#include "stdafx.h"

#include <sstream>

#include <SFML/Graphics.hpp>

#include "ZombieArena.h"

#include "Player.h"

#include "TextureHolder.h"

#include "Bullet.h"

#include "Pickup.h"

using namespace sf;

int main()

{

// Here is the instabce of TextureHolder

TextureHolder holder;

// The game will always be in one of four states

enum class State { PAUSED, LEVELING\_UP, GAME\_OVER, PLAYING };

// Start with the GAME\_OVER state

State state = State::GAME\_OVER;

// Get the screen resolution and create an SFML window

Vector2f resolution;

resolution.x = VideoMode::getDesktopMode().width;

resolution.y = VideoMode::getDesktopMode().height;

RenderWindow window(VideoMode(resolution.x, resolution.y),

"Zombie Arena", Style::Fullscreen);

// Create a an SFML View for the main action

View mainView(sf::FloatRect(0, 0, resolution.x, resolution.y));

// Here is our clock for timing everything

Clock clock;

// How long has the PLAYING state been active

Time gameTimeTotal;

// Where is the mouse in relation to world coordinates

Vector2f mouseWorldPosition;

// Where is the mouse in relation to screen coordinates

Vector2i mouseScreenPosition;

// Create an instance of the Player class

Player player;

// The boundaries of the arena

IntRect arena;

// Create the background

VertexArray background;

// Load the texture for our background vertex array

Texture textureBackground = TextureHolder::GetTexture(

"graphics/background\_sheet.png");

// Prepare for a horde of zombies

int numZombies;

int numZombiesAlive;

Zombie\* zombies = NULL;

// 100 bullets should do

Bullet bullets[100];

int currentBullet = 0;

int bulletsSpare = 24;

int bulletsInClip = 6;

int clipSize = 6;

float fireRate = 1;

// When was the fire button last pressed?

Time lastPressed;

// Hide the mouse pointer and replace it with crosshair

window.setMouseCursorVisible(false);

Sprite spriteCrosshair;

Texture textureCrosshair = TextureHolder::GetTexture("graphics/crosshair.png");

spriteCrosshair.setTexture(textureCrosshair);

spriteCrosshair.setOrigin(25, 25);

// Create a couple of pickups

Pickup healthPickup(1);

Pickup ammoPickup(2);

// About the game

int score = 0;

int hiScore = 0;

// For the home/game over screen

Sprite spriteGameOver;

Texture textureGameOver = TextureHolder::GetTexture

("graphics/background1.png");

spriteGameOver.setTexture(textureGameOver);

spriteGameOver.setPosition(0, 0);

// Create a view for the HUD

View hudView(sf::FloatRect(0, 0, resolution.x, resolution.y));

// Create a sprite for the ammo icon

Sprite spriteAmmoIcon;

Texture textureAmmoIcon = TextureHolder::GetTexture("graphics/ammo\_icon.png");

spriteAmmoIcon.setTexture(textureAmmoIcon);

spriteAmmoIcon.setPosition(28, 592);

// Load the font

Font font;

font.loadFromFile("fonts/zombiecontrol.ttf");

// Paused

Text pausedText;

pausedText.setFont(font);

pausedText.setCharacterSize(75);

pausedText.setFillColor(Color::White);

pausedText.setPosition(350, 200);

pausedText.setString("Press Enter \nto continue");

// Game Over

Text gameOverText;

gameOverText.setFont(font);

gameOverText.setCharacterSize(75);

gameOverText.setFillColor(Color::White);

gameOverText.setPosition(350, 200);

gameOverText.setString("Press Enter to play");

// Levelling up

Text levelUpText;

levelUpText.setFont(font);

levelUpText.setCharacterSize(75);

levelUpText.setFillColor(Color::White);

levelUpText.setPosition(50, 150);

std::stringstream levelUpStream;

levelUpStream <<

"1- Increased rate of fire" <<

"\n2- Increased clip size(next reload)" <<

"\n3- Increased max health" <<

"\n4- Increased run speed" <<

"\n5- More and better health pickups" <<

"\n6- More and better ammo pickups";

levelUpText.setString(levelUpStream.str());

// Ammo

Text ammoText;

ammoText.setFont(font);

ammoText.setCharacterSize(55);

ammoText.setFillColor(Color::White);

ammoText.setPosition(116, 592);

// Score

Text scoreText;

scoreText.setFont(font);

scoreText.setCharacterSize(55);

scoreText.setFillColor(Color::White);

scoreText.setPosition(20, 0);

// Hi Score

Text hiScoreText;

hiScoreText.setFont(font);

hiScoreText.setCharacterSize(55);

hiScoreText.setFillColor(Color::White);

hiScoreText.setPosition(980, 0);

std::stringstream s;

s << "Hi Score:" << hiScore;

hiScoreText.setString(s.str());

// Zombies remaining

Text zombiesRemainingText;

zombiesRemainingText.setFont(font);

zombiesRemainingText.setCharacterSize(55);

zombiesRemainingText.setFillColor(Color::White);

zombiesRemainingText.setPosition(935, 592);

zombiesRemainingText.setString("Zombies: 100");

// Wave number

int wave = 0;

Text waveNumberText;

waveNumberText.setFont(font);

waveNumberText.setCharacterSize(55);

waveNumberText.setFillColor(Color::White);

waveNumberText.setPosition(725, 592);

waveNumberText.setString("Wave: 0");

// Health bar

RectangleShape healthBar;

healthBar.setFillColor(Color::Red);

healthBar.setPosition(261, 592);

// When did we last update the HUD?

int framesSinceLastHUDUpdate = 0;

// What time was the last update

Time timeSinceLastUpdate;

// How often (in frames) should we update the HUD

int fpsMeasurementFrameInterval = 1000;

// The main game loop

while (window.isOpen())

{

/\*

\*\*\*\*\*\*\*\*\*\*\*\*

Handle input

\*\*\*\*\*\*\*\*\*\*\*\*

\*/

// Handle events

Event event;

while (window.pollEvent(event))

{

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

{

// Pause a game while playing

if (event.key.code == Keyboard::Return &&

state == State::PLAYING)

{

state = State::PAUSED;

}

// Restart while paused

else if (event.key.code == Keyboard::Return &&

state == State::PAUSED)

{

state = State::PLAYING;

// Reset the clock so there isn't a frame jump

clock.restart();

}

// Start a new game while in GAME\_OVER state

else if (event.key.code == Keyboard::Return &&

state == State::GAME\_OVER)

{

state = State::LEVELING\_UP;

}

if (state == State::PLAYING)

{

// Reloading

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

{

if (bulletsSpare >= clipSize)

{

// Plenty of bullets. Reload.

bulletsInClip = clipSize;

bulletsSpare -= clipSize;

}

else if (bulletsSpare > 0)

{

// Only few bullets left

bulletsInClip = bulletsSpare;

bulletsSpare = 0;

}

else

{

// More here soon?!

}

}

}

}

}// End event polling

// Handle the player quitting

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

{

window.close();

}

// Handle controls while playing

if (state == State::PLAYING)

{

// Handle the pressing and releasing of the WASD keys

if (Keyboard::isKeyPressed(Keyboard::W))

{

player.moveUp();

}

else

{

player.stopUp();

}

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

{

player.moveDown();

}

else

{

player.stopDown();

}

if (Keyboard::isKeyPressed(Keyboard::A))

{

player.moveLeft();

}

else

{

player.stopLeft();

}

if (Keyboard::isKeyPressed(Keyboard::D))

{

player.moveRight();

}

else

{

player.stopRight();

}

// Fire a bullet

if (sf::Mouse::isButtonPressed(sf::Mouse::Left))

{

if (gameTimeTotal.asMilliseconds()

- lastPressed.asMilliseconds()

> 1000 / fireRate && bulletsInClip > 0)

{

// Pass the centre of the player and the centre of the crosshair

// to the shoot function

bullets[currentBullet].shoot(

player.getCenter().x, player.getCenter().y,

mouseWorldPosition.x, mouseWorldPosition.y);

currentBullet++;

if (currentBullet > 99)

{

currentBullet = 0;

}

lastPressed = gameTimeTotal;

bulletsInClip--;

}

}// End fire a bullet

}// End WASD while playing

// Handle the levelling up state

if (state == State::LEVELING\_UP)

{

// Handle the player levelling up

if (event.key.code == Keyboard::Num1)

{

state = State::PLAYING;

}

if (event.key.code == Keyboard::Num2)

{

state = State::PLAYING;

}

if (event.key.code == Keyboard::Num3)

{

state = State::PLAYING;

}

if (event.key.code == Keyboard::Num4)

{

state = State::PLAYING;

}

if (event.key.code == Keyboard::Num5)

{

state = State::PLAYING;

}

if (event.key.code == Keyboard::Num6)

{

state = State::PLAYING;

}

if (state == State::PLAYING)

{

// Prepare thelevel

// We will modify the next two lines later

arena.width = 500 ;

arena.height = 500 ;

arena.left = 0;

arena.top = 0;

// Pass the vertex array by reference

// to the createBackground function

int tileSize = createBackground(background, arena);

// Spawn the player in the middle of the arena

player.spawn(arena, resolution, tileSize);

// Configure the pick-ups

healthPickup.setArena(arena);

ammoPickup.setArena(arena);

// Create a horde of zombies

numZombies = 10;

// Delete the previously allocated memory (if it exists)

delete[] zombies;

zombies = createHorde(numZombies, arena);

numZombiesAlive = numZombies;

// Reset the clock so there isn't a frame jump

clock.restart();

}

}// End levelling up

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

UPDATE THE FRAME

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

if (state == State::PLAYING)

{

// Update the delta time

Time dt = clock.restart();

// Update the total game time

gameTimeTotal += dt;

// Make a decimal fraction of 1 from the delta time

float dtAsSeconds = dt.asSeconds();

// Where is the mouse pointer

mouseScreenPosition = Mouse::getPosition();

// Convert mouse position to world coordinates of mainView

mouseWorldPosition = window.mapPixelToCoords(

Mouse::getPosition(), mainView);

// Set the crosshair to the mouse world location

spriteCrosshair.setPosition(mouseWorldPosition);

// Update the player

player.update(dtAsSeconds, Mouse::getPosition());

// Make a note of the players new position

Vector2f playerPosition(player.getCenter());

// Make the view centre around the player

mainView.setCenter(player.getCenter());

// Loop through each Zombie and update them

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

{

if (zombies[i].isAlive())

{

zombies[i].update(dt.asSeconds(), playerPosition);

}

}

// Update any bullets that are in-flight

for (int i = 0; i < 100; i++)

{

if (bullets[i].isInFlight())

{

bullets[i].update(dtAsSeconds);

}

}

// Update the pickups

healthPickup.update(dtAsSeconds);

ammoPickup.update(dtAsSeconds);

// Collision detection

// Have any zombies been shot?

for (int i = 0; i < 100; i++)

{

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

{

if (bullets[i].isInFlight() &&

zombies[j].isAlive())

{

if (bullets[i].getPosition().intersects

(zombies[j].getPosition()))

{

// Stop the bullet

bullets[i].stop();

// Register the hit and see if it was a kill

if (zombies[j].hit()) {

// Not just a hit but a kill too

score += 10;

if (score >= hiScore)

{

hiScore = score;

}

numZombiesAlive--;

// When all the zombies are dead (again)

if (numZombiesAlive == 0) {

state = State::LEVELING\_UP;

}

}

}

}

}

}// End zombie being shot

// Have any zombies touched the player

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

{

if (player.getPosition().intersects

(zombies[i].getPosition()) && zombies[i].isAlive())

{

if (player.hit(gameTimeTotal))

{

//More here later

}

}

}// End player touched

// Has the player touched health pickup

if (player.getPosition().intersects

(healthPickup.getPosition()) && healthPickup.isSpawned())

{

player.increaseHealthLevel(healthPickup.gotIt());

}

// Has the player touched ammo pickup

if (player.getPosition().intersects

(ammoPickup.getPosition()) && ammoPickup.isSpawned())

{

bulletsSpare += ammoPickup.gotIt();

}

// size up the health bar

healthBar.setSize(Vector2f(player.getHealth() \* 3, 70));

// Increment the number of frames since the last HUD calculation

framesSinceLastHUDUpdate++;

// Calculate FPS every fpsMeasurementFrameInterval frames

if (framesSinceLastHUDUpdate > fpsMeasurementFrameInterval)

{

// Update game HUD text

std::stringstream ssAmmo;

std::stringstream ssScore;

std::stringstream ssHiScore;

std::stringstream ssWave;

std::stringstream ssZombiesAlive;

// Update the ammo text

ssAmmo << bulletsInClip << "/" << bulletsSpare;

ammoText.setString(ssAmmo.str());

// Update the score text

ssScore << "Score:" << score;

scoreText.setString(ssScore.str());

// Update the high score text

ssHiScore << "Hi Score:" << hiScore;

hiScoreText.setString(ssHiScore.str());

// Update the wave

ssWave << "Wave:" << wave;

waveNumberText.setString(ssWave.str());

// Update the high score text

ssZombiesAlive << "Zombies:" << numZombiesAlive;

zombiesRemainingText.setString(ssZombiesAlive.str());

framesSinceLastHUDUpdate = 0;

}// End HUD update

}// End updating the scene

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Draw the scene

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

if (state == State::PLAYING)

{

window.clear();

// set the mainView to be displayed in the window

// And draw everything related to it

window.setView(mainView);

// Draw the background

window.draw(background, &textureBackground);

// Draw the zombies

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

{

window.draw(zombies[i].getSprite());

}

for (int i = 0; i < 100; i++)

{

if (bullets[i].isInFlight())

{

window.draw(bullets[i].getShape());

}

}

// Draw the player

window.draw(player.getSprite());

// Draw the pickups is currently spawned

if (ammoPickup.isSpawned())

{

window.draw(ammoPickup.getSprite());

}

if (healthPickup.isSpawned())

{

window.draw(healthPickup.getSprite());

}

//Draw the crosshair

window.draw(spriteCrosshair);

// Switch to the HUD view

window.setView(hudView);

// Draw all the HUD elements

window.draw(spriteAmmoIcon);

window.draw(ammoText);

window.draw(scoreText);

window.draw(hiScoreText);

window.draw(healthBar);

window.draw(waveNumberText);

window.draw(zombiesRemainingText);

}

if (state == State::LEVELING\_UP)

{

window.draw(spriteGameOver);

window.draw(levelUpText);

}

if (state == State::PAUSED)

{

window.draw(pausedText);

}

if (state == State::GAME\_OVER)

{

window.draw(spriteGameOver);

window.draw(gameOverText);

window.draw(scoreText);

window.draw(hiScoreText);

}

window.display();

}// End game loop

return 0;

}