#include<iostream>

#include<fstream>

#include<time.h>

#include<string>

#include<sstream>

#include<iomanip>

#include<SFML/Graphics.hpp>

#include<SFML/Audio.hpp>

#define max 20

using namespace std;

using namespace sf;

sf::RenderWindow window(sf::VideoMode(1300, 600), "Journey To Mars", sf::Style::Close | sf::Style::Titlebar | sf::Style::Resize);

class queue {

struct node {

int data;

node\* next;

};

node\* front;

public:

queue()

{

front = NULL;

}

void enqueue(int num)

{

node\* n = new node;

n->next = NULL;

n->data = num;

if (front != NULL)

{

node\* current = front;

while (current->next != NULL)

{

current = current->next;

}

current->next = n;

}

else front = n;

}

int dequeue()

{

if (isEmpty())

{

node\* current = front;

return current->data;

}

node\* temp = front;

int r = temp->data;

front = front->next;

delete temp;

return r;

}

bool isEmpty()

{

if (front == NULL)

return true;

else

return false;

}

void disp()

{

if (isEmpty())

{

return;

}

node\* current = front;

while (current != NULL)

{

std::cout << current->data << " ";

current = current->next;

}

std::cout << std::endl;

}

node\* getfront()

{

return front;

}

int getend()

{

node\* current = front;

while (current != NULL)

{

current = current->next;

}

return current->data;

}

};

class adjacencyMatrix

{

int n;

int\*\* arr;

bool\* visited;

public:

adjacencyMatrix(int n)

{

this->n = n;

visited = new bool[n];

arr = new int\* [n];

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

{

arr[i] = new int[n];

for (int j = 0; j < n; j++)

{

arr[i][j] = 0;

}

}

}

void addedge(int o, int t, int w)

{

if (o > n || t > n || o < 0 || t < 0)

{

std::cout << "Invalid Edge\n";

}

else

{

arr[o][t] = w;

}

}

bool DFSInitial(int s, int d)

{

visited = new bool[n];

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

{

visited[i] = false;

}

queue q;

visited[s] = true;

q.enqueue(s);

queue t;

while (!q.isEmpty())

{

int vis = q.dequeue();

t.enqueue(vis);

if (vis == d)

{

t.disp();

return true;

}

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

{

if (arr[vis][i] != 0 && !visited[i])

{

visited[i] = true;

q.enqueue(i);

}

}

}

return false;

}

};

template<typename t>

class list

{

unsigned size;

unsigned noofelements;

t\*\* arr;

void initialize(unsigned from)

{

for (int i = from; i < size; i++)

{

arr[i] = NULL;

}

}

void expand()

{

size \*= 2;

t\*\* temp = new t \* [size];

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

{

temp[i]=arr[i];

}

delete[]arr;

arr = temp;

initialize(noofelements);

}

public:

list(int size = 5)

{

this->size = size;

noofelements = 0;

arr = new t \* [size];

initialize(0);

}

list(const list& obj)

{

this->size = obj.size;

noofelements = obj.noofelements;

arr = new t \* [size];

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

{

arr[i] = new t(\*obj.arr[i]);

}

initialize(noofelements);

}

t& operator[](int index)

{

if (index < 0||index>=noofelements)

{

throw("Out of bounds");

}

return \*arr[index];

}

void operator=(const t& obj)

{

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

{

delete arr[i];

}

delete[]arr;

this->size = obj.size;

noofelements = obj.noofelements;

arr = new t \* [size];

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

{

arr[i] = new t(\*obj.arr[i]);

}

initialize(noofelements);

}

void add(const t element)

{

if (noofelements >= size)

{

expand();

}

arr[noofelements++] = new t(element);

}

void remove(int index)

{

if (index < 0 ||index >= noofelements)

{

throw("Out of bounds");

}

delete arr[index];

arr[index] = arr[noofelements - 1];

arr[--noofelements] = NULL;

}

int getsize() const

{

return noofelements;

}

void clear()

{

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

{

delete arr[i];

}

initialize(0);

noofelements = 0;

}

~list()

{

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

{

delete arr[i];

}

delete[]arr;

}

};

template<class T>

class LinkedList

{

template<class T>

struct node

{

T data;

node\* next;

};

node<T>\* first;

node<T>\* temp;

node<T>\* current;

public:

LinkedList<T>()

{

first = NULL;

temp = NULL;

current = NULL;

}

void add(T d)

{

node<T>\* n = new node<T>;

n->next = NULL;

n->data = d;

if (first != NULL)

{

current = first;

while (current->next != NULL)

{

current = current->next;

}

current->next = n;

}

else first = n;

}

T get(int index)

{

if (index == 0)

{

return this->first->data;

}

else

{

node<T>\* curr = this->first;

for (int i = 0; i < index; ++i) {

curr = curr->next;

}

return curr->data;

}

}

T& operator[](int index)

{

int i = 0;

current = first;

while (i < index)

{

i++;

current = current->next;

}

return current->data;

}

void erase(int position)

{

if (first == NULL)

return;

node<T>\* temp = first;

if (position == 0)

{

first = temp->next;

free(temp);

return;

}

for (int i = 0; temp != NULL && i < position - 1; i++)

temp = temp->next;

if (temp == NULL || temp->next == NULL)

return;

node<T>\* next = temp->next->next;

free(temp->next);

temp->next = next;

}

int size()

{

int i = 0;

node<T>\* current = first;

while (current!= NULL)

{

current = current->next;

i++;

}

return i;

}

};

class menu

{

Font font;

Text newgame;

Text loadgame;

Text exit;

Texture t;

Sprite selector;

ifstream save;

int selectime;

int selectorint;

int maxselecttime;

public:

menu()

{

selectorint = 0;

maxselecttime = 25;

selectime = maxselecttime;

t.loadFromFile("Textures/Selector.png");

font.loadFromFile("Fonts/arial.ttf");

selector.setTexture(t);

selector.setScale(0.35, 0.25);

selector.setPosition(550, 150);

selector.setOrigin(100, 100);

newgame.setFont(font);

loadgame.setFont(font);

exit.setFont(font);

newgame.setFillColor(Color::Red);

loadgame.setFillColor(Color::Red);

exit.setFillColor(Color::Red);

newgame.setCharacterSize(25);

loadgame.setCharacterSize(25);

exit.setCharacterSize(25);

newgame.setString("New Game");

loadgame.setString("Load Game");

exit.setString("exit");

newgame.setPosition(550, 150);

loadgame.setPosition(550, 250);

exit.setPosition(550, 350);

}

int getselector()

{

return selectorint;

}

void motion(int &hp,int &score,int &time)

{

if (selectime >= maxselecttime)

{

int x=0, y=0;

if (Keyboard::isKeyPressed(Keyboard::W)|| Keyboard::isKeyPressed(Keyboard::Up))

{

if (selector.getPosition().y >= 250)

selector.setPosition(550, selector.getPosition().y - 100);

}

if (Keyboard::isKeyPressed(Keyboard::S) || Keyboard::isKeyPressed(Keyboard::Down))

{

if(selector.getPosition().y <= 250)

selector.setPosition(550, selector.getPosition().y + 100);

}

if (Keyboard::isKeyPressed(Keyboard::Enter))

{

if (selector.getPosition().y == 150)

selectorint = 1;

if (selector.getPosition().y == 250)

{

save.open("Save file/save.txt",ios::out);

if (!save.is\_open())

{

cout << "File not found: ";

window.close();

}

save >> selectorint>> hp >> score >> time;

save.close();

}

if (selector.getPosition().y == 350)

window.close();

}

selectime -= 5;

}

}

void draw()

{

window.draw(newgame);

window.draw(loadgame);

window.draw(exit);

window.draw(selector);

}

void update(int &hp,int &score,int &time)

{

motion(hp,score,time);

if (selectime < maxselecttime)

{

selectime++;

}

}

};

class levelchanger

{

Font font;

Text port;

Text starboard;

Texture t;

Sprite selector;

int selectime;

int select1;

bool randcheck;

int select2;

int maxselecttime;

public:

levelchanger()

{

randcheck = false;

srand(time(NULL));

select1 = rand() % 6 + 7;

select2=4;

maxselecttime = 25;

selectime = 10;

t.loadFromFile("Textures/Selector.png");

font.loadFromFile("Fonts/arial.ttf");

selector.setTexture(t);

selector.setScale(0.35, 0.25);

selector.setPosition(350, 200);

selector.setOrigin(100, 100);

port.setFont(font);

starboard.setFont(font);

port.setFillColor(Color::Red);

starboard.setFillColor(Color::Red);

port.setCharacterSize(25);

starboard.setCharacterSize(25);

port.setString("Port");

starboard.setString("Starboard");

port.setPosition(350, 200);

starboard.setPosition(850, 200);

}

void motion(adjacencyMatrix m,int level,bool &test,bool& penalty)

{

if(randcheck==false)

select2 = rand() % 2;

cout << select2 << endl;

randcheck = true;

if (selectime >= maxselecttime)

{

if (Keyboard::isKeyPressed(Keyboard::A) || Keyboard::isKeyPressed(Keyboard::Left))

{

selector.setPosition(350,200);

}

if (Keyboard::isKeyPressed(Keyboard::D) || Keyboard::isKeyPressed(Keyboard::Right))

{

selector.setPosition(850,200);

}

if (Keyboard::isKeyPressed(Keyboard::Enter))

{

if (select2==1)

{

if (selector.getPosition() == Vector2f(350, 200))

{

test=m.DFSInitial(level, level + 1);

}

else if (selector.getPosition() == Vector2f(850, 200))

{

penalty = true;

}

}

else

{

if (selector.getPosition() == Vector2f(350, 200))

{

penalty = true;

}

else if (selector.getPosition() == Vector2f(850, 200))

{

test=m.DFSInitial(level, level + 1);

}

}

randcheck = false;

}

selectime -= 5;

}

}

void draw()

{

window.draw(port);

window.draw(starboard);

window.draw(selector);

}

void update(adjacencyMatrix m,int x,bool &t,bool &penalty)

{

motion(m,x,t,penalty);

if (selectime < maxselecttime)

{

selectime++;

}

}

};

class weapon

{

Texture\* texture;

Sprite wsprite;

Vector2f maxvelocity;

sf::SoundBuffer buffer;

sf::Sound wsound;

public:

weapon(Texture \*t,Vector2f position,Vector2f v=Vector2f(15,0))

{

buffer.loadFromFile("Audio/missile.wav");

wsound.setBuffer(buffer);

texture = t;

wsprite.setTexture(\*texture);

maxvelocity = v;

wsprite.setScale(0.04f, 0.04f);

wsprite.setPosition(position);

}

void motion()

{

wsprite.move(maxvelocity.x, maxvelocity.y);

}

void update()

{

motion();

}

void draw()

{

window.draw(wsprite);

}

FloatRect getweaponbounds()

{

return wsprite.getGlobalBounds();

}

Vector2f getpos()

{

return wsprite.getPosition();

}

~weapon()

{

}

};

class player

{

sf::Sprite PlayerDesign;

Texture\* texture;

Texture\* wtexture;

RectangleShape PlayerRegion;

list<weapon> playerweopon;

int level;

int exp;

int expnext;

int hp;

int totalhp;

int score;

int damage;

int damagemax;

int shoottimer;

int maxshoottime;

int damagetimer;

int maxdamagetime;

public:

player(Texture\* t,Texture\*wt)

{

texture = t;

wtexture = wt;

PlayerDesign.setTexture(\*texture);

PlayerDesign.setScale(0.2f, 0.2f);

PlayerDesign.setOrigin(80, 150);

PlayerDesign.setPosition(130, 250);

level = 1;

exp = 0;

expnext = 100;

hp = 10;

totalhp = 10;

damage = 1;

damagemax = 2;

score = 0;

maxshoottime=25;

shoottimer = maxshoottime;

maxdamagetime =25;

damagetimer=maxdamagetime;

}

list<weapon>& getweapon()

{

return playerweopon;

}

Vector2f getpos()

{

return PlayerDesign.getPosition();

}

FloatRect getplayerbounds()

{

return PlayerDesign.getGlobalBounds();

}

int &getscore()

{

return score;

}

int &gethp()

{

return hp;

}

int &getmaxhp()

{

return totalhp;

}

int& getdamagetime()

{

return damagetimer;

}

int& getmaxdamagetime()

{

return maxdamagetime;

}

int getlevel()

{

return level;

}

void motion()

{

if (sf::Keyboard::isKeyPressed(sf::Keyboard::D) || sf::Keyboard::isKeyPressed(sf::Keyboard::Right))

{

PlayerDesign.move(15, 0);

PlayerDesign.setRotation(0);

if (PlayerDesign.getPosition().x >= 1258)

{

PlayerDesign.move(-15, 0);

}

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::S) || sf::Keyboard::isKeyPressed(sf::Keyboard::Down))

{

PlayerDesign.move(0, 5);

if (PlayerDesign.getPosition().y >= 588)

{

PlayerDesign.move(0, -5);

}

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::A) || sf::Keyboard::isKeyPressed(sf::Keyboard::Left))

{

PlayerDesign.move(-10, 0);

if (PlayerDesign.getPosition().x <= -8)

{

PlayerDesign.move(10, 0);

}

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::W) || sf::Keyboard::isKeyPressed(sf::Keyboard::Up))

{

PlayerDesign.move(0, -5);

if (PlayerDesign.getPosition().y <= 8)

{

PlayerDesign.move(0, 5);

}

}

}

void combat()

{

if (Keyboard::isKeyPressed(Keyboard::Space) && shoottimer >= maxshoottime)

{

playerweopon.add(weapon(wtexture, PlayerDesign.getPosition()));

shoottimer = 0;

}

}

void draw()

{

window.draw(PlayerDesign);

for (int i = 0; i < playerweopon.getsize(); i++)

{

playerweopon[i].draw();

}

}

void update()

{

if (shoottimer < maxshoottime)

{

shoottimer++;

}

if (damagetimer <maxdamagetime)

{

damagetimer++;

}

combat();

motion();

}

};

class boss

{

sf::Sprite bossp;

Texture\* texture;

Texture\* wtexture;

list<weapon> weapon1;

int level;

int exp;

int maxmove;

int hp;

int totalhp;

int movetimer;

int damage;

int damagemax;

int shoottimer;

int maxshoottime;

int damagetimer;

int maxdamagetime;

bool x;

bool y;

public:

boss(Texture\* t, Texture\* wt)

{

texture = t;

wtexture = wt;

bossp.setTexture(\*texture);

bossp.setScale(1.f, 1.f);

bossp.setOrigin(80, 150);

bossp.setPosition(830, 250);

level = 1;

exp = 0;

maxmove = 100;

hp = 20;

totalhp = 10;

damage = 1;

damagemax = 2;

movetimer = maxmove;

maxshoottime = 25;

shoottimer = maxshoottime;

maxdamagetime = 25;

damagetimer = maxdamagetime;

x = false, y = false;

}

list<weapon>& getweapon1()

{

return weapon1;

}

Vector2f getpos()

{

return bossp.getPosition();

}

FloatRect getplayerbounds()

{

return bossp.getGlobalBounds();

}

int& gethp()

{

return hp;

}

int& getmaxhp()

{

return totalhp;

}

int& getdamagetime()

{

return damagetimer;

}

int& getmaxdamagetime()

{

return maxdamagetime;

}

int getlevel()

{

return level;

}

void motion()

{

if (x == false)

{

bossp.move(0, 4);

if (bossp.getPosition().y > 558)

{

x = true;

}

}

else

{

bossp.move(0, -4);

if (bossp.getPosition().y < 5)

{

x = false;

}

}

}

void combat()

{

if (movetimer >= maxmove)

{

weapon1.add(weapon(wtexture, bossp.getPosition(), Vector2f(-15, 0)));

movetimer -= 20;

}

else movetimer++;

for (int i = 0; i < weapon1.getsize(); i++)

{

weapon1[i].motion();

}

}

void draw()

{

window.draw(bossp);

for (int i = 0; i < weapon1.getsize(); i++)

{

weapon1[i].draw();

}

}

void update()

{

motion();

combat();

}

};

class genericEnemy

{

Texture\* texture;

Sprite gnsprite;

Texture \*ewtexture;

list<weapon> eweapon;

int shootdelay;

int type;

int hp;

int maxhp;

int damage;

int maxdamage;

public:

genericEnemy(Texture\* wt,Texture \*t,Vector2f positon)

{

texture = t;

ewtexture=wt;

shootdelay = 50;

gnsprite.setOrigin(0, 90);

gnsprite.setTexture(\*texture);

gnsprite.setScale(0.3f, 0.3f);

gnsprite.setPosition(positon);

type = 1;

hp = 10;

maxhp = 100;

damage = 1;

maxdamage = 2;

}

list<weapon>& getweapon()

{

return eweapon;

}

FloatRect getenemybounds()

{

return gnsprite.getGlobalBounds();

}

Vector2f getpos()

{

return gnsprite.getPosition();

}

void takeDamage()

{

hp -= damage;

if (hp <= 0)

{

hp = 0;

}

}

void motion()

{

if (shootdelay >= 100)

{

eweapon.add(weapon(ewtexture, gnsprite.getPosition(),Vector2f(-15,0)));

shootdelay -= 200;

}

else shootdelay++;

for (int i = 0; i < eweapon.getsize(); i++)

{

eweapon[i].motion();

}

}

void update()

{

motion();

switch (type)

{

case 1:

gnsprite.move(-2,0);

break;

}

}

void draw()

{

window.draw(gnsprite);

for (int i = 0; i < eweapon.getsize(); i++)

{

eweapon[i].draw();

}

}

Sprite& getenemy()

{

return gnsprite;

}

};

class Game

{

int x;

queue l;

adjacencyMatrix \*map;

menu Gmenu;

player\* Player;

boss\* b;

SoundBuffer buffer;

Sound explode;

Music music;

list<genericEnemy>Enemy;

int hp;

int score;

int keypressdelay;

Font font;

int enemyspawntime;

int maxenemyspawntime;

list<Text> staticText;

list<Text> Activetext;

Text bosshp;

Text gameover;

Text pause;

std::string s;

bool backcheck;

Texture ptexture;

Texture wtexture;

Texture etexture;

Texture ewtexture;

Texture btexture;

Text story;

Sprite distance;

RectangleShape path;

levelchanger levelchange;

int timer;

bool check;

bool levelendcheck;

bool levelchangelaunch;

bool savecheck;

Texture t;

Texture bwt;

LinkedList<string> storymessages;

Sprite back;

ifstream infile;

ofstream outfile;

public:

void prepareUI()

{

Text temp;

temp.setFont(font);

temp.setCharacterSize(16);

temp.setFillColor(sf::Color::Blue);

temp.setString("Health: "+std::to\_string(hp));

staticText.add(temp);

temp.setString("Score: " + std::to\_string(score));

staticText.add(temp);

temp.setString("level: " + std::to\_string(x));

staticText.add(temp);

}

Game()

{

infile.open("Messages/Level 1.txt");

while (!infile.eof())

{

getline(infile, s);

storymessages.add(s);

}

infile.close();

timer = 2400;

backcheck = false;

check = false;

levelendcheck = false;

levelchangelaunch = false;

savecheck = false;

map = new adjacencyMatrix(10);

map->addedge(0, 1,1);

map->addedge(1, 2, 1);

map->addedge(2, 3, 1);

map->addedge(3, 4, 1);

map->addedge(4, 5, 1);

map->addedge(1, 6, 13);

map->addedge(6, 2, 1);

music.openFromFile("Audio/Backgroundmusic.wav");

music.setVolume(20);

music.play();

music.setLoop(true);

buffer.loadFromFile("Audio/explosion.wav");

explode.setBuffer(buffer);

explode.setVolume(70);

ptexture.loadFromFile("Textures/Cruise Fighter12.png");

wtexture.loadFromFile("Textures/Weapon.png");

etexture.loadFromFile("Textures/EnemyType1.png");

ewtexture.loadFromFile("Textures/EnemyWeapon.png");

btexture.loadFromFile("Textures/boss1.png");

bwt.loadFromFile("Textures/EnemyWeapon.png");

font.loadFromFile("Fonts/arial.ttf");

distance.setTexture(ptexture);

distance.setScale(0.1,0.1);

path.setFillColor(Color::Blue);

path.setSize(Vector2f(1300, 40));

path.setPosition(0, 560);

gameover.setFont(font);

gameover.setCharacterSize(40);

gameover.setPosition(420, 150);

gameover.setFillColor(Color::Green);

pause.setFont(font);

pause.setCharacterSize(40);

pause.setPosition(530, 250);

pause.setFillColor(Color::Blue);

pause.setString("Paused");

bosshp.setFont(font);

bosshp.setCharacterSize(16);

bosshp.setPosition(1200, 10);

bosshp.setFillColor(Color::Red);

Player = new player(&this->ptexture,&this->wtexture);

b = new boss(&this->btexture, &this->bwt);

hp = Player->gethp();

score = Player->getscore();

window.setFramerateLimit(60);

maxenemyspawntime=20;

enemyspawntime=maxenemyspawntime;

prepareUI();

x = 0;

keypressdelay = 25;

}

void ui()

{

for (int i = 0; i < Activetext.getsize(); i++)

{

Activetext[i].setPosition(Player->getpos());

}

for (int i = 0; i < staticText.getsize(); i++)

{

staticText[i].setPosition(10, 10+i\*20);

}

}

void levelend(bool &endcheck)

{

if (endcheck == true)

{

for (int i = 0; i < Enemy.getsize(); i++)

{

Enemy.remove(i);

}

Text t1;

t1.setFont(font);

t1.setCharacterSize(45);

t1.setFillColor(Color::Blue);

t1.setString("Congratulations! you have finished this level");

t1.setPosition(222, 205);

window.draw(t1);

t1.setCharacterSize(25);

t1.setString("Press S to Save game");

t1.setPosition(822, 465);

window.draw(t1);

Text t;

t.setFont(font);

t.setCharacterSize(25);

t.setFillColor(Color::Blue);

t.setString("Press enter to Continue");

t.setPosition(822, 500);

window.draw(t);

if (Keyboard::isKeyPressed(Keyboard::Enter))

{

levelchangelaunch = true;

endcheck = false;

}

if (Keyboard::isKeyPressed(Keyboard::S))

{

outfile.open("Save file/save.txt",ios::trunc);

outfile << x <<"\t"<< Player->gethp() <<"\t"<< Player->getscore() <<"\t"<< true<<"\t"<<true;

outfile.close();

x = 0;

window.close();

}

}

if (levelchangelaunch == true)

{

bool t = false;

bool pen = false;

window.clear();

levelchange.draw();

levelchange.update(\*map,x,t,pen);

if (t == true)

{

x = x + 1;

Player->gethp() = Player->getmaxhp();

levelchangelaunch = false;

timer = 2400;

}

else if(pen==true)

{

levelchangelaunch = false;

timer = 2400;

}

}

}

void setbackground(string background)

{

t.loadFromFile("Textures/" + background);

back.setTexture(t);

}

void drawback()

{

window.draw(back);

}

void storymessage()

{

for (int i = 0; i < storymessages.size(); i++)

{

Text t;

t.setFont(font);

t.setCharacterSize(25);

t.setFillColor(Color::Blue);

t.setString(storymessages[i]);

t.setPosition(122, 15 + 40 \* i);

window.draw(t);

}

Text t;

t.setFont(font);

t.setCharacterSize(25);

t.setFillColor(Color::Blue);

t.setString("Press enter to Continue");

t.setPosition(822, 500);

window.draw(t);

if (keypressdelay >= 40)

{

if (Keyboard::isKeyPressed(Keyboard::Enter))

{

check = true;

keypressdelay = 25;

}

}

keypressdelay += 5;

}

void playercombat()

{

staticText[0].setString("Health: " + std::to\_string(Player->gethp()));

staticText[1].setString("Score: " + std::to\_string(Player->getscore()));

staticText[2].setString("Level: " + std::to\_string(x));

for (int i = 0; i < Player->getweapon().getsize(); i++)

{

Player->getweapon()[i].update();

if (Player->getweapon()[i].getpos().x >= window.getSize().x)

{

Player->getweapon().remove(i);

}

for (int k = 0; k < Enemy.getsize(); k++)

{

if (i< Player->getweapon().getsize()&&Player->getweapon()[i].getweaponbounds().intersects(Enemy[k].getenemybounds()))

{

explode.play();

Enemy.remove(k);

Player->getweapon().remove(i);

Player->getscore() += 10;

}

}

}

for (int i = 0; i < Enemy.getsize(); i++)

{

if (Player->getplayerbounds().intersects(Enemy[i].getenemybounds()))

{

if (Player->getdamagetime() >= Player->getmaxdamagetime())

{

if (Player->gethp() > 1)

{

Player->gethp()--;

Player->getdamagetime() -= 20;

Enemy.remove(i);

explode.play();

}

else

{

explode.play();

x = 10;

}

}

}

}

for (int i = 0; i < Enemy.getsize(); i++)

{

Enemy[i].update();

if (Enemy[i].getpos().x < -50|| Enemy[i].getpos().y < -50|| Enemy[i].getpos().y > 650)

{

Enemy.remove(i);

}

}

}

void enemycombat()

{

for (int k=0; k < Enemy.getsize(); k++)

{

for (int i = 0; i < Enemy[k].getweapon().getsize(); i++)

{

Enemy[k].getweapon()[i].update();

if (Enemy[k].getweapon()[i].getpos().x < 0)

{

Enemy[k].getweapon().remove(i);

}

if (i < Enemy[k].getweapon().getsize() && Enemy[k].getweapon()[i].getweaponbounds().intersects(Player->getplayerbounds()))

{

explode.play();

Player->gethp()--;

if (Player->gethp() == 0)

{

x = 10;

}

Enemy[k].getweapon().remove(i);

}

}

}

}

void enemyspawn()

{

if (enemyspawntime < maxenemyspawntime)

{

enemyspawntime++;

}

if (enemyspawntime >= maxenemyspawntime)

{

srand(time(NULL));

float y = rand() % 500 + 100;

Enemy.add(genericEnemy(&ewtexture,&etexture, Vector2f(1300, rand() % 500)));

enemyspawntime -= 50;

}

}

void bosscombat()

{

for (int k = 0; k < b->getweapon1().getsize(); k++)

{

cout << "weapon no " << k << ": " << b->getweapon1()[k].getpos().x << endl;

if (b->getweapon1()[k].getpos().x < 0)

{

b->getweapon1().remove(k);

}

if (b->getweapon1()[k].getweaponbounds().intersects(Player->getplayerbounds()))

{

explode.play();

b->getweapon1().remove(k);

Player->gethp() -= 2;

if (Player->gethp() == 0)

{

x = 10;

}

}

}

for (int i = 0; i < Player->getweapon().getsize(); i++)

{

if (Player->getweapon()[i].getweaponbounds().intersects(b->getplayerbounds()))

{

explode.play();

Player->getweapon().remove(i);

b->gethp() -= 1;

}

}

if (Player->getplayerbounds().intersects(b->getplayerbounds()))

{

explode.play();

x = 10;

}

}

void level1update()

{

if (timer > 0)

{

string s1;

Player->update();

enemycombat();

playercombat();

enemyspawn();

ui();

timer--;

}

else

{

levelendcheck = true;

backcheck = false;

}

}

void level1draw()

{

if (check == false)

{

storymessage();

}

else

{

if (backcheck == false)

{

setbackground("Low Earth orbit.png");

backcheck = true;

}

drawback();

Player->draw();

srand(time(NULL));

for (int i = 0; i < Enemy.getsize(); i++)

{

Enemy[i].getenemy().move(0, rand() % 4 - 1);

Enemy[i].draw();

}

drawUI();

}

}

void level2update()

{

if (timer > 0)

{

Player->update();

enemycombat();

playercombat();

etexture.loadFromFile("Textures/EnemyType2.png");

enemyspawn();

ui();

timer--;

}

else

{

levelendcheck = true;

backcheck = false;

}

}

void level2draw()

{

if (backcheck == false)

{

setbackground("Distant Earth.png");

backcheck = true;

}

drawback();

Player->draw();

srand(time(NULL));

for (int i = 0; i < Enemy.getsize(); i++)

{

Enemy[i].getenemy().move(0, rand() % 4 - 1);

Enemy[i].draw();

}

drawUI();

}

void level3update()

{

if (b->gethp() > 0)

{

bosshp.setString("Boss HP: " + std::to\_string(b->gethp()));

Player->update();

b->update();

bosscombat();

playercombat();

ui();

}

else

{

levelendcheck = true;

backcheck = false;

}

}

void level3draw()

{

if (backcheck == false)

{

setbackground("Earth and moon.png");

backcheck = true;

}

drawback();

Player->draw();

b->draw();

drawUI();

window.draw(bosshp);

}

void update()

{

if (Keyboard::isKeyPressed(Keyboard::Escape))

{

if (keypressdelay >= 25)

{

if (x == 11)

{

x = 1;

}

else x = 11;

keypressdelay -= 5;

}

keypressdelay++;

}

switch (x)

{

case 1:

level1update();

break;

case 2:

level2update();

break;

case 3:

level3update();

break;

case 4:

break;

default:

Gmenu.update(Player->gethp(), Player->getscore(), timer);

break;

}

}

void drawUI()

{

for (int i = 0; i < Activetext.getsize(); i++)

{

window.draw(Activetext[i]);

}

for (int i = 0; i < staticText.getsize(); i++)

{

window.draw(staticText[i]);

}

}

void draw()

{

window.clear();

distance.setPosition(1300-timer, 570);

if (levelendcheck == false)

{

switch (x)

{

case 1:

level1draw();

break;

case 2:

level2draw();

break;

case 3:

level3draw();

break;

case 4:

break;

case 10:

gameover.setString("Alas! You have crashed");

window.draw(gameover);

break;

case 11:

window.draw(pause);

break;

default:

Gmenu.draw();

x = Gmenu.getselector();

}

}

else levelend(levelendcheck);

if (x >= 1 && x <= 6)

{

window.draw(path);

window.draw(distance);

}

window.display();

}

~Game()

{

outfile.close();

delete Player;

}

};

int main()

{

Game game;

while (window.isOpen())

{

sf::Event evnt;

window.clear();

while (window.pollEvent(evnt))

{

switch (evnt.type)

{

case sf::Event::Closed:

window.close();

exit(1);

break;

}

}

game.update();

game.draw();

}

}

