the screen.
2. To utilize the OpenGL library to render the game's graphics, including the ball, paddle, bricks, and game environment.

3. To incorporate a mechanism to store and display players' high scores for motivation and competition.

4. To implement collision detection algorithms to detect interactions between the ball, paddle, and bricks accurately.

5. To implement a game loop structure to ensure continuous updating of game state and rendering of visuals in each iteration.

3 Game Logic and Features

The figure 3.1 depicts the scenario of the whole game.

3.1 Game Flow

1. The game starts by displaying the game window with a bar, a ball, and a set of bricks arranged in rows and columns.

2. The player can move the bar left and right using the arrow keys or by moving the mouse. Initially, the ball is stationary.

3. The player can start the game by clicking the left mouse button. The ball starts moving with a fixed velocity.

4. The ball bounces off the walls, the bar, and the bricks. When the ball collides with a brick, the brick is destroyed, and the player earns points.

5. If the ball hits the bottom of the screen, the player loses a chance. The game continues until the player runs out of chances or completes all stages.

6. If the player reaches a score of 1000 points, the game progresses to the next stage. In the next stage, the bricks are regenerated, and the player's score is reset.

3.2 Game Elements

- **Bar**: The bar is the player-controlled element used to bounce the ball and prevent it from going below the screen.

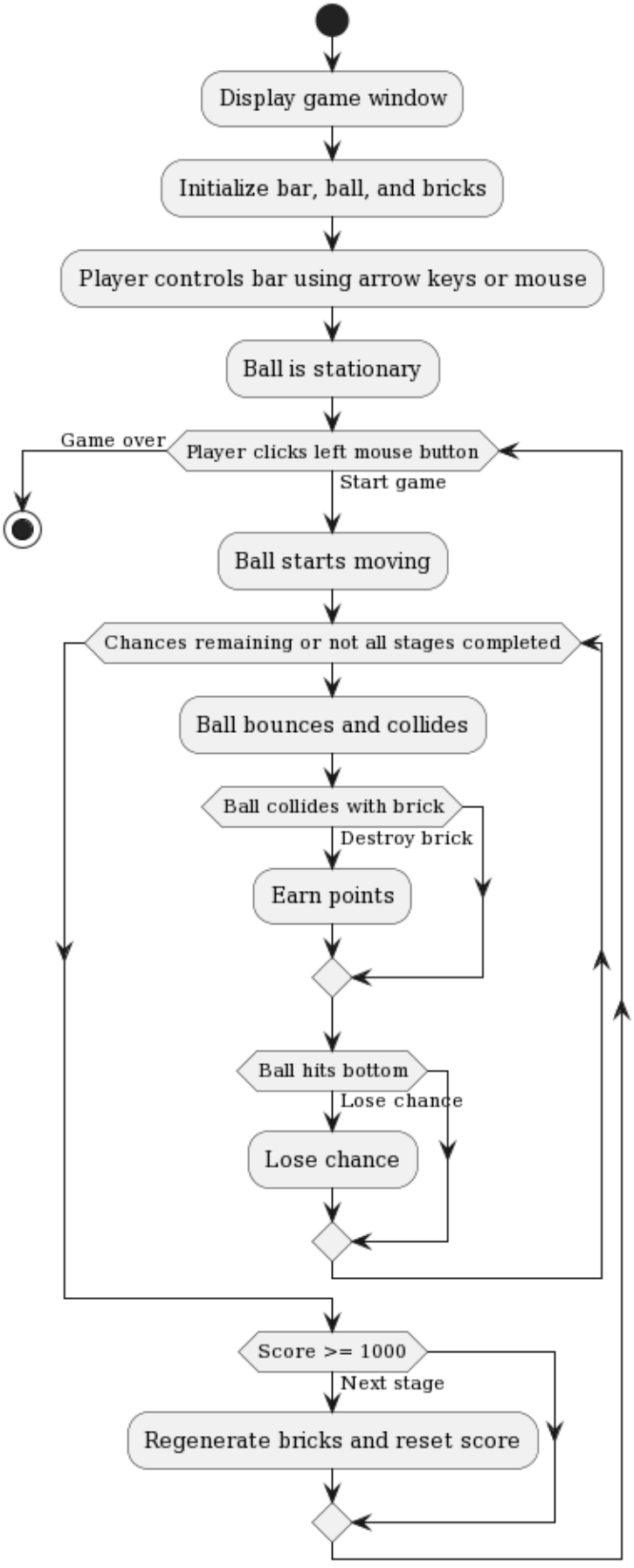

Figure 3.1: Game's Flow Chart

- **Ball**: The ball is the main element that bounces on the screen. The player must control the bar to keep the ball in play.
- **Bricks**: The bricks are destructible elements placed on the screen in a grid pattern. The player earns points by hitting the ball against the bricks, destroying them in the process.

3.3 Score and Scoring Logic

The player earns 10 points for each brick they destroy. The score is displayed on the screen in real time. The game keeps track of the player's highest score and displays it as well.

3.4 Controls

- **Mouse Click**: Start the game and release the ball.
- **Keyboard Arrow Keys**: Move the bar left and right to control the ball's direction.
- **Mouse Motion**: Move the bar left and right by moving the mouse on the screen.

4 Real Time Game Interface

In the interface 4.2, a starting interface of the game is shown. This interface will appear as soon as the user starts the game. Initially, all 3 chances are available as no chances have been wasted yet. In the interface 4.3, it can be seen that some bricks were broken as the game proceeded. And players might have wasted 2 chances already with 1 chance being left. Again in the interface 4.4, the highest score and the previous score of the player's current session appeared. It indicates that the highest score will update automatically after each trial. Finally in the interface 4.5 below, a game over screen is shown. When players waste all of their attempts, they have to restart the game again.

Figure 4.2: Starting Interface with all three chances left

Figure 4.3: Game playing display while only one chance left

Figure 4.4: Previous score and highest score display

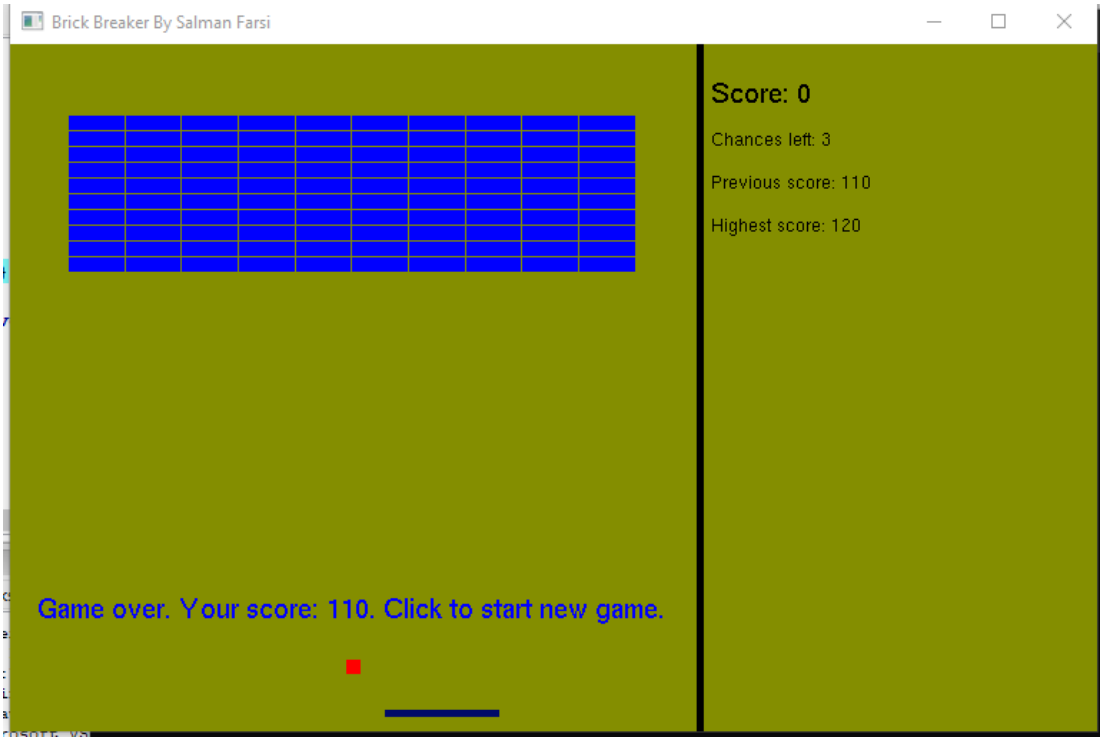

Figure 4.5: Game Over Window

5 Source Code Explanation

5.1 Libraries and Global Variables

Function 1: Libraries and Global Variables

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

```

Explanation:

- The ‘windows.h’ library is included for Windows-specific functions.
- ‘stdio.h’ and ‘iostream’ are included for standard input/output.
- ‘GL/glut.h’ is included for OpenGL graphics functions.
- ‘string’ and ‘sstream’ are included for string manipulation.

- The ‘using namespace std;’ statement allows us to use elements of the ‘std’ namespace without explicitly qualifying them.
- The global variables are used to store the positions, dimensions, and velocities of the bar and ball, the score, number of chances, and other game-related data.
- ‘brickAmount’ represents the total number of bricks in the game.
- The ‘bricks’ struct is defined to store information about each brick, including its position, width, height, and whether it is currently alive (not destroyed).
- ‘bricksArray’ is an array of ‘bricks’ structures, used to store the information of all the bricks in the game.

5.2 Bricks Creations

Function 2: bricksCreate()

```
1   void createBricks() {
2   float brickX = 41, brickY = 50;
3   for(int i = 0; i < brickAmount; i++) {
4       if(brickX > 400) {
5           brickX = 41;
6           brickY += 11;
7       }
8       bricksArray[i].x = brickX;
9       bricksArray[i].y = brickY;
10      bricksArray[i].width = 38.66;
11      bricksArray[i].height = 10;
12      brickX += 39.66;
13  }
14  glColor3ub(0,0,255);
15  glBegin(GL_QUADS);
16  for(int i = 0; i < brickAmount; i++) {
17      if(bricksArray[i].isAlive == true) {
18          glVertex2f(bricksArray[i].x, bricksArray[i].y);
19          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();
}
```

Explanation:

- This function is responsible for creating and drawing the bricks on the screen.
- It uses a nested loop to initialize the positions of the bricks in a grid pattern.
- The outer loop iterates through the rows, and the inner loop iterates through the columns of the grid.
- The 'bricksArray' is updated with the positions, width, and height of each brick.
- The bricks are drawn on the screen using 'GL_QUADS' to create rectangles for each brick.
- The color of the bricks is set to blue (RGB: 0, 0, 255).

5.3 Print Game Elements

Function 3: print()

```
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
   ]);
    }
}
}
```

Explanation:

- This function is responsible for displaying various game-related information on the screen, such as the player's score, remaining chances, previous score, and highest score.
- The 'glRasterPos2f()' function is used to set the position on the screen where the text will be displayed.
- 'stringstream' is used to convert integer values to strings for display.
- The strings to be displayed are formed by concatenating the respective labels with the corresponding values (score, chances, etc.).

- The 'glutBitmapCharacter()' function is used to render each character of the strings on the screen.

5.4 Ball Movement

Function 4: moveBall()

```
void moveBall() {
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;
```

```
34          ballY -= ballVelY;
35
36          for (int i = 0; i < brickAmount; i++) {
37              if (bricksArray[i].isAlive == true) {
38                  if (checkCollision(ballX, ballY, ballWH, ballWH,
    bricksArray[i].x, bricksArray[i].y, bricksArray[i].width,
    bricksArray[i].height) == true) {
39                      ballVelX = -ballVelX;
40                      bricksArray[i].isAlive = false;
41                      score += 10;
42                      break;
43                  }
44              }
45          }
46          if (ballX < 0 || ballX + ballWH > 480) {
47              ballVelX = -ballVelX;
48          }
49          if (ballY < 0) {
50              ballVelY = -ballVelY;
51          }
52          if (checkCollision(ballX, ballY, ballWH, ballWH, barX,
    barY, barWidth, barheight) == true) {
53              ballVelY = -ballVelY;
54          }
55          if (ballY + ballWH > 480) {
56              if (chances <= 1) {
57                  barX = 200;
58                  barY = 465;
59                  ballX = 235;
60                  ballY = 430;
61                  ballVelX = 0;
62                  ballVelY = 0;
63                  float brickX = 2, brickY = 2;
64                  for (int i = 0; i < brickAmount; i++) {
65                      if (brickX > 450) {
66                          brickX = 2;
67                          brickY += 11;
68                      }
69                      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(300);
                message(flag);
            }
            else {
                chances--;
                ballX = 235;
                ballY = 430;
                if (ballVelY < 0) {
                    ballVelY = -ballVelY;
                }
                Sleep(300);
            }
        }
    }
    glutPostRedisplay();
}
```

Explanation:

- This function is responsible for updating the position and movement of the ball, handling collisions, scoring, and managing game flow.
- When the player's score reaches 1000 points, the function resets the ball, bar, and bricks for the next stage and updates the score, chances, and the highest score accordingly.

- If the player's score is less than 1000, the ball is moved in both x and y directions.
- The function checks for collisions between the ball and the bricks and handles the ball's direction and scoring logic accordingly.
- It also checks for collisions with the bar and reverses the ball's direction if it hits the bar.
- If the ball goes below the screen, the function reduces the chances and resets the ball position. If no chances remain, the game over message is displayed.
- The 'glutPostRedisplay()' function is used to mark the window for redrawing, ensuring that the display is updated.

5.5 Collsion Checking of Ball with Bricks

Function 5: checkCollision()

```
1    bool checkCollision(float aX, float aY, float aW, float aH,
     float bX, float bY, float bW, float bH) {
2    if (aY + aH < bY)
3        return false;
4    else if (aY > bY + bH)
5        return false;
6    else if (aX + aW < bX)
7        return false;
8    else if (aX > bX + bW)
9        return false;
10   else
11       return true;
12 }
```

Explanation:

- This function checks for collisions between two rectangles (in this case, the ball and the bricks).
- It takes the coordinates (x, y), width (w), and height (h) of two rectangles (a and b) as input parameters.

- The function uses basic axis-aligned bounding box collision detection to check if two rectangles overlap.
- If the rectangles do not overlap in any direction (top, bottom, left, right), the function returns 'false'; otherwise, it returns 'true'.

5.6 Message for Game Over

Function 6: message()

```
    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 a new game.";
        int len = s.length();
        for(int i = 0; i < len; i++) {
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, s[i]);
        }
    }
}
```

Explanation:

- This function displays a game over message when the player loses all their chances (when 'a' is 'false').
- The 'glRasterPos2f()' function is used to set the position on the screen where the message will be displayed.
- The player's previous score is obtained and converted to a string using 'stringstream'.
- The message string is formed by concatenating the player's previous score and other relevant information.
- The 'glutBitmapCharacter()' function is used to render each character of the message on the screen.

5.7 Keyboard Operation

Function 7: keyboard()

```
 void keyboard(int key, int x, int y) {
switch(key) {
        case GLUT_KEY_LEFT:
            barX -= 50;
            if(barX < 0) {
                barX = 0;
            }
            glutPostRedisplay();
            break;
        case GLUT_KEY_RIGHT:
            barX += 50;
            if(barX + barWidth > 480) {
                barX = 480 - barWidth;
            }
            glutPostRedisplay();
            break;
        default:
            break;
    }
}
```

Explanation:

- This function is the callback for handling keyboard input for moving the bar left and right using the arrow keys.
- The function takes the 'key', 'x', and 'y' parameters from the GLUT library.
- When the left arrow key is pressed, the bar's x-position is decreased by 50 pixels, and when the right arrow key is pressed, the bar's x-position is increased by 50 pixels.
- The 'if' statements ensure that the bar does not go beyond the screen boundaries (0 to 480 - barWidth).
- The 'glutPostRedisplay()' function is used to mark the window for redrawing, ensuring that the display is updated after the bar position is changed.

- This function is the callback for handling keyboard input for moving the bar left and right using the arrow keys.
- The function takes the 'key', 'x', and 'y' parameters from the GLUT library.
- When the left arrow key is pressed, the bar's x-position is decreased by 50 pixels, and when the right arrow key is pressed, the bar's x-position is increased by 50 pixels.
- The 'if' statements ensure that the bar does not go beyond the screen boundaries (0 to 480 - barWidth).
- The 'glutPostRedisplay()' function is used to mark the window for redrawing, ensuring that the display is updated after the bar position is changed.

5.8 Mouse Movement

Function 8: mouse()

```
1   void motion(int x, int y) {
2       float mouseX = static_cast<float>(x) / glutGet(
    GLUT_WINDOW_WIDTH);
3       float mouseY = static_cast<float>(y) / glutGet(
    GLUT_WINDOW_HEIGHT);
4
5       float newBarX = mouseX * (480 - barWidth);
6       if (newBarX < 0) {
7           barX = 0;
8       } else if (newBarX + barWidth > 480) {
9           barX = 480 - barWidth;
10      } else {
11          barX = newBarX;
12      }
13
14      glutPostRedisplay(); // Mark the window for redrawing
15  }
```

Explanation:

- This function is the callback for handling mouse motion to move the bar based on the mouse position.

- The function takes the ‘x’ and ‘y’ parameters from the GLUT library, which represent the current mouse coordinates on the window.
- The mouse coordinates are converted to normalized device coordinates (NDC) by dividing by the window width and height (‘glutGet(GLUT_-WINDOW_WIDTH)’ and ‘glutGet(GLUT _WINDOW_HEIGHT)’).
- The new position of the bar (‘newBarX’) is calculated by multiplying the normalized x-coordinate by the available space for the bar to move (480 - barWidth).
- The ‘if’ statements ensure that the bar does not go beyond the screen boundaries (0 to 480 - barWidth).
- The ‘glutPostRedisplay()’ function is used to mark the window for redrawing, ensuring that the display is updated after the bar position is changed.

5.9 Display Function

Function 9: myDisplay()

```
void myDisplay(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);

    glBegin(GL_QUADS);
    glColor3ub(255, 204, 102);
    glVertex2f(barX, barY);
    glVertex2f(barX + barWidth, barY);
    glVertex2f(barX + barWidth,

    barY + barheight);
    glVertex2f(barX, barY + barheight);
    glEnd();

    glBegin(GL_QUADS);
    glColor3ub(255, 0, 0);
    glVertex2f(ballX, ballY);
    glVertex2f(ballX + ballWH, ballY);
    glVertex2f(ballX + ballWH, ballY + ballWH);
```

```
    glVertex2f(ballX, ballY + ballWH);
    glEnd();

    createBricks();
    print(score);
    if (flag == false) {
        message(flag);
    }
    glFlush();
}
```

Explanation:

- This function is responsible for displaying the game elements and game-related information on the screen.
- 'glClear(GL_COLOR_BUFFER_BIT)' clears the color buffer with the color specified in 'glClearColor()' (white in this case).
- The bar is drawn as a colored rectangle using 'GL_QUADS'.
- The ball is drawn as a colored rectangle using 'GL_QUADS'.
- The bricks are drawn using the 'createBricks()' function.
- The player's score and game-related information are displayed using the 'print()' and 'message()' functions, respectively.
- 'glFlush()' forces all issued OpenGL commands to be executed as quickly as possible.

6 Application

Entertainment and Recreation:

- The game provides a fun and engaging leisure activity for players of all ages, offering a nostalgic experience reminiscent of classic arcade games.
- The game provides a fun and engaging leisure activity for players of all ages, offering a nostalgic experience reminiscent of classic arcade games.

Skill Development:

- The game requires players to develop hand-eye coordination, reflexes, and spatial awareness as they control the paddle and aim to hit the ball accurately.
- It can serve as a platform for players to enhance their concentration and reaction time.

Stress Relief and Relaxation:

- Playing the game can offer a stress-relieving outlet, allowing players to focus on the gameplay and momentarily escape from stressors.

Cognitive Engagement:

- The game's mechanics encourage strategic thinking as players aim to predict the ball's trajectory and plan their paddle movements to hit specific targets
- It offers a mental challenge while remaining accessible to players of varying skill levels.

Gamification of Learning:

- Educational institutions could utilize similar game mechanics to create interactive learning environments for subjects involving physics, geometry, and spatial reasoning.
- The concept of angles, trajectories, and collisions could be explored through a playful context.

Memory and Focus Training:

- Variants of the game could be employed as memory and focus training tools, requiring players to remember brick patterns and react quickly to changing scenarios.

Mobile and Web Applications:

- The game's concept can be adapted for mobile devices or web browsers, offering a convenient and accessible gaming experience on a wider range of platforms.

7 Required Resources

7.1 Hardware Resources

- Personal Computer

7.2 Software Resources

- OS: Windows 10
- IDE: Codeblocks
- Libraries: OpenGL libraries and Regular C++ libraries
- Programming Language: C++

8 Conclusion

In conclusion, it can be said that the implementation of this straightforward Brick Breaker Ball Bouncing Game enhances user experience and ensures smooth gameplay. Players find enjoyment in challenging themselves for higher scores, while the use of the OpenGL library creates a visually engaging interface. Further potential lies in developing multiple levels with varying brick arrangements to sustain player engagement and challenge. Additionally, the game's adaptability makes it suitable for conversion into an Android application or similar platform, broadening its accessibility in the near future.