ECE 270: Computer Methods in ECE

Audible Brick Breaker

Final Project

Mohamed El-Abdallah

December 10, 2022

1 Statement of the Problem

For my final project I decided to make the retro game Brick Breaker or what was once knows as Breakout from the Atari. I thought this game out have been good to make because it includes 2D sampling in both the x and y direction. It also would have to include moving objects, changing vectors between intersections as well as complex nested for loops.

2 Description of Solution

2.1 Object Settings

For the starting vector for the moving ball I made the program choose a random number between -15 and 15. This means that the ball will go in a random direction and random speed to trick the user upon start up of the game so the user does not get used to the direction.

For the Ball and Platform I wanted to start them out in the center of the screen so that it gives the user a fair chance to catch the ball going in either direction.

```
/*-----------BALL VECTOR SETTINGS------------*/

negvelmin = -15;
posvelmin = 15;
velrecx = negvelmin + rand() % posvelmin - negvelmin + 1;
velrecy = negvelmin + rand() % posvelmin - negvelmin + 1;

/*-------BALL STARTING POSITION AND SIZE-----------*/

Ball.set(500, 300, 15, 15);

/*-------PLATFORM SIZE AND POSITION PARAMETERS-----------*/

Platform.set(500, 700, 200, 25);
```

2.2 2D Sampling

The sampling for this final had to be done in both the x and y directions. I used the same equation for both directions and it is the famous equation we have been using all semester

$$nsamples = \frac{max - min}{step} + 1$$

The above equations is used to calculate the amount of samples that will be made.

I then take this amount and use it for the max number of loops for my for loop that will store the x and y coordinates of my bricks.

In a later for loop I assign these arrays of x and y to a matrix called $Bricks$ of the $ofRectangle$ class. From this now we have our $Bricks$ for Brick Breaker

```
/*---------2D X AND Y SAMPLING----------*/

nx = (xmax - xmin) / xstep + 1;

for (int j = 0; j < nx; j++)
{
    x[j] = xmin + j * xstep;
}

ny = (ymax - ymin) / ystep + 1;

for (int i = 0; i < ny; i++)
{
    y[i] = ymin + i * ystep;
}

/*---------COLOR AND POSITION SETTING----------*/

for (int i = 0; i < ny; i++)
{
    for (int j = 0; j < nx; j++)
    {
        Matrix[i][j] = 1;
        BrickColors[i][j].set(rand() % 255, rand() % 255, rand() % 255);

        Bricks[i][j].set(x[j], y[i], 50, 50);
    }
```

```
}
```

2.3 Ball Physics

To start the movement of the ball the $Ball.x$ and $Ball.y$ will recursively take the sum of the randomly generated $velrecx$ and $velrecy$ and its initial starting points.

From there we need to test for boundary intersections and what we want to do with said intersections. I decided it would be the most simple to make four if statements. One for each boundary. How I have these physics working is if it hits either the left or right side of the ball then it will invert the direction of the x velocity. Then I have the ball changing its y velocity of intersecting the top boundary.

Now for the bottom boundary. I wanted to include a way that the user knew they had lost. Once the user allows or misses the ball and it reaches the bottom boundary, all movement ceases and the user is required to restart the program to restart.

Then we have the ball bouncing off of the platform. I did this by using the $,inside$ function in the ofRectangle class. I then had to set this as a Boolean variable which will detect if the ball would be inside the platform in the next frame then it would make my Boolean variable equal to 1. Then an if statement would run to invert the y velocity to change direction.

```
    /*-----------BALL PHYSICS---------------*/

Ball.x = Ball.x + velrecx;
Ball.y = Ball.y + velrecy;

if (Ball.x + Ball.width > 1024 && velrecx > 0)
{
    velrecx = -velrecx; //hits right bound, reverse direction
}

if (Ball.x < 0 && velrecx < 0)
{
    velrecx = -velrecx; //hits left bound, reverse direction
```

```
}

if (Ball.y + Ball.height > 768 && velrecy > 0)
{
    velrecy = 0; //hits lowest bound, stop all movement
    velrecx = 0;
}

if (Ball.y < Ball.height && velrecy < 0)
{
    velrecy = -velrecy; //hits highest bound, reverse direction
}

b = Platform.inside(Ball.x, Ball.y + Ball.height);
if (b == 1)
{
    velrecy = -velrecy; //hits platform, reverse direction
}
```

2.4 Intersections Between Bricks and Ball

Calculating and finding a way to remove the bricks when the ball intersects them was the hardest part of this game. The way I have done this was making a matrix named *Matrix* the same size as the *Bricks* matrix and set every element in *Matrix* equal to 1. Then I created a Boolean flag variable that will help me sense intersections. I needed all of these to be in a nested *for* loop to count up to the element of the matrix that was intersected. Inside of all of that was bool variable. If it senses an intersection between the ball from any side and the Brick, and the *Matrix* brick is still visible. It will run a *if* statement that will turn that element in *Matrix* to 0 to show it is not visible anymore, will change the color to black to match the background, will reverse the y velocity, and will play the sound I have recorded. For this to work over and over again I needed to return the flag variable back to *true* once the loop would finish.

```
 /*-------INTERSECTIONS BALL TO BRICK----------*/

bool vflag = false;
```

```
for (int i = 0; i < ny; i++)
{
    for (int j = 0; j < nx; j++)
    {
        I = Bricks[i][j].intersects(Ball);
        if (I == 1 && Matrix[i][j] == 1)
        {
            Matrix[i][j] = 0;
            BrickColors[i][j].set(0,0,0);
                if (!vflag)
                {
                    velrecy = -velrecy;
                    Watta.play();
                }
            vflag = true;
        }
    }
}
```

2.5 Drawings

The drawing section of my project is easily the most simple part of this project. I only needed to nest a *for* loop to draw the matrix of bricks and its respected colors. After the loop I then drew the ball and platform objects setting both to a white color.

```
for (int i = 0; i < ny; i++)
{
    for (int j = 0; j < nx; j++)
    {
        ofSetColor(BrickColors[i][j]);
        ofDrawRectangle(Bricks[i][j]);
    }
}

ofSetColor(255, 255, 255);
ofDrawRectangle(Platform);
ofDrawRectangle(Ball);
```

2.6 Key-Pressing

Brick Breaker would not be a game if the user could not control the platform to aim the ball. For this it was only two nested if that needed to be made. Every key press of the left or right arrow key will move the platform 100 pixels at a time. However if we only did this the user would be able to move the platform off of the screen to fix these I made it so if the left side is less than the pixel amount of the left side of the screen it will keep the platform at 0. Then if the x coordinate plus the width of the platform is greater than the right side of the screen it will then keep the x coordinate at the length of the screen minus the width of the platform.

```
/*---------PLATFORM INCREMENTATION----------*/
 if (key == OF_KEY_LEFT)
 {
     Platform.x = Platform.x - 100;

     if (Platform.x <= 0)
     {
         Platform.x = 0;
     }
 }
 if (key == OF_KEY_RIGHT)
 {
     Platform.x = Platform.x + 100;

     if (Platform.x + Platform.width >= 1024)
     {
         Platform.x = 824;
     }
 }
```

2.7 Sound

For this I used the class $ofSoundPlayer$ that is built into OpenFrameWorks. I then just recorded an audio clip of Professor Watta saying "Where is your notebook?" and I will have that play every time the ball is intersecting a brick.

```
    ofSoundPlayer Watta;
    .
    .
    .
     /*----------SOUND SETUP---------------*/

    Watta.load("watta2.mp3");
    Watta.setVolume(1);
    .
    .
    .
    Watta.play();
```

3 Code

```
    #pragma once

#include "ofMain.h"

#define N_MAX 500

class ofApp : public ofBaseApp {
public:
   void setup();
   void update();
   void draw();

   void keyPressed(int key);
   void keyReleased(int key);
   void mouseMoved(int x, int y);
   void mouseDragged(int x, int y, int button);
   void mousePressed(int x, int y, int button);
   void mouseReleased(int x, int y, int button);
   void mouseEntered(int x, int y);
   void mouseExited(int x, int y);
   void windowResized(int w, int h);
   void dragEvent(ofDragInfo dragInfo);
   void gotMessage(ofMessage msg);

   float velrecx;
   float velrecy;

   float recvy;
```

```
    float recvx;

    float nx;
    float xmin;
    float xmax;
    float xstep;

    float ny;
    float ymin;
    float ymax;
    float ystep;

    int negvelmin;
    int posvelmin;

    bool b;
    bool I;
    bool Matrix[100][100];

    ofSoundPlayer Watta;
    ofSoundPlayer Lose;
    ofSoundPlayer Suck;

    float x[N_MAX];
    float y[N_MAX];

    ofRectangle Bricks[100][100];
    ofRectangle Platform;
    ofRectangle Ball;
    ofColor BrickColors[100][100];

};

#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup(){

    srand(time(NULL));
    ofSetBackgroundColor(0, 0, 0);

    /*-----------BRICK SETTINGS-----------*/

    xmin = 0;
    xmax = ofGetScreenWidth();
    xstep = 50;
```

```
ymin = 50;
ymax = 150;
ystep = 50;

/*-----BALL VECTOR SETTINGS------------*/

negvelmin = -15;
posvelmin = 15;
velrecx = negvelmin + rand() % posvelmin - negvelmin + 1;
velrecy = negvelmin + rand() % posvelmin - negvelmin + 1;

/*-------BALL STARTING POSITION AND SIZE-----------*/

Ball.set(500, 300, 15, 15);

/*-------PLATFORM SIZE AND POSITION PARAMETERS-----------*/

Platform.set(500, 700, 200, 25);

/*----------SOUND SETUP---------------*/

Watta.load("watta2.mp3");
Watta.setVolume(1);

/*---------2D X AND Y SAMPLING----------*/

nx = (xmax - xmin) / xstep + 1;

for (int j = 0; j < nx; j++)
{
    x[j] = xmin + j * xstep;
}

ny = (ymax - ymin) / ystep + 1;

for (int i = 0; i < ny; i++)
{
    y[i] = ymin + i * ystep;
}

/*---------COLOR AND POSITION SETTING----------*/

for (int i = 0; i < ny; i++)
{
    for (int j = 0; j < nx; j++)
    {
        Matrix[i][j] = 1;
```

```
            BrickColors[i][j].set(rand() % 255, rand() % 255, rand() % 255);

            Bricks[i][j].set(x[j], y[i], 50, 50);
        }
    }
}

//--------------------------------------------------------------
void ofApp::update(){

    /*-----------BALL PHYSICS---------------*/

    Ball.x = Ball.x + velrecx;
    Ball.y = Ball.y + velrecy;

    if (Ball.x + Ball.width > 1024 && velrecx > 0)
    {
        velrecx = -velrecx; //hits right bound, reverse direction
    }

    if (Ball.x < 0 && velrecx < 0)
    {
        velrecx = -velrecx; //hits left bound, reverse direction
    }

    if (Ball.y + Ball.height > 768 && velrecy > 0)
    {
        velrecy = 0; //hits lowest bound, stop all movement
        velrecx = 0;
    }

    if (Ball.y < Ball.height && velrecy < 0)
    {
        velrecy = -velrecy; //hits highest bound, reverse direction
    }

    b = Platform.inside(Ball.x, Ball.y + Ball.height);
    if (b == 1)
    {
        velrecy = -velrecy; //hits platform, reverse direction
    }

    /*-------INTERSECTIONS BALL TO BRICK----------*/
    bool vflag = false;
    for (int i = 0; i < ny; i++)
    {
        for (int j = 0; j < nx; j++)
```

```
        {
            I = Bricks[i][j].intersects(Ball);
            if (I == 1 && Matrix[i][j] == 1)
            {
                Matrix[i][j] = 0;
                BrickColors[i][j].set(0,0,0);
                    if (!vflag)
                    {
                        velrecy = -velrecy;
                        Watta.play();
                    }
                vflag = true;

            }
        }
    }

}

//--------------------------------------------------------------
void ofApp::draw(){

    for (int i = 0; i < ny; i++)
    {
        for (int j = 0; j < nx; j++)
        {
            ofSetColor(BrickColors[i][j]);
            ofDrawRectangle(Bricks[i][j]);
        }
    }

    ofSetColor(255, 255, 255);
    ofDrawRectangle(Platform);
    ofDrawRectangle(Ball);
}

//--------------------------------------------------------------
void ofApp::keyPressed(int key){

    /*---------PLATFORM INCREMINATION----------*/
    if (key == OF_KEY_LEFT)
    {
        Platform.x = Platform.x - 100;

        if (Platform.x <= 0)
        {
            Platform.x = 0;
```

```
        }
    }
    if (key == OF_KEY_RIGHT)
    {
        Platform.x = Platform.x + 100;

        if (Platform.x + Platform.width >= 1024)
        {
            Platform.x = 824;
        }
    }

}
```