**Coursework for Intro to Games Programming**

**Name: Raiyan Mehmood**

**Matriculation Number: S2030922**

**Course: Computer Games (Design)**

*Disclaimer:*

*I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award*.

*Signature: Raiyan M.*

**Table of Contents**

|  |  |
| --- | --- |
| Cover page | 1 |
| Table of contents | 2 |
| Code found challenging | 3 |
| Description of code (below) |  |
| Ball Script | 4 - 6 |
| Paddle script | 7 - 8 |
| Bricks script | 9 - 10 |
| Ball Spawner script | 11 |
| Main Menu script | 12 |

**Code found challenging**

An area of the code I found difficult in the coursework was creating perimeters for both the Bat and Paddle. To implement a solution for both paddle and bat, I gave the barriers, the paddle and bat, Colliders. This allowed for these objects to detect collisions, which I affirmed using a print statement to the console, which could tell me whether the ball and paddle could collide with each other and could also both collide with the barriers. Firstly, for the ball, I wanted it to ricochet off the barriers, however the ball would go through barriers as well as the paddle. To solve this, I made it that the value of directional X and Y components for the ball would become opposite (dependent on whether if its positive or negative ) when it hits the barriers and the paddle, using switch statements to compare the ball to appropriately named barriers, and paddle within the cases and execute an instruction which would turn the values to the opposite. This was successful in making the ball bounce of the barriers and the paddle.

The paddle could also detect collisions with both left and right barriers however it would also go directly through them. Similarly, I created a method (OnCollisionEnter2D) in the Paddle script which used switch statements to compare the paddle object to the barriers. I created two Boolean variables that would remain true until one of the cases were matched during collision. I was successful in stopping the paddle however this stopped the functioning of the paddle completely therefore I needed a way for the paddle to move again. To solve this, I created another method (OnCollisionExit2D) that contained a switch statement which would turn the two Boolean variables from false back to true again. This allowed the paddle to move again, but now within a set perimeter

Ball Script:

The ball script contains methods that are used for the movement, positioning, and defining what happens when the ball encounters the Bricks, Paddle, and the Barriers.

# Ball Code - Expand this section to see code (Further description of the code can be found in the internal commentary)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

// This class describes the movement of the ball and sets conditions for when it hits bricks and the barriers.

class ball : MonoBehaviour

{

public BallSpawner spawner ; // reference to BallSpawner script

public Bricks br; // reference to Bricks script

public Paddle paddle; // reference to paddle script

float size = 0.5f; // sets value for size

float speed = 0.115f; // sets value for speed

float directionX = 0.55f; // initial value for horizontal direction

float directionY = 0.55f; // initial value for vertical direction

new AudioSource audio; // reference to audiosource component

// Start is called before the first frame is updated

void Start()

{

audio = GetComponent<AudioSource>(); // gives audio variable the value of AudioSource component attached to Ball

}

// Update is called once every frame

public void Update()

{

Vector2 position = transform.localPosition; // Positions the ball to where it is positioned in Unity

position.x += speed \* directionX; // sets the x value of the component

position.y += speed \* directionY;// sets the y value of the component

transform.localPosition = position; //

}

// This method is called whenever the ball hits something. It is given the col parameter

//which contains information about the collision and what it collided with.

void OnCollisionEnter2D(Collision2D col)

{

// Switch statement which compares the gameobject to the cases named within the statement

// If there is a collision with one of the matched cases then the directives(commands) below the case will be carried out

//until break

switch (col.gameObject.name)

{

case "Paddle": // if the gamobject has a collision with anything named "Right barrier" "paddle"

case "Top barrier": // if the gameobject has a collision with anything named "Right barrier" "Top barrier"

directionY = -directionY; // if any cases are matched above, this changes the direction of the ball vertically

break; //ends statement

case "Left barrier":// if the gameobject has a collision with anything named "Left barrier"

case "Right barrier": // if the gameobject has a collision with anything named "Right barrier"

directionX = -directionX; // if any cases are matched above, this changes the direction of the ball horizontally

break; //ends statement

}

switch (col.gameObject.tag) // switch statment which compares the col gameobject with tags described within the cases

{

case "Brick": // if the gameobject has a collision with anything with tag "Brick"

directionY = -directionY; // Direction of ball changes vertically

br.IncreaseScore(); // Method is called from bricks class to increase the score by 1 after hitting "Brick"

audio.Play(); //Plays audio if ball hits "brick"

br.BrickArray(); // Method is called from bricks class to subtract a brick from the brick array

break; //ends statement

case "Bottom barrier": // if the gameobject has a collision with anything with tag "Bottom barrier"

br.Updatelives(); // Method is called from bricks class to decrease lives by 1 after hitting "Bottom barrier"

spawner.SpawnBall(this); // Method is called from Ball Spawner class which respawns ball back

break;//ends statement

}

}

public void SetDirection(int angleInDegrees) // Method which sets the direction for the ball. It is given the 'angleInDegrees' parameter

{

float angleInRadians = angleInDegrees \* Mathf.Deg2Rad; //converts angle in degrees to angle in radians

directionX = Mathf.Cos(angleInRadians);// Sets the variable to the horizontal componnent using the Cos Function

directionY = Mathf.Sin(angleInRadians); // Sets the variable to the vertcial componnent using the Sin Function

}

}

Paddle script:

The paddle script contains methods for the movement, positioning, and defines what happens when the paddle collides with the barriers and the ball.

# Paddle Code - Expand this section to see code ((Further description of the code can be found in the internal commentary)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

// This class describes movement, positioning and defines perimeters for the paddle

class Paddle : MonoBehaviour

{

float speed = 0.15f; //variable for paddle speed

float direction = 0.0f; //variable for paddle direction

public KeyCode moveRightKey = KeyCode.RightArrow; //variable given the value of the right arrow key

public KeyCode moveLeftKey = KeyCode.LeftArrow; //variable given the value of the left arrow key

bool PositiveMotion = true; //variable to allow movement to the right

bool NegativeMotion = true; // //variable to allow movement to the left

public Bricks br; // reference to bricks script

// Method which sets the paddle's position

void FixedUpdate ()

{

Vector2 position = transform.localPosition; // Sets the position of paddle to the chosen position within Unity

position.x += speed \* direction; //sets x value of the component

transform.localPosition = position;//

}

//Method which gives the paddle movement if conditions are met (if keys are pressed)

public void Update()

{

bool RightKey = Input.GetKey(moveRightKey); //Gives local variable the value as the right arrow key input

bool LeftKey = Input.GetKey(moveLeftKey);//Gives local variable the value as the left arrow key input

if (RightKey && PositiveMotion)// if both conditions are met (if right key is pressed)

{

direction = 1.0f; // paddle moves to the right

}

else if (LeftKey && NegativeMotion) // else if both conditions are met (if left key is pressed)

{

direction = -1.0f; // paddle moves to the left

}

else

{

direction = 0.0f; // if no button pressed, paddle remains stationary

}

}

void OnCollisionEnter2D(Collision2D col)//Method for the paddle to restrict movement beyond the barriers when it hits them

{

switch (col.gameObject.name) //switch statement which compares gameobject to the values described within the case

{

case "Right barrier": // if paddle hits the right barrier

PositiveMotion = false; // , movement to the right is restrcited

break;//end statment

case "Left barrier": // if paddle hits Left barrier

NegativeMotion = false; // , movement to the left is restricted

break; // end statement

}

}

void OnCollisionExit2D(Collision2D other) // Method which allows paddle to move again after hitting the barriers( creates perimeter for paddle )

{

switch (other.gameObject.name) //switch statement which compares gameobject to the values described within the case

{

case "Right barrier":

PositiveMotion = true; // allows for movement to the right again

break; // end statement

case "Left barrier":

NegativeMotion = true; // allows for movement to the left again

break; // end statement

}

}

}

Bricks Script:

This script contains methods for how variables such as dynamic variables such as lives, score, and the number of bricks and how they are affected as well as the game object brick itself when they meet conditions.

# Bricks code – Expand this section to see code (Further description of the code can be found in the internal commentary)

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

// This class describes how collision with the brick affect other components such as lives, score

//and defines what happens when it comes into contact with the ball

class Bricks : MonoBehaviour

{ //Declaration all of variables

public ball Ball; //reference to ball script

int lives = 3; // variable to track lives

public int score; // variable to track score

public Text scoreLabelText; // variable for text for score

public Text livesText; // variable for text for lives

bool PlayGame = true; // variable for if the game is in play

public int bricknumber; // Varaible to track number of bricks

public Text GameOver = null; // Variable for Gameover Text

public Paddle Paddle; // Reference to paddle script

BallSpawner spawn; // Refrence for bBallSpawner Script

// This method is carried out before the first frame

public void Start()

{

bricknumber = GameObject.FindGameObjectsWithTag("Brick").Length; // Gives the variable "bricknumber" the value of the number of bricks within the array

livesText.text = lives.ToString("Lives: " + lives); // Converts the text to string message

scoreLabelText.text = score.ToString("Score: " + score); // Converts the text to string message

}

public void Updatelives() //Method which deducts lives from the player

{

lives--; // Decrements the number of lives by 1

livesText.text = lives.ToString(" Lives: " + lives); // Lives Text is updated on screen, showing the new value for the variable 'lives'

{

if (lives == 0) // if the amount of lives reaches 0,

gameOver(); //the gameOver method is called and executed(Ends the game)

}

}

public void IncreaseScore() // method which updates the player's score

{

score++; //increments the score by 1

scoreLabelText.text = score.ToString("Score: " + score); // score Text is updated on screen, showing the new value for the variable 'score'

{ if (score == 32) // sets a condition if the score is equal to 32

gameOver();//the gameOver method is called and executed(Ends the game)

}

}

public void BrickArray()// method which is called to detract bricks from an array

{

bricknumber--; //decrements 1 brick each time this method is called

{

if (bricknumber == 0) // if the number of bricks reaches zero,

gameOver(); //the gameOver method is called and executed(Ends the game)

}

}

public void gameOver() // Method to end the game

{

PlayGame = false; // Sets the value to false making the game end

GameOver.gameObject.SetActive(true); // Game Over message is activated and shown on screen

spawn.gameObject.SetActive(false); //Makes the ball inactive to siginify end game

Paddle.gameObject.SetActive(false); // Makes the paddle inactive to siginify end game

}

public void OnCollisionEnter2D(Collision2D other)

// Method which detects collisions for the brick object

//with the parameters set to 'other' which contains information about the collisions

{

if (other.gameObject.tag == "Ball") // makes a collision with the Brick(s) if the condition in which gameObject with the Tag "Ball" is fulfilled

BrickArray();// Method is called to subtract 1 brick from the brick array

Destroy(gameObject);// Destroys Brick object the ball has a collision with

}

}

Ball spawner script:

The BallSpawner script allows the ball gameobject to be respawned by creating copies of it and giving the clones the same functionality.

# BallSpawner code – Expand this section to see code (Further description of the code can be found in the internal commentary)

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

// This class describes an object which can spawn ball gameobjects by copying

// a template of the ball gameobject which is already within the scene .

class BallSpawner : MonoBehaviour

{

//Declaration of variables

// Sets balltemplate to the existing ball object

public ball ballTemplate;

// The number of balls that are spawned initially is 1

int ballnumber = 1;

public Bricks br; // reference to bricks script

// This method spawns the ball template

void Start()

{

// Spawns ball using the parameter "ball template"

SpawnBall(ballTemplate);

}

public void SpawnBall(ball template) // Method which creates a clone of the ball object

{

// Creats a clone of the template gameobject

ball Clone = Instantiate(template);

// Moves the ball clone to the spawner's position

Clone.transform.position = transform.position;

// Gives the ball clone a random direction within a certain range of numbers

int angle = Random.Range(-40,-180);

//calls the setdirection method from ball class and sets the clone's direction to the angle

Clone.SetDirection(angle);

// Activates the ball clone in Unity

Clone.gameObject.SetActive(true);

}

}

Main menu script:

The main menu script allows for the game scene to loaded via the main menu

# Main Menu code – Expand this section to see code (Further description of the code can be found in the internal commentary)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

// This class allows to load the main from the main menu

public class Main : MonoBehaviour

{

public void Startgame() // Method which is called when at the main menu to start the game

{

SceneManager.LoadScene("BrickOut Game"); //Loads the scene with the game

}

}