ECE 270: Computer Methods in ECE

DEARBORN

Final Project

Centipede

Joel Bhattarai

July 11, 2023

1 Statement of the Problem

In the game Centipede, there are a variety of enemies and obstacles that the player can shoot and eliminate for points. The main enemy is the Centipede who snakes its way down the screen. If the player shoots one segment of the Centipede, that segment will be eliminated and a mushroom will be generated in its spot, which serves to separate the Centipede. Next, are the mushrooms, 60 mushrooms are generated upon starting a new game, but mushrooms are also generated throughout the game by other enemies. Next is the spider, the spider bounces around the area that the player can move in. Every time the player kills the spider the spider comes back sooner, maxing out at a respawn time of 3 seconds. Finally, is the flea, the flea falls down the screen generating mushrooms on his trail as he falls. He, too, comes back sooner every time you kill him but there is no maximum like the spider, so eventually he respawns instantly. He also respawns instantly if you fail to eliminate him. Every one of these enemies kills the player on contact, including the mushrooms, which is one way that my version varies from the original arcade game, as in the original game, the mushrooms are non-lethal. The goal of the player in this game is to stay alive as long as possible and earn as many points as possible.

2 Description of Solution

To create this game, I generated different classes for each entity in the game. I then performed the necessary actions in ofApp.cpp. Following the inclusion of every new class I handled the interaction of the new class with every existing class in the update section.

2.1 Player Class

The player class should simply read in the mouse location and draw the sprite on that location. For the functions, I simply need to call setup, update, and draw in the second gamestate so that they only get used when the game is being played. In setup I simply load the image and initialize the players location and size. For draw, I must use the dimensions to draw an image.

```
#include "Player.h"

Player::Player()

{

}

void Player::setup()

{

playerImage.load("../../images/player.jpg");

x=ofGetWidth()/2;

y=ofGetHeight() / 1.125;

width=10;

height=20;

}

void Player::update()

{

}

void Player::draw()

{

ofSetColor(255, 255, 255);

playerImage.draw(x,y,width,height);

}
```

There will be nothing in update since the event handler updates the rectangles position. The event handler reads in the mouse's location up to a point so that the player can only move so far up.

```
void ofApp::mouseMoved(int x, int y){

if (y > 700)
{
player.y = y;
}
else {
player.y = 700;
}
if (x < 1000)
{
player.x = x - player.width / 2;
}
else
{
player.x = 990;
}
}
```

Following is the UML Diagram for this class.

```
---------------------------------
Player
---------------------------------
+x:int
+y:int
+width:int
+height:int
+playerImage:ofImage
---------------------------------
Player()
setup():void
update():void
```

```
draw():void
intersects(other:ofRectangle):bool
---------------------------------
```

2.2 Tree Class

The tree class is used for the mushrooms in the game. This is because at first I did not realize that they were mushrooms. The tree class only needs to store a rectangle and all four sprites for the damaged mushrooms. The interesting parts of this class are the if statements that determine which sprite should be used in the draw function. I cannot initialize the location because the setup function must be used later for the new mushrooms being created.

```
#include "Tree.h"
22
Tree::Tree()
{
}
void Tree::setup()
{
hp4.load("../../images/hp4.jpg");
hp3.load("../../images/hp3.jpg");
hp2.load("../../images/hp2.jpg");
hp1.load("../../images/hp1.jpg");
hitpoints = 4;
width = 10;
height = 10;
}
void Tree::update()
{
}
void Tree::draw()
{
if (hitpoints == 4)
{
hp4.draw(x, y,width,height);
}
else if (hitpoints == 3)
{
hp3.draw(x, y,width,height);
}
else if (hitpoints == 2)
{
hp2.draw(x, y,width,height);
}
else if (hitpoints == 1)
{
hp1.draw(x, y,width,height);
}
else if (hitpoints == 0)
{
x = 10000;
}
}
```

As for the ofApp.cpp file's update section, all we must do is check each frame if they intersect a player and if so we change the gamestate to end the game.

```
for (int i = 0; i < treecount; i++)
{
```

```
trees[i].update();

if (player.intersects(trees[i])) {

die.play();

gamestate = 3;

}

}
```

Following is the UML Diagram for the tree class.

```
---------------------------------

Tree

---------------------------------

+x:int

+y:int

+width:int

+height:int

+hp4:ofImage

+hp3:ofImage

+hp2:ofImage

+hp1:ofImage

---------------------------------

Tree()

setup():void

update():void

draw():void

intersects(other:ofRectangle):bool

---------------------------------
```

2.3 Projectile Class

The projectile class must be spawned from the location of the mouse, this will be handled in the mousePressed eventhandler. All we must do in the setup is set the velocity, width, and height. For update, simply keep the motion going and for draw I did not use an image this time because the projectile is only a rectangle so I just drew a red rectangle.

```
#include "Projectile.h"
Projectile::Projectile()
{
}
void Projectile::setup()
{
vy = -10;
width = 2;
height = 10;
}
void Projectile::update()
{
y = y + vy;
}
void Projectile::draw()
{
ofSetColor(255, 0, 0);
ofDrawRectangle(x,y,width,height);
}
```

As for in ofApp.cpp's update section, I must check if the projectiles intersect with any trees and if they do remove the projectile and decrease the hitpoints of the tree.

```
for (int i = 0; i < N_MAX; i++)
```

```
{
projectiles[i].update();
}
for (int i = 0; i < projectileCount; i++)
{
for (int j = 0; j < treecount; j++)
{
if (projectiles[i].intersects(trees[j])) {
hit.play();
score = score + 10;
trees[j].hitpoints = trees[j].hitpoints - 1;
projectiles[i].x = 10000000000000000000;
}
}
}
```

For the starting position of the projectile, this is handled in the second gamestate of the mousePressed event handler. Although we are reading the mouse's location when pressed, we want the projectiles to come from the player so the coordinates must be similarly restricted.

```
if (gamestate == 2)
{
shoot.play();
if (x < 1000) {
projectiles[projectileCount].x = x;
}
else {
projectiles[projectileCount].x = 1000;
}
if (y > 700) {

projectiles[projectileCount].y = y;

}

else {

projectiles[projectileCount].y = 700;

}

projectileCount = projectileCount + 1;

}
```

Following is the UML Diagram of the projectile class.

```
---------------------------------

Projectile

---------------------------------

+x:int

+y:int

+width:int

+height:int

+vy:int

---------------------------------

Projectile()

setup():void

update():void

draw():void

intersects(other:ofRectangle):bool

---------------------------------
```

2.4 Spider Class

The spider class must create the spider on the screen and move him around the bottom section of the screen. The setup section will simply initialize the values and load the image. Update will move him depending on his velocity while restricting him to the bottom of the screen. Finally, draw will draw the image on the screen.

```
#include "Spider.h"

Spider::Spider()

{

}

void Spider::setup()

{

vy = -5;

vx = 5;

x = 900;

y = 900;

width = 25;

height = 25;

spiderImage.load("../../images/spider.jpg");

}

void Spider::update()

{

x = x + vx;

y = y + vy;

if (x>975||x<0) {

vx = vx * -1;

}

if (y>975||y<675) {

vy = vy * -1;
}
}
void Spider::draw()
{
spiderImage.draw(x, y, width, height);
}
```

For ofApp.cpp's update, I must update the spider, check if he intersects the player, if so end the game, and then check if he intersects any projectiles. If so kill the spider and move away the projectile. If the spider dies, I capture the time that it died and then I check if the difference between current time and the death of the spider is greater than the respawn time. If yes, I respawn the spider. Also, I continually decrease the respawn time of the spider every time it dies so he comes back sooner. I only do this up to 3000 milliseconds or 3 seconds because it would become instant which I thought would be too fast.

```
spider.update();
if (spider.intersects(player))
{
die.play();
gamestate = 3;
}
for (int i = 0; i < projectileCount; i++)
{
if (projectiles[i].intersects(spider)) {
hit.play();
score = score + 100;
spider.x = -1000000000;
spider.width = 0;
projectiles[i].x = -1000000000000000;
spiderDeathTime = ofGetElapsedTimeMillis();
if (spiderRespawnTime > 3000) {
spiderRespawnTime = spiderRespawnTime - 1000;
}
}
}
if (spider.width==0) {
if (ofGetElapsedTimeMillis()-spiderDeathTime>spiderRespawnTime) {
spider.setup();
}
}
```

Following is my UML Diagram for my spider class.

```
---------------------------------
Spider
---------------------------------
+x:int
+y:int
+width:int
+height:int
+vy:int
+vx:int
+spiderImage:ofImage
---------------------------------
Spider()
setup():void
update():void
draw():void
intersects(other:ofRectangle):bool
```

```
---------------------------------
```

2.5 Centipede Class

The centipede class will create one section of the centipede and it will snake its way down the screen bouncing off the walls and trees and killing the player when they touch. The Setup section will simply initialize the velocity width height and load the image. The update section will change the position and make the centipede bounce off walls and kill it if it goes below the screen. Width is set to zero to easily check whether or not the centipede is alive. Finally, draw simply draws the centipede section.

```
#include "Centi.h"
Centi::Centi()
{
}
void Centi::setup()
{
vx = 10;
width = 10;
height = 10;
centiImage.load("../../images/centi.jpg");
}
void Centi::update()
{
x = vx + x;
if (x>=1000||x<=0) {
vx = vx * -1;
y = y + 10;
}
if (y >= 1000)
{
x = -10000000;
width = 0;
}
}
void Centi::draw()
{
ofSetColor(255, 255, 255);
centiImage.draw(x, y, width, height);
}
```

For the update section of ofApp.cpp for the centipede, we must update every section, check if any of them interact with trees and if they do we will bump the section down 10 pixels and invert its velocity. Next, we check if any centipede section interacts with the player. If it does, we change the gamestate. Then, if we see an intersection between the centipede and any projectile we have to kill the centipede move away the projectile and create a mushroom at that spot. Finally, every cycle we check how many centipedes have no width meaning they are dead. If all ten of them are dead we create a new centipede.

```
for (int i = 0; i < centiCount; i++)
{
myCentipede[i].update();
}
for (int i = 0; i < centiCount; i++)
20
{
for (int j = 0; j < treecount; j++)
{
```

```
if (myCentipede[i].intersects(trees[j])) {
myCentipede[i].vx = myCentipede[i].vx * -1;
myCentipede[i].y = myCentipede[i].y + 10;
}
}
}
for (int i = 0; i < centiCount; i++)
{
if(myCentipede[i].intersects(player)) {
die.play();
gamestate = 3;
}
}
for (int i = 0; i < centiCount; i++)
{
for (int j = 0; j < projectileCount; j++)
{
if (myCentipede[i].intersects(projectiles[j]))
{
hit.play();
score = score + 100;
trees[treecount].setup();
trees[treecount].x = myCentipede[i].x;
trees[treecount].y = myCentipede[i].y;
myCentipede[i].x = -10000000;
myCentipede[i].width = 0;
projectiles[j].x = 10000000;
treecount = treecount + 1;
}
```

```
}
}
int deadCentipedes=0;
for (int i = 0; i < centiCount; i++)
{
if (myCentipede[i].width==0)
{
deadCentipedes = deadCentipedes + 1;
}
}
if (deadCentipedes == 10)
{
for (int i = 0; i < centiCount; i++)
{
myCentipede[i].setup();
21
myCentipede[i].x = (10 * i) + 10;
myCentipede[i].y = 0;
}
}
```

Following is my centipede class's UML Diagram.

```
---------------------------------
Centi
---------------------------------
+x:int
+y:int
+width:int
+height:int
```

```
+vx:int

+centiImage:ofImage

---------------------------------

Centi()

setup():void

update():void

draw():void

intersects(other:ofRectangle):bool

---------------------------------
```

2.6 Flea Class

The flea class will store a y velocity and an image as well as the usual rectangle dimensions. The setup function for the flea will initialize this velocity, the width, height and coordinates. It will set the flea to start at $y = 0$ and a random x. Update will simply move the flea according to the velocity, and draw will simply draw the flea image at the dimensions created by the rectangle.

```
#include "Centi.h"

Centi::Centi()

{

}

void Centi::setup()

{

vx = 10;

width = 10;

height = 10;

centiImage.load("../../images/centi.jpg");

}
```

```
void Centi::update()
{
x = vx + x;
if (x>=1000||x<=0) {
vx = vx * -1;
y = y + 10;
}
if (y >= 1000)
{
x = -10000000;
width = 0;
}
}
void Centi::draw()
{
ofSetColor(255, 255, 255);
centiImage.draw(x, y, width, height);
}
```

In the update section, we will first call the flea's update section. Then, I will check if the flea intersects the player, if he does I will change the gamestate. Next I check if any projectile intersects the flea, if so both are moved out of sight. Then I check if the flea has reached the bottom without being shot. If it has I instantly bring him back. After that, I check if the flea is moved out by being shot. If so, I run a similar time check to with the spider to see if it should be brought back yet. The only difference is that this timer doesn't max out at any amount of seconds. If you kill the flea enough it will come back instantly. Lastly, I roll a chance to see if the flea will produce a tree during that cycle and if it does I create the tree.

```
flea.update();
```

```
if (flea.intersects(player))
{
gamestate = 3;
die.play();
}
for (int i = 0; i < projectileCount; i++)
{
if (flea.intersects(projectiles[i]))
{
projectiles[i].x=-100000000000;
flea.x=-100000000;
hit.play();
score = score + 50;
fleaDeathTime = ofGetElapsedTimeMillis();
fleaRespawnTime = fleaRespawnTime - 500;
}
}
if (flea.y > 1000&&flea.x>0)
{
flea.setup();
}
if (flea.x < -1000)
{
if (ofGetElapsedTimeMillis() - fleaDeathTime >= fleaRespawnTime) {
flea.setup();
}
}
flearoll = 1 + rand() % 40;
if (flearoll == 3) {
```

```
treecount = treecount + 1;

trees[treecount].setup();

trees[treecount].x = flea.x;

trees[treecount].y = flea.y;

}
```

The following is my UML Diagram for the flea class.

```
---------------------------------

Flea

---------------------------------

+x:int

+y:int

+width:int

+height:int

+vy:int

+fleaImage:ofImage

---------------------------------

Flea()

setup():void

update():void

draw():void

intersects(other:ofRectangle):bool

---------------------------------
```

---

3 Testing and Output

For testing the game, not only did I play the game a huge amount of times myself, throughout the development of the project, I also had members of my family, such as my brother and dad, playtest the game for me. Through my dad's play testing I was able to discover a bug that I never would've seen myself. As an avid player of video games myself, I find it very easy to shoot down the flea before it plants too many mushrooms, my dad, however, was unable to, and I realized I hadn't addressed the case of the player missing the flea entirely. It was an easy fix but as a result of my wide player sample I was able to fix a major bug.

My game has 3 different game states, the welcome screen, the gameplay screen, and the game over screen. The first one is the welcome screen which has two options. One is to move to the gameplay screen and the other is to exit out of the game. In the bottom right there is just a note to stay out of fullscreen mode as the display will be distorted.

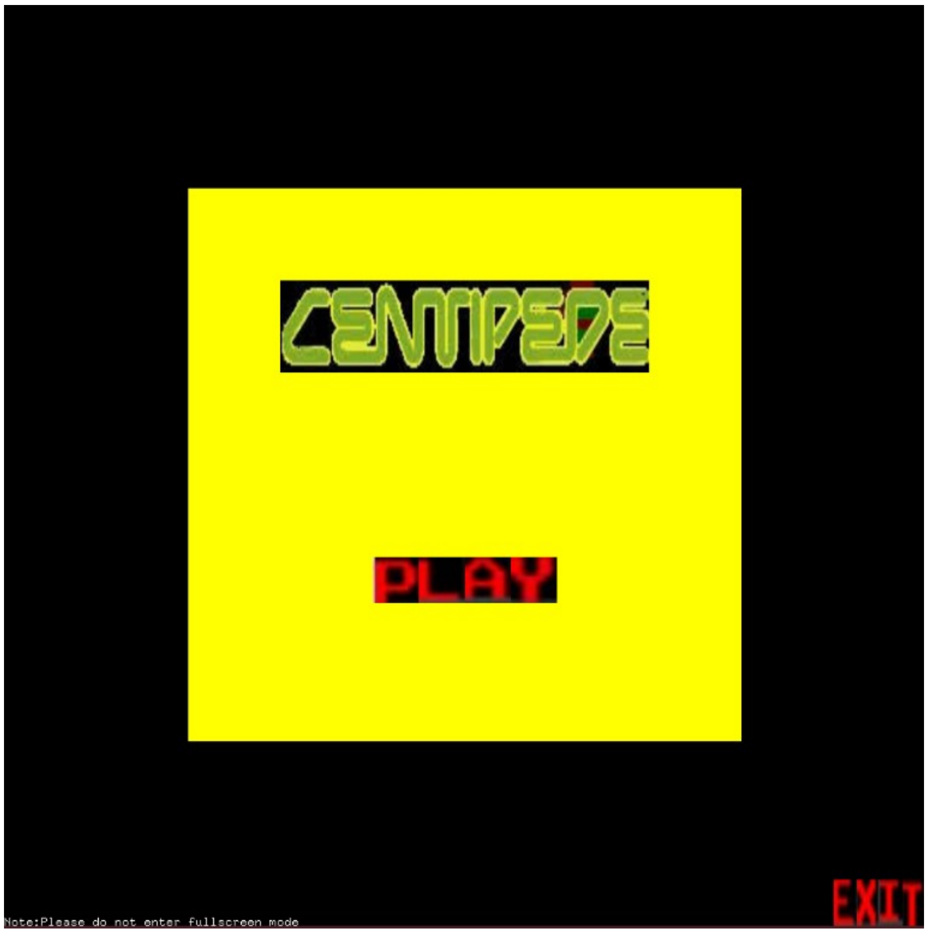

Figure 1: Welcome Screen

The game screen is where the gameplay occurs. This is where I load in all of the enemies, structures and the player model. There is also a running counter of the score in the top left.

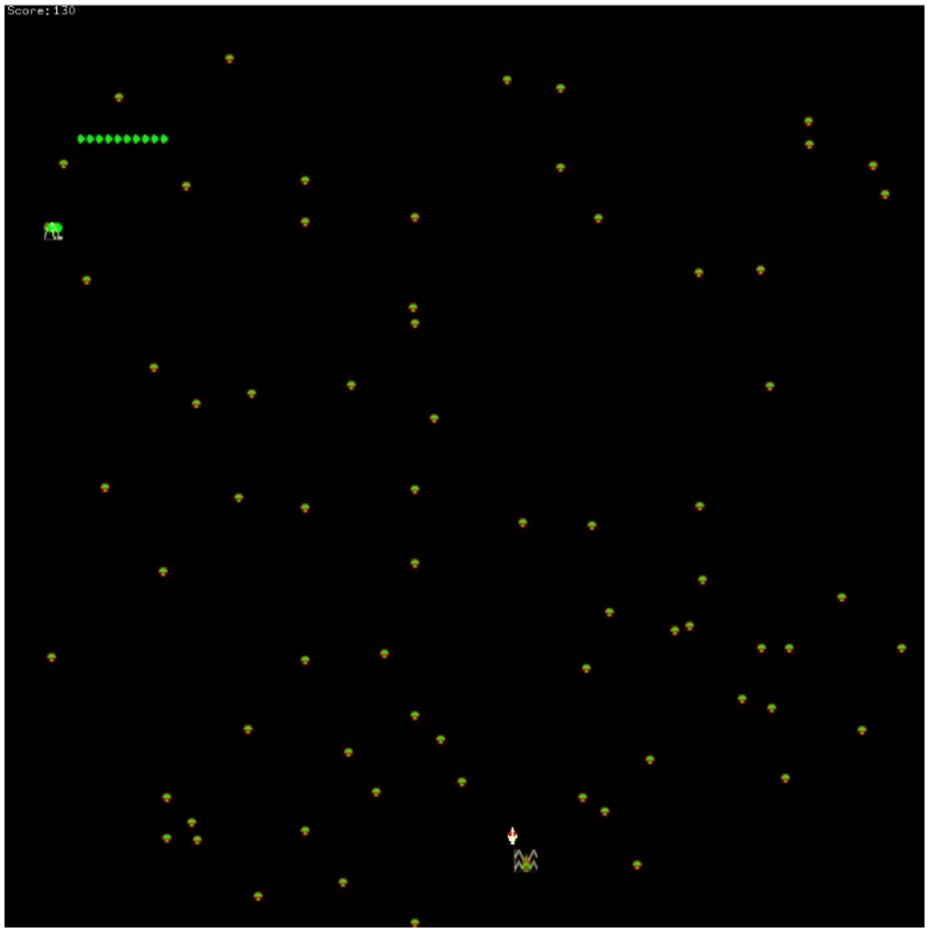

Figure 2: Game Screen

The game over screen is shown every time you die in the gameplay screen. This screen gives you the option to exit the app or play again, it will reset the game and allow you to play from scratch.

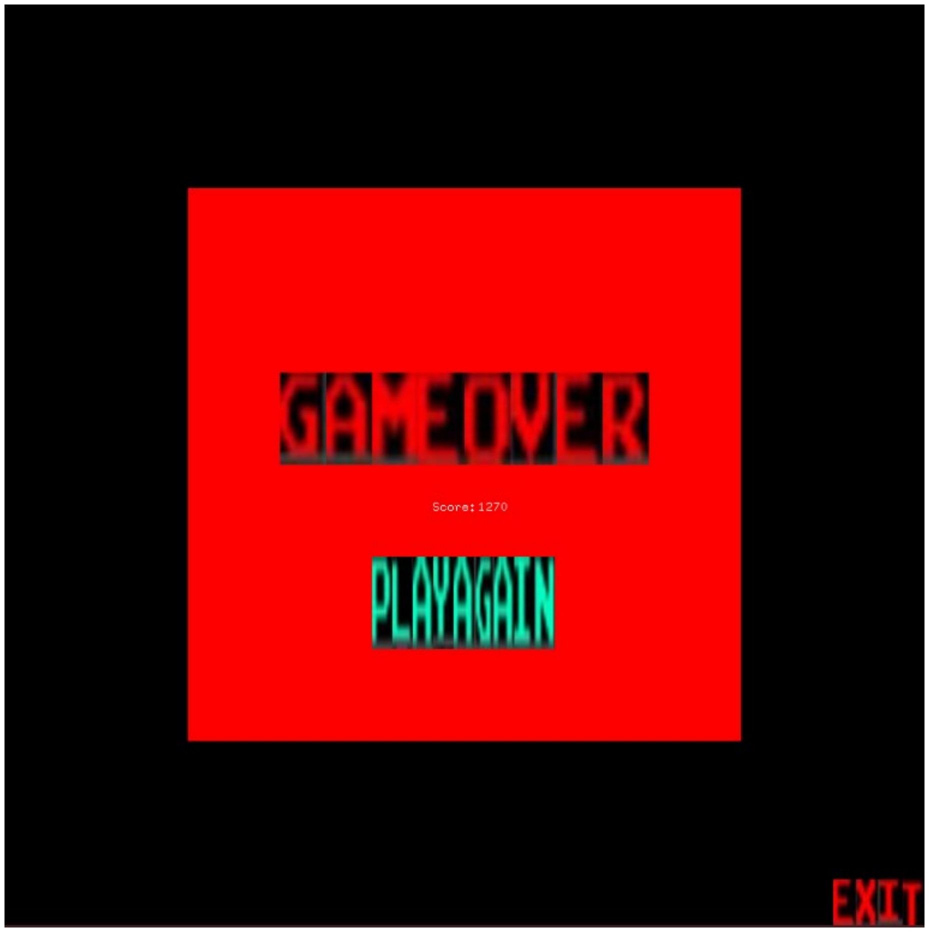

Figure 3: Game Over Screen

4 Code

4.1 main.cpp

```
#include "ofMain.h"
#include "ofApp.h"

//========================================================================
int main( ){

   ofSetupOpenGL(1000,1000, OF_WINDOW);     // <-------- setup the GL context

   // this kicks off the running of my app
   // can be OF_WINDOW or OF_FULLSCREEN
   // pass in width and height too:
   ofRunApp(new ofApp());

}
```

4.2 ofApp.h

```
#pragma once
#include "..\Player.h"
#include "..\Tree.h"
#include "..\Projectile.h"
#include "..\Spider.h"
#include "..\Centi.h"
#include "..\Flea.h"
#include "ofMain.h"
#define N_MAX 10000

class ofApp : public ofBaseApp{
   public:
      int gamestate = 1;
      Player player;
      Tree trees[N_MAX];
      int treecount;
      int projectileCount;
      Projectile projectiles[N_MAX];
      Spider spider;
      float spiderDeathTime;
      float spiderRespawnTime;
      Centi myCentipede[10];
      int centiCount = 10;
```

```
     int score;
     string displayScore;
     Flea flea;
     int flearoll;
     float fleaDeathTime;
     float fleaRespawnTime;
     ofSoundPlayer bgm, shoot, hit, click, die;
     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);
};
```

4.3 ofApp.cpp

```
#include "ofApp.h"
using namespace std;

//--------------------------------------------------------------
void ofApp::setup(){
   ofBackground(0, 0, 0);
   score = 0;
   die.load("../../sounds/die.mp3");
   bgm.load("../../sounds/bgm.mp3");
   shoot.load("../../sounds/shoot.mp3");
   hit.load("../../sounds/hit.mp3");
   click.load("../../sounds/click.mp3");
   die.setVolume(1);
   bgm.setVolume(0.25);
   shoot.setVolume(1);
   shoot.setSpeed(2);
   hit.setVolume(0.5);
   click.setVolume(0.5);
   bgm.play();
   bgm.setLoop(true);
   player.setup();
```

```
  treecount = 60;
   spiderRespawnTime = 15000;
  for (int i = 0; i < treecount; i++)
    {

      trees[i].setup();
      trees[i].x = 25+rand() % (975-25 + 1);
      trees[i].y = 25+rand() % (975-25 + 1);
    }
  projectileCount = 0;
  for (int i = 0; i < N_MAX; i++)
  {
    projectiles[i].setup();
  }
  spider.setup();

  for (int i = 0; i < centiCount; i++)
  {
    myCentipede[i].setup();
    myCentipede[i].x = (10 * i) + 10;
    myCentipede[i].y = 0;
  }
  flea.setup();
  fleaRespawnTime = 10000;
}

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

  if (gamestate == 2) {
    ofHideCursor();
    displayScore = ofToString(score);
    player.update();
    for (int i = 0; i < treecount; i++)
    {
      trees[i].update();
      if (player.intersects(trees[i])) {
        die.play();
        gamestate = 3;
      }
    }
    for (int i = 0; i < N_MAX; i++)
    {
      projectiles[i].update();
    }
    for (int i = 0; i < projectileCount; i++)
    {
```

```
   for (int j = 0; j < treecount; j++)
   {
      if (projectiles[i].intersects(trees[j])) {
         hit.play();
         score = score + 10;
         trees[j].hitpoints = trees[j].hitpoints - 1;
         projectiles[i].x = 10000000000000000000;
      }

   }
}



spider.update();
if (spider.intersects(player))
{
   die.play();
   gamestate = 3;
}
for (int i = 0; i < projectileCount; i++)
{
   if (projectiles[i].intersects(spider)) {
      hit.play();
      score = score + 100;
      spider.x = -1000000000;
      spider.width = 0;
      projectiles[i].x = -1000000000000000;
      spiderDeathTime = ofGetElapsedTimeMillis();
      if (spiderRespawnTime > 3000) {
         spiderRespawnTime = spiderRespawnTime - 1000;
      }

   }
}
if (spider.width==0) {
   if (ofGetElapsedTimeMillis()-spiderDeathTime>spiderRespawnTime) {
      spider.setup();
   }
}


for (int i = 0; i < centiCount; i++)
{
   myCentipede[i].update();
}
for (int i = 0; i < centiCount; i++)
```

```
{
   for (int j = 0; j < treecount; j++)
   {
      if (myCentipede[i].intersects(trees[j])) {
         myCentipede[i].vx = myCentipede[i].vx * -1;
         myCentipede[i].y = myCentipede[i].y + 10;
      }
   }
}
for (int i = 0; i < centiCount; i++)
{
   if(myCentipede[i].intersects(player)) {
      die.play();
      gamestate = 3;
   }
}
for (int i = 0; i < centiCount; i++)
{
   for (int j = 0; j < projectileCount; j++)
   {
      if (myCentipede[i].intersects(projectiles[j]))
      {
         hit.play();
         score = score + 100;
         trees[treecount].setup();
         trees[treecount].x = myCentipede[i].x;
         trees[treecount].y = myCentipede[i].y;
         myCentipede[i].x = -10000000;
         myCentipede[i].width = 0;
         projectiles[j].x = 10000000;
         treecount = treecount + 1;
      }
   }
}
int deadCentipedes=0;
for (int i = 0; i < centiCount; i++)
{
   if (myCentipede[i].width==0)
   {
      deadCentipedes = deadCentipedes + 1;
   }
}
if (deadCentipedes == 10)
{
   for (int i = 0; i < centiCount; i++)
   {
      myCentipede[i].setup();
```

```
            myCentipede[i].x = (10 * i) + 10;
            myCentipede[i].y = 0;
        }
    }



    flea.update();
    if (flea.intersects(player))
    {
        gamestate = 3;
        die.play();
    }
    for (int i = 0; i < projectileCount; i++)
    {
        if (flea.intersects(projectiles[i]))
        {
            projectiles[i].x=-100000000000;
            flea.x=-100000000;
            hit.play();
            score = score + 50;
            fleaDeathTime = ofGetElapsedTimeMillis();
            fleaRespawnTime = fleaRespawnTime - 500;
        }
    }
    if (flea.y > 1000&&flea.x>0)
    {
        flea.setup();
    }
    if (flea.x < -1000)
    {
        if (ofGetElapsedTimeMillis() - fleaDeathTime >= fleaRespawnTime) {
            flea.setup();
        }
    }

    flearoll = 1 + rand() % 40;
    if (flearoll == 3) {
        treecount = treecount + 1;
        trees[treecount].setup();
        trees[treecount].x = flea.x;
        trees[treecount].y = flea.y;

    }
}
if (gamestate == 3) {
    ofShowCursor();
```

```
   }
}

//--------------------------------------------------------------
void ofApp::draw(){
   if (gamestate == 1) {
      ofSetColor(255, 255, 0);
      ofDrawRectangle(200, 200, 600, 600);
      ofSetColor(255, 255, 255);
      ofImage rede, redx, redi, redt, redp, redl, reda, redy, title;
      rede.load("../../images/rede.jpg");
      redx.load("../../images/redx.jpg");
      redi.load("../../images/redi.jpg");
      redt.load("../../images/redt.jpg");
      redp.load("../../images/redp.jpg");
      redl.load("../../images/redl.jpg");
      reda.load("../../images/reda.jpg");
      redy.load("../../images/redy.jpg");
      title.load("../../images/title.jpg");
      rede.draw(900, 950, 25, 50);
      redx.draw(925, 950, 25, 50);
      redi.draw(950, 950, 25, 50);
      redt.draw(975, 950, 25, 50);
      redp.draw(400,600,50,50);
      redl.draw(450,600,50,50);
      reda.draw(500, 600, 50, 50);
      redy.draw(550, 600, 50, 50);
      title.draw(300,300,400,100);
      ofDrawBitmapString("Note:Please do not enter fullscreen mode",0,1000);
   }
   else if (gamestate == 2) {
      ofSetColor(255, 255, 255);
      ofDrawBitmapString("Score: ", 5, 10);
      ofDrawBitmapString(displayScore, 55, 10);
      player.draw();
      for (int i = 0; i < treecount; i++)
      {
         trees[i].draw();
      }
      spider.draw();
      for (int i = 0; i < projectileCount; i++)
      {
         projectiles[i].draw();
      }

      for (int i = 0; i < centiCount; i++)
      {
          myCentipede[i].draw();
      }
      flea.draw();

  }
  else if (gamestate == 3) {
      ofSetColor(255, 0, 0);
      ofDrawRectangle(200, 200, 600, 600);
      ofSetColor(255, 255, 255);
      ofImage rede, redx, redi, redt, redg, reda, redm, redo, redv, redr, cyanp,
          cyanl, cyana, cyany, cyang, cyani, cyann;
      rede.load("../../images/rede.jpg");
      redx.load("../../images/redx.jpg");
      redi.load("../../images/redi.jpg");
      redt.load("../../images/redt.jpg");
      redg.load("../../images/redg.jpg");
      reda.load("../../images/reda.jpg");
      redm.load("../../images/redm.jpg");
      redo.load("../../images/redo.jpg");
      redv.load("../../images/redv.jpg");
      redr.load("../../images/redr.jpg");
      cyanp.load("../../images/cyanp.jpg");
      cyanl.load("../../images/cyanl.jpg");
      cyana.load("../../images/cyana.jpg");
      cyany.load("../../images/cyany.jpg");
      cyang.load("../../images/cyang.jpg");
      cyani.load("../../images/cyani.jpg");
      cyann.load("../../images/cyann.jpg");
      rede.draw(900, 950, 25, 50);
      redx.draw(925, 950, 25, 50);
      redi.draw(950, 950, 25, 50);
      redt.draw(975, 950, 25, 50);
      redg.draw(300, 400, 50, 100);
      reda.draw(350, 400, 50, 100);
      redm.draw(400, 400, 50, 100);
      rede.draw(450, 400, 50, 100);
      redo.draw(500, 400, 50, 100);
      redv.draw(550, 400, 50, 100);
      rede.draw(600, 400, 50, 100);
      redr.draw(650, 400, 50, 100);
      cyanp.draw(400, 600, 22, 100);
      cyanl.draw(422, 600, 22, 100);
      cyana.draw(444, 600, 22, 100);
      cyany.draw(466, 600, 22, 100);
      cyana.draw(488, 600, 22, 100);
      cyang.draw(510, 600, 22, 100);
      cyana.draw(532, 600, 22, 100);
```

```
        cyani.draw(554, 600, 22, 100);
        cyann.draw(576, 600, 22, 100);
        ofDrawBitmapString("Score: ", 465, 550);
        ofDrawBitmapString(displayScore, 515, 550);
    }



}

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

}

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

}

//--------------------------------------------------------------
void ofApp::mouseMoved(int x, int y){

    if (y > 700)
    {
        player.y = y;
    }
    else {
        player.y = 700;
    }
    if (x < 1000)
    {
        player.x = x - player.width / 2;
    }
    else
    {
        player.x = 990;
    }


}

//--------------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){

}
```

```
//--------------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
   if (gamestate == 1)
   {
      if (x > 900 && y > 900) {
         click.play();
         OF_EXIT_APP(0);
      }
      if (x > 400 && x < 600 && y>600 && y < 650) {
         click.play();
         gamestate = 2;
      }
   }
   if (gamestate == 2)
   {
      shoot.play();
      if (x < 1000) {
         projectiles[projectileCount].x = x;
      }
      else {
         projectiles[projectileCount].x = 1000;
      }

      if (y > 700) {
         projectiles[projectileCount].y = y;
      }
      else {
         projectiles[projectileCount].y = 700;
      }
      projectileCount = projectileCount + 1;
   }
   if (gamestate == 3)
   {
      if (x > 900 && y > 900) {
         click.play();
         OF_EXIT_APP(0);
      }
      if (x>=400&&x<=600&&y>=600&&y<=700)
      {
         click.play();
         gamestate = 2;
         setup();
      }
   }
}

//--------------------------------------------------------------
```

```
void ofApp::mouseReleased(int x, int y, int button){

}

//--------------------------------------------------------------
void ofApp::mouseEntered(int x, int y){

}

//--------------------------------------------------------------
void ofApp::mouseExited(int x, int y){

}

//--------------------------------------------------------------
void ofApp::windowResized(int w, int h){

}

//--------------------------------------------------------------
void ofApp::gotMessage(ofMessage msg){

}

//--------------------------------------------------------------
void ofApp::dragEvent(ofDragInfo dragInfo){

}
```

4.4 Player.h

```
#pragma once
#include "ofMain.h"
class Player :public ofRectangle
{
public:
   ofImage playerImage;

   Player();
   void setup();
   void update();
   void draw();
};
```

4.5 Player.cpp

```
#include "Player.h"
Player::Player()
{

}
void Player::setup()
{
   playerImage.load("../../images/player.jpg");
   x=ofGetWidth()/2;
   y=ofGetHeight() / 1.125;
   width=10;
   height=20;
}
void Player::update()
{

}
void Player::draw()
{
   ofSetColor(255, 255, 255);
   playerImage.draw(x,y,width,height);
}
```

4.6 Tree.h

```
#pragma once
#include "ofMain.h"
class Tree :public ofRectangle
{
public:
   ofImage hp4,hp3,hp2, hp1;
   int hitpoints;

   Tree();
   void setup();
   void update();
   void draw();
};
```

4.7 Tree.cpp

```
#include "Tree.h"
```

```
Tree::Tree()
{

}
void Tree::setup()
{
   hp4.load("../../images/hp4.jpg");
   hp3.load("../../images/hp3.jpg");
   hp2.load("../../images/hp2.jpg");
   hp1.load("../../images/hp1.jpg");
   hitpoints = 4;
   width = 10;
   height = 10;
}
void Tree::update()
{

}
void Tree::draw()
{
    if (hitpoints == 4)
   {
      hp4.draw(x, y,width,height);
   }
   else if (hitpoints == 3)
   {
      hp3.draw(x, y,width,height);
   }
   else if (hitpoints == 2)
   {
      hp2.draw(x, y,width,height);
   }
   else if (hitpoints == 1)
   {
      hp1.draw(x, y,width,height);
   }
   else if (hitpoints == 0)
   {
      x = 10000;
   }

}
```

4.8 Projectile.h

```
#pragma once
```

```cpp
#include "ofMain.h"
class Projectile :public ofRectangle
{
public:
   int vy;

   Projectile();
   void setup();
   void update();
   void draw();
};
```

4.9 Projectile.cpp

```cpp
#include "Projectile.h"
Projectile::Projectile()
{

}
void Projectile::setup()
{
   vy = -10;
   width = 2;
   height = 10;
}
void Projectile::update()
{
   y = y + vy;
}
void Projectile::draw()
{
   ofSetColor(255, 0, 0);
   ofDrawRectangle(x,y,width,height);
}
```

4.10 Spider.h

```cpp
#pragma once
#include "ofMain.h"
class Spider :public ofRectangle
{
public:
   int vy;
   int vx;
   ofImage spiderImage;

   Spider();
   void setup();
   void update();
   void draw();
};
```

4.11 Spider.cpp

```
#include "Spider.h"
Spider::Spider()
{

}
void Spider::setup()
{
   vy = -5;
   vx = 5;
   x = 900;
   y = 900;
   width = 25;
   height = 25;
   spiderImage.load("../../images/spider.jpg");


}
void Spider::update()
{
   x = x + vx;
   y = y + vy;
   if (x>975||x<0) {
      vx = vx * -1;
   }
   if (y>975||y<675) {
      vy = vy * -1;
   }
}
void Spider::draw()
{
   spiderImage.draw(x, y, width, height);
}
```

4.12 Centi.h

```
#pragma once
#include "ofMain.h"
class Centi :public ofRectangle
{
public:
   int vx;
   ofImage centiImage;

   Centi();
   void setup();
   void update();
   void draw();
};
```

4.13 Centi.cpp

```
#include "Centi.h"
Centi::Centi()
{

}
void Centi::setup()
{
   vx = 10;
   width = 10;
   height = 10;
   centiImage.load("../../images/centi.jpg");
}
void Centi::update()
{
   x = vx + x;
   if (x>=1000||x<=0) {
      vx = vx * -1;
      y = y + 10;
   }
   if (y >= 1000)
   {
      x = -10000000;
      width = 0;
   }
}
void Centi::draw()
{
   ofSetColor(255, 255, 255);
   centiImage.draw(x, y, width, height);
}
```

4.14 Flea.h

```
#pragma once
#include "ofMain.h"
class Flea :public ofRectangle
{
public:
   int vy;
   ofImage fleaImage;

   Flea();
   void setup();
   void update();
   void draw();
};
```

4.15 Flea.cpp

```
#include "Flea.h"
Flea::Flea()
{

}
void Flea::setup()
{
   vy = 5;
   width = 20;
   height = 20;
   x = 10 + rand() % (1000 - 10 + 1);
   y = 0;
   fleaImage.load("../../images/flea.png");
}
void Flea::update()
{
   y = y + vy;
}
void Flea::draw()
{
   ofSetColor(255, 255, 255);
   fleaImage.draw(x, y, width, height);
}
```