# Computer Graphics (UCS505)

# Project on

# Tile Smasher Extreme

**Submitted By**

Abhay Bedi 102016081

Ishaan Bansal 102196011

# 3CS11

**B.E. Third Year – CSE**

**Submitted To:**

**Ms.Jasmine Kaur**



**Computer Science and Engineering Department Thapar Institute of Engineering and Technology Patiala – 147001**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Description** | **Page No.** |
| 1. | Introduction to Project | 2 |
| 2. | Computer Graphics concepts used | 3 |
| 3. | User Defined Functions & classes | 4 |
| 4. | Code | 5 |
| 5. | Output/ Screen shots | 14 |

**1. Introduction to project**

The objective of our project is to create a 2-D game called "Tile Smasher Extreme" utilizing openGL. This game will involve manipulating objects through various movements and transformations such as translation. Additionally, we are designing the game to have a user-friendly interface.

**1.1. Brick Breaker:**

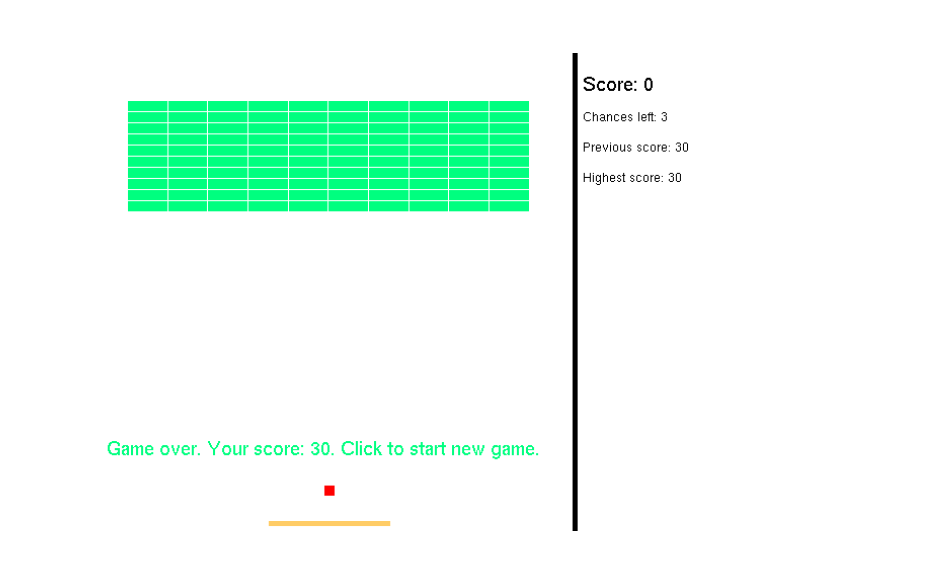
Brick Breaker, a platformer game, mimics Breakout as the player's goal is to obliterate a brick wall by deflecting a bouncing ball with a paddle.

The paddle can move horizontally and is navigated using either the BlackBerry's trackwheel, computer mouse, or keyboard. Initially, the player is given 3 lives, and a life is deducted each time the ball collides with the bottom of the screen.

Once all bricks have been destroyed, the game resets. As soon as the score hits 300, the speed amplifies by 1.5 times. The game ends once all lives are lost.

**1.2. Explanation about project:**

This mini project involves implementing Brick Breaker using OPENGL for computer graphics, using C/C++ programming language. Initially, the player has three lives, but this can be adjusted later on. To play the game, the player must use the arrow keys to control the board.



**2. Computer Graphics concept used:**

This project utilizes OPENGL to demonstrate the game of Brick Breaker and incorporates computer graphics concepts like transformation and translation.

**2.1. Viewing Functions**

**2.1.1 Position**

Positioning are interactions used in computer graphics. The positioning technique

is used to move the object from one position to another position.

**2.2 Transformation:**

Transformation means changing some graphics into something else by applying rules. We

can have various types of transformation such as translation, scaling, rotation, shearing etc.

**2.2.1 Translation**

Translation involves moving objects without changing their shape. It involves shifting every point or position by a consistent amount. When a straight line is translated, it remains straight and is drawn using its original endpoints. Similarly, when a polygon is translated, each of its vertices is shifted to a new position. The glTranslatef(dx, dy, dz) function is used to perform translations.

**3. User Defined Classes**

**3.1. class Bricks**

This class contains all the details about that specific block (we are using multiple blocks).

**4. User Defined Functions**

**4.1. createBricks()**

This function creates the bricks in the playing arena.

**4.2. print()**

This function prints the side menu for the score and chances.

**4.3. message()**

Prints the end game message on game over.

**4.4. completeMessage()**

Prints the message on completion of the stage.

**4.5. myDisplay()**

This is the complete display window of the game.

**4.6. checkCollision()**

This function checks if the ball and brick are in a collision state.

**4.7. moveBall()**

Move the ball continuously and call the calculation functions.

**4.8. keyboard()**

Takes the input from the keyboard and moves the bar accordingly.

**4.9. mouse()**

Takes the input from the mouse and moves the bar accordingly.

CODE:

#include<stdio.h>

#include<iostream>

#include<GL/glut.h>

#include<string>

#include<sstream>

using namespace std;

float

startBarX = 180, startBarY = 465, startBallX = 235, startBallY = 430, startBallVelX = 4,

startBallVelY = 4;

float barX = startBarX, barY = startBarY, barWidth = 120, barheight = 5;

float ballX = startBallX, ballY = startBallY, ballWH = 10, ballVelX =

startBallVelX, ballVelY = startBallVelY;

const int brickAmount = 100;

int score = 0, chances = 3, previousScore = 0, highestScore = 0;

bool flag = true, flag2 = true;

// this is the structure for the brick

// x,y are the position of the brick

// height and width is the height and the width of the brick

// isAlive defines a flag to define if the brick is already hit or not

class bricks {

public:

float x;

float y;

float width;

float height;

bool isAlive;

bricks()

{

x = y = width = height = 0;

isAlive = true;

}

};

bricks bricksArray[brickAmount];

// creates bricks in the playing area

// starting positon of row is 41,50

// there will be 10 bricks in one row

// total of 100 bricks (as calculated)

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, 255, 127);

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();

}

// this function prints the side menu for the score and chances

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]); //used to

//draw text with specified font

}

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]);

}

}

// prints the end game message on game over.

// a flag defines if the game has ended or not

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]);

}

}

// prints the message on completion of the stage

// if the user finishes all the blocks without loosing all the lifes

void completeMessage(bool a)

{

if (a == false)

{

glRasterPos2f(20, 400);

stringstream ss;

ss << score;

string s = "STAGE COMPLETE. Your score: 1000. Click to start new game.";

int len = s.length();

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

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12, s[i]);

}

}

// IMPORTANT

// this is the complete display window of the game

void myDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0);

//Bar (moved by the user)

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(10.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 760, 480, 0.0);

}

// this function checks of the ball and brick are in collision state

// ans returns the value in true or false

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;

return true;

}

// moves the ball continuously and call the calculation functions

// ballvel is the ball velocity

// made the reflection rules by just changing the symbol of velocity

void moveBall()

{

if (score == 300)

{

ballVelX = ballVelX \* 1.5;

ballVelY = ballVelX \* 1.5;

}

if (score >= 1000)

{

barX = startBarX;

barY = startBarY;

ballX = startBallX;

ballY = startBallY;

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;

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 = startBarX;

barY = startBarY;

ballX = startBallX;

ballY = startBallY;

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;

message(flag);

}

else

{

--chances;

ballX = startBallX;

ballY = startBallY;

if (ballVelY < 0)

ballVelY = -ballVelY;

}

}

if (checkCollision(ballX, ballY, ballWH, ballWH, barX, barY,

barWidth, barheight) == true)

ballVelY = -ballVelY;

}

glutPostRedisplay();

}

// takes the input from the keyboard and moves the bar accordingly.

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;

}

}

// takes the input from the mouse and moves the bar accordingly.

void mouse(int button, int state, int x, int y)

{

switch (button)

{

case GLUT\_LEFT\_BUTTON:

if (state == GLUT\_DOWN)

{

ballWH = 10;

flag = true;

if (ballVelX <= 0 && ballVelY <= 0)

{

ballVelX = startBallVelX;

ballVelY = startBallVelY;

}

glutIdleFunc(moveBall);

}

break;

case GLUT\_RIGHT\_BUTTON:

if (state == GLUT\_DOWN)

{

ballWH = 100;

}

break;

default:

break;

}

// Main function

}

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();

}

Output:

