#include <SFML/Graphics.hpp>

#include <time.h>

#include <list>

#include <cmath>

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

using namespace sf;

const int W = 1920;

const int H = 1080;

const int AsteroidSpawnFreq = 150; //less -> frequently

const bool IsVertSynch = true;

const int MaxFrames = 60;

const int RotateSpeed = 4; //deg/sec

const float DEGTORAD = 0.0174533; //rad in deg

const int CSpaceCD = 12;

const int MaxPlayerSpeed = 15;

string NumberToString ( int Number )

{

ostringstream ss;

ss << Number;

return ss.str();

}

class Animation

{

public:

float Frame, speed;

Sprite sprite;

std::vector<IntRect> frames;

Animation(){}

Animation (Texture &t, int x, int y, int w, int h, int count, float Speed)

{

Frame = 0;

speed = Speed;

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

frames.push\_back( IntRect(x+i\*w, y, w, h) );

sprite.setTexture(t);

sprite.setOrigin(w/2,h/2);

sprite.setTextureRect(frames[0]);

}

void update()

{

Frame += speed;

int n = frames.size();

if (Frame >= n) Frame -= n;

if (n>0) sprite.setTextureRect( frames[int(Frame)] );

}

bool isEnd()

{

return Frame+speed>=frames.size();

}

};

class sprite

{

public:

float x,y,Vx,Vy,R,angle;

bool life;

string name;

Animation anim;

sprite()

{

life=1;

}

void settings(Animation &a,int X,int Y,float Angle=0,int radius=1)

{

anim = a;

x=X; y=Y;

angle = Angle;

R = radius;

}

virtual void update(){};

void draw(RenderWindow &app)

{

anim.sprite.setPosition(x,y);

anim.sprite.setRotation(angle+90);

app.draw(anim.sprite);

}

virtual ~sprite(){};

};

class asteroid: public sprite

{

public:

asteroid()

{

Vx=rand()%8-4;

Vy=rand()%8-4;

name="asteroid";

}

void update()

{

x+=Vx;

y+=Vy;

if (x>W) x=0; if (x<0) x=W;

if (y>H) y=0; if (y<0) y=H;

}

};

class bullet: public sprite

{

public:

bullet()

{

name="bullet";

}

void update()

{

Vx=cos(angle\*DEGTORAD)\*17;

Vy=sin(angle\*DEGTORAD)\*17;

angle+=rand()%3-1;

x+=Vx;

y+=Vy;

if (x>W || x<0 || y>H || y<0) life=0;

}

};

class player: public sprite

{

public:

bool thrust;

bool spawning;

int score;

int lifes;

int SpaceCD;

sf::Clock spawn\_time;

player()

{

name="player";

spawning=true;

lifes=3;

spawn\_time.restart();

score=0;

}

void update()

{

if (thrust)

{ Vx+=cos(angle\*DEGTORAD)\*0.2;

Vy+=sin(angle\*DEGTORAD)\*0.2; }

else

{ Vx\*=0.98;

Vy\*=0.98; }

float speed = sqrt(Vx\*Vx+Vy\*Vy);

if (speed>MaxPlayerSpeed)

{ Vx \*= MaxPlayerSpeed/speed;

Vy \*= MaxPlayerSpeed/speed; }

x+=Vx;

y+=Vy;

if (spawn\_time.getElapsedTime().asSeconds()>3)

spawning=false;

if (x>W) x=0; if (x<0) x=W;

if (y>H) y=0; if (y<0) y=H;

}

};

class enemy: public sprite

{

};

bool isCollide(sprite \*a,sprite \*b){

return (b->x - a->x)\*(b->x - a->x)+(b->y - a->y)\*(b->y - a->y)<(a->R + b->R)\*(a->R + b->R);

}

std::list<sprite\*> sprites;

int main() {

srand(time(0));

sf::Font font;

if (!font.loadFromFile("ARLRDBD.TTF"))

cout<<"Error loading font\n";

sf::Text LifesText, ScoreText;

LifesText=sf::Text("Lifes: ", font, 30);

LifesText.setColor(sf::Color(128, 191, 255));

LifesText.setPosition(0,30);

LifesText.setStyle(sf::Text::Bold);

ScoreText=sf::Text("Score: ", font, 30);

ScoreText.setColor(sf::Color(128, 191, 255));

ScoreText.setPosition(0,0);

ScoreText.setStyle(sf::Text::Bold);

RenderWindow app(VideoMode(W, H), "Astroids", sf::Style::Fullscreen);

app.setFramerateLimit(MaxFrames);

app.setVerticalSyncEnabled(IsVertSynch);

Texture t1,t2,t3,t4,t5,t6,t7,t8,t9;

t1.loadFromFile("images/spritesheetSpaceship.png");

t2.loadFromFile("images/background.jpg");

t3.loadFromFile("images/explosions/type\_C.png");

t4.loadFromFile("images/rock.png");

t5.loadFromFile("images/fire\_blue.png");

t6.loadFromFile("images/rock\_small.png");

t7.loadFromFile("images/explosions/type\_B.png");

t8.loadFromFile("images/explosions/Shield.png");

t9.loadFromFile("images/explosions/type\_A.png");

//t1.setSmooth(true);

//t2.setSmooth(true);

Sprite background(t2);

Animation sExplosion\_ship(t7, 0,0,192,192, 64, 0.5);

Animation sRock(t4, 0,0,64,64, 16, 0.2);

Animation sRock\_small(t6, 0,0,64,64, 16, 0.2);

Animation sBullet(t5, 0,0,32,64, 16, 0.8);

Animation sPlayer(t1, 192,0,96,96, 1, 0);

Animation sPlayer\_L(t1, 288,0,96,96, 1, 0);

Animation sPlayer\_R(t1, 96,0,96,96, 1, 0);

Animation sPlayer\_go(t1, 0,0,96,96, 1, 0);

Animation sExplosion(t3, 0,0,256,256, 48, 0.5);

Animation sExplosion\_ship1(t1, 0,96,96,96, 7, 0.2);

Animation Shield(t8, 0,0,96,96, 1, 0);

for(int i=0;i<10;i++)

{

asteroid \*a = new asteroid();

a->settings(sRock, rand()%W, rand()%H, rand()%360, 25);

sprites.push\_back(a);

}

player \*p = new player();

p->settings(sPlayer,W/2,H/2,0,40);

p->SpaceCD=0;

sprites.push\_back(p);

/////main loop/////

while (app.isOpen())

{

Event event;

while (app.pollEvent(event))

{

if (event.type == Event::Closed)

app.close();

if (event.type == Event::KeyPressed){

if (event.key.code == Keyboard::Escape)

{

app.close();

}

/\*if (event.key.code == Keyboard::P) //pause

{

app.waitEvent(event);

if (event.key.code==Keyboard::P)

{

}

}\*/

if (event.key.code == Keyboard::Escape)

{

app.close();

}

}

}

if (!Keyboard::isKeyPressed(Keyboard::Up)){

p->thrust=false;

p->anim = sPlayer;

}

if (Keyboard::isKeyPressed(Keyboard::Right)) {

p->angle+=RotateSpeed;

p->anim=sPlayer\_R;

}

if (Keyboard::isKeyPressed(Keyboard::Left)) {

p->angle-=RotateSpeed;

p->anim=sPlayer\_L;

}

if (Keyboard::isKeyPressed(Keyboard::Up)) {

p->thrust=true;

p->anim = sPlayer\_go;

}

if (Keyboard::isKeyPressed(Keyboard::Space))

if (p->SpaceCD>CSpaceCD)

{

bullet \*b = new bullet();

b->settings(sBullet,p->x,p->y,p->angle,15);

sprites.push\_back(b);

p->SpaceCD=0;

}

p->SpaceCD++;

for(auto a:sprites)

for(auto b:sprites)

{

if (a->name=="asteroid" && b->name=="bullet")

if ( isCollide(a,b) )

{

a->life=false;

b->life=false;

if (a->R==25) p->score+=10;

if (a->R==15) p->score+=5;

sprite \*e = new sprite();

e->settings(sExplosion,a->x,a->y);

e->name="explosion";

sprites.push\_back(e);

for(int i=0;i<2;i++)

{

if (a->R==15) continue;

sprite \*e = new asteroid();

e->settings(sRock\_small,a->x,a->y,rand()%360,15);

sprites.push\_back(e);

}

}

if (a->name=="player" && b->name=="asteroid")

if ( isCollide(a,b) && !p->spawning)

{

b->life=false;

sprite \*e = new sprite();

e->settings(sExplosion\_ship1,a->x,a->y,p->angle);

e->name="explosion";

sprites.push\_back(e);

p->settings(sPlayer,W/2,H/2);

p->lifes--;

p->spawning=true;

p->spawn\_time.restart();

p->Vx=0; p->Vy=0;

}

/\*if (a->name=="asteroid" && b->name=="asteroid" && a!=b)

if ( isCollide(a,b))

{

cout<<"1";

int Vx0=(a->Vx+b->Vx)/2;

int Vy0=(a->Vy+b->Vy)/2;

a->Vx=a->Vx\*-1;

a->Vy=a->Vy\*-1;

//b->Vx=b->Vx\*-1;

//b->Vy=b->Vy\*-1;

}\*/

}

for(auto e:sprites)

if (e->name=="explosion")

if (e->anim.isEnd()) e->life=0;

if (rand()%AsteroidSpawnFreq==0)

{

asteroid \*a = new asteroid();

a->settings(sRock, 0,rand()%H, rand()%360, 25);

sprites.push\_back(a);

}

for(auto i=sprites.begin();i!=sprites.end();)

{

sprite \*e = \*i;

e->update();

e->anim.update();

if (e->life==false) {i=sprites.erase(i); delete e;}

else i++;

}

//////draw//////

app.draw(background);

for(auto i:sprites)

i->draw(app);

if(IntRect(0,30,LifesText.getLocalBounds().width,30).contains(Mouse::getPosition(app)))

LifesText.setColor(sf::Color::Red);

else LifesText.setColor(sf::Color(128, 191, 255));

if (ScoreText.getLocalBounds().contains(Mouse::getPosition(app).x,Mouse::getPosition(app).y))

ScoreText.setColor(sf::Color::Red);

else ScoreText.setColor(sf::Color(128, 191, 255));

LifesText.setString("Lifes: "+NumberToString(p->lifes));

ScoreText.setString("Score: "+NumberToString(p->score));

app.draw(LifesText);

app.draw(ScoreText);

app.display();

}

return 0;

}