// (c) Martyn Prigmore 2013 and beyond. All rights reserved.

//

// Defines the classes and helper methods declared in SpriteGame.h

//

// Notes

// 1) In source code files I use namespaces to avoid the need for fully qualified identifiers.

//Note: pch.h is a set of pre-compiled headers, providing access to the major system

// headers. In our case this includes various DirectX header files.

#include "pch.h"

#include <SpriteGame.h>

//Include DirectXTK libraries

#include <WICTextureLoader.h>

#include <CommonStates.h>

#include <SimpleMath.h>

#include <SpriteBatch.h>

#include <Xinput.h>

#include <stdio.h> /\* printf, NULL \*/

#include <stdlib.h> /\* srand, rand \*/

#include <time.h> /\* time \*/

using namespace Microsoft::WRL;

using namespace Windows::UI::Core;

using namespace Windows::Graphics::Display;

using namespace DirectX;

using namespace Platform;

using namespace DirectX::SimpleMath;

SpriteGame::SpriteGame() :

d3dDevice ( nullptr )

, d3dDeviceContext ( nullptr )

, swapChain ( nullptr )

, backBufferTarget ( nullptr )

, d3dFeatureLevel ( D3D\_FEATURE\_LEVEL\_11\_1 )

//Initialize the DirectXTK objects

, spriteBatch ( nullptr )

, commonStates ( nullptr )

// Initialize the Game World objects

// Initialize the rocky Rock object

, rockTexture ( nullptr )

, rockPosition (10.0f, 10.0f)

, rockColor ( 1.0f, 0.85f, 0.0f, 1.0f )

// Initialize the organised Cabinet object

, cabinetTexture ( nullptr )

, cabinetPosition (100.0f, 10.0f)

, cabinetColor ( 1.0f, 1.0f, 1.0f, 1.0f )

, cabinetSourceRect ( nullptr )

// Initialize the animated Grass object

, grassTexture ( nullptr )

, grassPosition (260.0f, 10.0f)

, grassColor ( 1.0f, 1.0f, 1.0f, 1.0f )

, grassSourceRect ( nullptr )

, grassFrame ( 0 )

, grassLastFrameChange ( 0 )

, grassSheetLength ( 4 )

, grassIncrement ( 1 )

// Initialize the almighty Square

, squareTexture ( nullptr )

, squarePosition ( 10.0f, 100.0f )

, squareColor ( 1.0f, 1.0f, 1.0f, 1.0f )

, squareSourceRect ( nullptr )

, squareVelocity ( 2.0f )

// Initialize the fear-inspiring Barrels

, barrelTexture ( nullptr )

, barrelPosition ( 0.0f, 0.0f )

, barrelColor ( 1.0f, 1.0f, 1.0f, 1.0f )

, barrelSourceRect ( nullptr )

, barrelFrameHeight ( 64 )

, barrelFrameWidth ( 127 )

// Initialize the drunk, flipped Barrels

, flippedTexture ( nullptr )

, flippedPosition ( 10.0f, 250.0f )

, flippedColor (1.0f, 1.0f, 1.0f, 1.0f)

, flippedSourceRect ( nullptr )

// Initialize the partial Cabinets

, partialTexture ( nullptr )

, partialPosition ( 150.0f, 250.0f )

, partialColor (1.0f, 1.0f, 1.0f, 1.0f)

, partialSourceRect ( nullptr )

// Initialize the cupboard

, cupboardTexture ( nullptr )

, cupboardPosition ( 182.0f, 250.0f )

, cupboardColor (1.0f, 1.0f, 1.0f, 1.0f)

, cupboardSourceRect ( nullptr )

// Initialize the trophy

, trophyTexture ( nullptr )

, trophyPosition ( 214.0f, 250.0f )

, trophyColor (1.0f, 1.0f, 1.0f, 1.0f)

, trophySourceRect ( nullptr )

// Initialize the glass

, glassTexture ( nullptr )

, glassPosition ( 182.0f, 250.0f )

, glassColor (1.0f, 1.0f, 1.0f, 1.0f)

, glassSourceRect (nullptr)

// Initialize the reversed animation grass sprite

, reverseTexture ( nullptr )

, reversePosition (260.0f, 250.0f)

, reverseColor ( 1.0f, 1.0f, 1.0f, 1.0f )

, reverseSourceRect ( nullptr )

, reverseFrame ( 0 )

, reverseLastFrameChange ( 0 )

, reverseSheetLength ( 4 )

, reverseSubtraction ( 1 )

// Initialize the related disc variables

, discTexture (nullptr )

, discPosition ( 0.0f, 0.0f )

, discColor (1.0f, 1.0f, 1.0f, 1.0f )

, discVelocity ( 2.0f, 5.0f )

, discSourceRect (nullptr)

, leftPosition (0.0f, 0.0f)

, rightPosition (200.0f, 0.0f)

, topPosition (0.0f, 400.0f)

, bottomPosition (0.0f, 600.0f)

// Initialize the plasma affected rock

, plasmaTexture ( nullptr )

, plasmaPosition ( 400.0f, 800.0f )

, plasmaColor (1.0f, 1.0f, 1.0f, 1.0f)

, plasmaSourceRect (nullptr)

, addToThisColor (0.001f, 0.0f, 0.0f, 0.0f)

, addGreen (0.0f, -0.001f, 0.0f, 0.0f)

, addBlue (0.0f, 0.0f, -0.01f, 0.0f)

// Background Colour

, clearToThisColor (DirectX::Colors::White.v)

// Initialize the worm variables

, wormTexture (nullptr)

, wormPosition (800.0f, 600.0f)

, wormColor (1.0f, 1.0f, 1.0f, 1.0f)

, wormSourceRect (nullptr)

, wormColumn (0)

, wormRow (0)

, wormFrameWidth (35)

, wormFrameHeight (50)

, wormLastFrameChange ( 0 )

, wormFrameRate (3)

, wormSheetWidth (3)

// Initialize the gnome/troll variables

, creatureFrameHeight (64)

, creatureFrameWidth (64)

, creatureSheetWidth (6)

, creaturePosition (650.0f, 900.0f)

, creatureColor (1.0f, 1.0f, 1.0f, 1.0f)

, creatureSourceRect (nullptr)

, creatureFrame (0)

, creatureLastFrameChange (0)

, creatureFrameRate (6)

, creatureGnomeTexture (nullptr)

, creautureTrollTexture (nullptr)

, creatureCurrentTexture (nullptr)

// Controlled Moving Disc

, movingDiscTexture (nullptr)

, movingDiscPosition (0.0f, 0.0f)

, movingDiscColor (1.0f, 1.0f, 1.0f, 1.0f)

, movingDiscboundingBox (0.0f, 0.0f, 32.0f, 32.0f)

, movingDiscSpeed (500.0f)

// The troll!

, trollTexture (nullptr)

, trollPosition (rand())

, trollColor (1.0f, 1.0f, 1.0f, 1.0f)

, trollSourceRect (nullptr)

, trollFrameHeight (63)

, trollFrameWidth (64)

// gamepad

, gamepad ( )

// Tracking ticks

, trackTicks ( )

, trackTrigger ( )

//Interval

, interval ( 0 )

//Useful tools

, changeColor (0.0f, 0.0f, 1.0f, 1.0f)

{

}

SpriteGame::~SpriteGame()

{

}

void SpriteGame::CreateDeviceIndependentResources()

{

//None for a basic game

}

void SpriteGame::CreateDeviceResources()

{

UINT creationFlags = D3D11\_CREATE\_DEVICE\_BGRA\_SUPPORT; //For Direct2D compatibility

D3D\_FEATURE\_LEVEL featureLevels[] =

{

D3D\_FEATURE\_LEVEL\_11\_1,

D3D\_FEATURE\_LEVEL\_11\_0,

D3D\_FEATURE\_LEVEL\_10\_1,

D3D\_FEATURE\_LEVEL\_10\_0,

};

unsigned int totalFeatureLevels = ARRAYSIZE ( featureLevels );

//First, get a D3D11 device and device context

ComPtr<ID3D11Device> device;

ComPtr<ID3D11DeviceContext> context;

ThrowIfFailed(

D3D11CreateDevice(

nullptr,

D3D\_DRIVER\_TYPE\_HARDWARE, //Could loop through D3D\_DRIVER\_TYPE\_WARP and D3D\_DRIVER\_TYPE\_REFERENCE in case no graphics card

nullptr,

creationFlags,

featureLevels,

totalFeatureLevels,

D3D11\_SDK\_VERSION,

&device,

&this->d3dFeatureLevel,

&context

)

);

//Second, convert the D3D11 objects to D3D11.1

ThrowIfFailed(

device.As(&this->d3dDevice)

);

ThrowIfFailed(

context.As(&this->d3dDeviceContext)

);

}

void SpriteGame::CreateWindowSizeDependentResources()

{

this->windowBounds = this->coreWindow->Bounds; //measured in DIPs

float windowHeightPhysicalPixels = ConvertDipsToPixels(this->windowBounds.Height);

float windowWidthPhysicalPixels = ConvertDipsToPixels(this->windowBounds.Width);

//Swap height and width if the window is not in landscape

this->displayOrientation = DisplayProperties::CurrentOrientation;

if (this->displayOrientation == DisplayOrientations::Portrait

|| this->displayOrientation == DisplayOrientations::PortraitFlipped)

{

this->backBufferTargetSize.Height = windowWidthPhysicalPixels;

this->backBufferTargetSize.Width = windowHeightPhysicalPixels;

}

else

{

this->backBufferTargetSize.Height = windowHeightPhysicalPixels;

this->backBufferTargetSize.Width = windowWidthPhysicalPixels;

}

if (this->swapChain != nullptr)

{

//Resize the existing swap chain

ThrowIfFailed(

this->swapChain->ResizeBuffers(

2,

static\_cast<UINT>(this->backBufferTargetSize.Width),

static\_cast<UINT>(this->backBufferTargetSize.Height),

DXGI\_FORMAT\_B8G8R8A8\_UNORM,

0

)

);

}

else

{

//Create a swap chain

DXGI\_SWAP\_CHAIN\_DESC1 swapChainDesc = {0};

swapChainDesc.Width = static\_cast<UINT>(this->backBufferTargetSize.Width);

swapChainDesc.Height = static\_cast<UINT>(this->backBufferTargetSize.Height);

swapChainDesc.Format = DXGI\_FORMAT\_B8G8R8A8\_UNORM;

swapChainDesc.Stereo = false;

swapChainDesc.SampleDesc.Count = 1;

swapChainDesc.SampleDesc.Quality = 0;

swapChainDesc.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;

swapChainDesc.BufferCount = 2;

swapChainDesc.Scaling = DXGI\_SCALING\_NONE;

swapChainDesc.SwapEffect = DXGI\_SWAP\_EFFECT\_FLIP\_SEQUENTIAL;

swapChainDesc.Flags = 0;

ComPtr<IDXGIDevice1> dxgiDevice;

ThrowIfFailed(

this->d3dDevice.As(&dxgiDevice)

);

ComPtr<IDXGIAdapter> dxgiAdapter;

ThrowIfFailed(

dxgiDevice->GetAdapter(&dxgiAdapter)

);

ComPtr<IDXGIFactory2> dxgiFactory;

ThrowIfFailed(

dxgiAdapter->GetParent(\_\_uuidof(IDXGIFactory2), &dxgiFactory)

);

Windows::UI::Core::CoreWindow^ ourWindow = this->coreWindow.Get();

ThrowIfFailed(

dxgiFactory->CreateSwapChainForCoreWindow(

this->d3dDevice.Get(),

reinterpret\_cast<IUnknown\*>(ourWindow),

&swapChainDesc,

nullptr,

&this->swapChain

)

);

ThrowIfFailed(

dxgiDevice->SetMaximumFrameLatency(1)

);

}

//Create the render target view from the swap chain back buffer

ComPtr<ID3D11Texture2D> backBuffer;

ThrowIfFailed(

this->swapChain->GetBuffer(

0,

\_\_uuidof(ID3D11Texture2D),

&backBuffer

)

);

ThrowIfFailed(

this->d3dDevice->CreateRenderTargetView(

backBuffer.Get(),

nullptr,

&this->backBufferTarget

)

);

//Set the rendering viewport to target the entire window

CD3D11\_VIEWPORT viewport(

0.0f,

0.0f,

this->backBufferTargetSize.Width,

this->backBufferTargetSize.Height

);

this->d3dDeviceContext->RSSetViewports(1, &viewport);

}

void SpriteGame::Initialize(CoreWindow^ ourWindow)

{

this->coreWindow = ourWindow;

CreateDeviceIndependentResources();

CreateDeviceResources();

CreateWindowSizeDependentResources();

//Initialize member variables

//Create new SpriteBatch and CommonStates objects and pass their locations to

// our pointer member variables.

this->spriteBatch = std::unique\_ptr<SpriteBatch>(new SpriteBatch(this->d3dDeviceContext.Get()));

this->commonStates = std::unique\_ptr<CommonStates>(new CommonStates(this->d3dDevice.Get()));

this->spriteFont = std::unique\_ptr<SpriteFont>(new SpriteFont(this->d3dDevice.Get(), L"Assets/italic.spritefont"));

this->eleFont = std::unique\_ptr<SpriteFont>(new SpriteFont(this->d3dDevice.Get(), L"Assets/Elephant.spritefont"));

//Load the texture object using a DirectXTK function

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/rock.png",

0,

&this->rockTexture

)

);

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/cabinets.png",

0,

&this->cabinetTexture

)

);

this->cabinetSourceRect = std::unique\_ptr<RECT>(new RECT());

this->cabinetSourceRect->left = 0;

this->cabinetSourceRect->top = 5;

this->cabinetSourceRect->right = 128;

this->cabinetSourceRect->bottom = 74;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/SpriteAnimation.png",

0,

&this->grassTexture

)

);

this->grassSourceRect = std::unique\_ptr<RECT>(new RECT());

this->grassSourceRect->left = 0;

this->grassSourceRect->top = 0;

this->grassSourceRect->right = 175;

this->grassSourceRect->bottom = 200;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/Square.png",

0,

&this->squareTexture

)

);

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/barrel.png",

0,

&this->barrelTexture

)

);

//Set the initial position of the sprite

barrelPosition.x = this->windowBounds.Width - this->barrelFrameWidth;

barrelPosition.y = this->windowBounds.Height - this->barrelFrameHeight;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/barrel.png",

0,

&this->flippedTexture

)

);

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/cabinets.png",

0,

&this->partialTexture

)

);

this->partialSourceRect = std::unique\_ptr<RECT>(new RECT());

this->partialSourceRect->left = 32;

this->partialSourceRect->top = 5;

this->partialSourceRect->right = 64;

this->partialSourceRect->bottom = 74;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/cabinets.png",

0,

&this->cupboardTexture

)

);

this->cupboardSourceRect = std::unique\_ptr<RECT>(new RECT());

this->cupboardSourceRect->left = 32;

this->cupboardSourceRect->top = 101;

this->cupboardSourceRect->right = 64;

this->cupboardSourceRect->bottom = 170;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/cabinets.png",

0,

&this->trophyTexture

)

);

this->trophySourceRect = std::unique\_ptr<RECT>(new RECT());

this->trophySourceRect->left = 96;

this->trophySourceRect->top = 5;

this->trophySourceRect->right = 128;

this->trophySourceRect->bottom = 74;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/cabinets.png",

0,

&this->glassTexture

)

);

this->glassSourceRect = std::unique\_ptr<RECT>(new RECT());

this->glassSourceRect->left = 32;

this->glassSourceRect->top = 197;

this->glassSourceRect->right = 64;

this->glassSourceRect->bottom = 264;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/SpriteAnimation.png",

0,

&this->reverseTexture

)

);

this->reverseSourceRect = std::unique\_ptr<RECT>(new RECT());

this->reverseSourceRect->left = 0;

this->reverseSourceRect->top = 0;

this->reverseSourceRect->right = 175;

this->reverseSourceRect->bottom = 200;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/Disc.png",

0,

&this->discTexture

)

);

discPosition.x = 0.0f;

discPosition.y = 500.0f;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/rock.png",

0,

&this->plasmaTexture

)

);

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/big\_worm.png",

0,

&this->wormTexture

)

);

this->wormSourceRect = std::unique\_ptr<RECT>(new RECT);

this->wormSourceRect->left = this->wormFrameWidth \* this->wormColumn;

this->wormSourceRect->top = this->wormFrameHeight \* this-> wormRow;

this->wormSourceRect->right = this->wormFrameWidth \* (this->wormColumn + 1);

this->wormSourceRect->bottom = this->wormFrameHeight \* (this-> wormRow + 1);

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/GnomeIntoTroll.png",

0,

&this->creatureGnomeTexture

)

);

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/TrollIntoGnome.png",

0,

&this->creautureTrollTexture

)

);

this->creatureCurrentTexture = this->creatureGnomeTexture;

this->creatureSourceRect = std::unique\_ptr<RECT>(new RECT);

this->creatureSourceRect->left = this->creatureFrameWidth \* this->creatureFrame;

this->creatureSourceRect->top = 0;

this->creatureSourceRect->right = this->creatureFrameWidth \* (this->creatureFrame + 1);

this->creatureSourceRect->bottom = this->creatureFrameHeight;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/Disc.png",

0,

&this->movingDiscTexture

)

);

movingDiscPosition.x = windowBounds.Width/2 -16.0f;

movingDiscPosition.y = windowBounds.Height/2 -16.0f;

ThrowIfFailed(

CreateWICTextureFromFile(

this->d3dDevice.Get(),

this->d3dDeviceContext.Get(),

L"Assets/TrollIdleStance.png",

0,

&this->trollTexture

)

);

this->trollSourceRect = std::unique\_ptr<RECT>(new RECT);

this->trollSourceRect->left = this->trollFrameWidth \* 0;

this->trollSourceRect->top = 0;

this->trollSourceRect->right = this->trollFrameWidth \* 1;

this->trollSourceRect->bottom = this->trollFrameHeight;

}

void SpriteGame::Update(int tickTotal, int tickDelta, float secondsTotal, float secondsDelta)

{

// Updating the grass animation

if ( tickTotal >= this->grassLastFrameChange + 12)

{

//Two ticks have elapsed since the last frame change

this->grassFrame = this-> grassFrame + this->grassIncrement;

this->grassLastFrameChange = tickTotal;

if (this->grassFrame >= this->grassSheetLength)

{

this->grassFrame = 0;

}

//Frame has changed so change the source rectangle

this->grassSourceRect->left = this->grassFrame \* 175;

this->grassSourceRect->top = 0;

this->grassSourceRect->right = (this->grassFrame + this->grassIncrement) \* 175;

this->grassSourceRect->bottom = 200;

}

// Deal with almighty square motion

float distanceToMove = tickDelta \* this->squareVelocity;

this->squarePosition.x = this->squarePosition.x + distanceToMove;

// Updating the grass animation with added reverse

if ( tickTotal >= this->reverseLastFrameChange + 12)

{

//Two ticks have elapsed since the last frame change

this->reverseFrame = this->reverseFrame + this->reverseSubtraction;

this->reverseLastFrameChange = tickTotal;

if(this->reverseFrame >= this->reverseSheetLength - 1)

{

this->reverseSubtraction = this->reverseSubtraction \* -1;

}

else if (this->reverseFrame <= 0 )

{

this->reverseSubtraction = this->reverseSubtraction \* -1;

}

this->reverseSourceRect->left = this->reverseFrame \* 175;

this->reverseSourceRect->top = 0;

this->reverseSourceRect->right = (this->reverseFrame + 1) \* 175;

this->reverseSourceRect->bottom = 200;

}

// Dealing with motion

Vector2 discTranslation = tickDelta \* this->discVelocity;

this->discPosition = this->discPosition + discTranslation;

// The disc has reached the bottom border of the square

if( this->discPosition.y >= this->bottomPosition.y)

{

this->discVelocity.y = -5.0f;

}

// The disc has reached the right border of the square

if (this->discPosition.x >= this->rightPosition.x)

{

this->discVelocity.x = -2.0f;

}

// The disc has reached the left border of the square

if (this->discPosition.x <= this->leftPosition.x)

{

this->discVelocity.x = 2.0f;

}

// The disc has reached the top border of the square

if (this->discPosition.y <= topPosition.y)

{

this->discVelocity.y = 5.0f;

}

// SimpleMath::Color addColor = SimpleMath::Color (0.0f, -0.0015f, -0.01f);

// Dealing with changing rock colour

float newBluelValue = this->plasmaColor.B() + this->addBlue.B();

this->plasmaColor.B(newBluelValue);

float newGreenValue = this->plasmaColor.G() + this->addGreen.G();

this->plasmaColor.G(newGreenValue);

if (this->plasmaColor.G() <= this->rockColor.G())

{

this->addBlue = this->addBlue \* -1;

this->addGreen = this->addGreen \* -1;

}

if (this->plasmaColor.G() >= this->wormColor.G())

{

this->addBlue = this->addBlue \* -1;

this->addGreen = this->addGreen \* -1;

}

// Dealing with worm

//Local variable to track frame changes

bool frameChanged = false;

//This is where you manage the state of game objects

if ( tickTotal >= this->wormLastFrameChange + this->wormFrameRate)

{

//Time to change frame

this->wormColumn = this->wormColumn + 1;

frameChanged = true;

if (this->wormColumn >= this->wormSheetWidth)

{

this->wormColumn= 0;

}

//Now we are using the full sprite sheet we need to set all the values for

// the source rectangle

this->wormSourceRect->left = this->wormFrameWidth \* this->wormColumn;

this->wormSourceRect->top = this->wormFrameHeight \* this-> wormRow;

this->wormSourceRect->right = this->wormFrameWidth \* (this->wormColumn + 1);

this->wormSourceRect->bottom = this->wormFrameHeight \* (this-> wormRow + 1);

}

//Move from showing row 0 to showing row 1 after five seconds, i.e. 120 ticks

if ( ( tickTotal == 120 && tickTotal > this->wormLastFrameChange )

|| ( tickTotal == 240 && tickTotal > this->wormLastFrameChange )

|| ( tickTotal == 360 && tickTotal > this->wormLastFrameChange))

{

//Time to change to drawing the damaged ship

this->wormRow = this->wormRow + 1;

//For tidyness, start from the first frame on this row

this->wormColumn = 0;

//As we are changing row, we are also changing frame

frameChanged = true;

}

if ( frameChanged )

{

this->wormLastFrameChange = tickTotal;

}

//Dealing with gnome/troll transition

// Local variable to track frame changes

bool creatureChanged = false;

// This is where you manage the state of game objects

if ( tickTotal >= this->creatureLastFrameChange + this->creatureFrameRate)

{

//Time to change frame

this->creatureFrame = this->creatureFrame + 1;

creatureChanged = true;

if (this->creatureFrame >= this->creatureSheetWidth && this->creatureCurrentTexture == this->creatureGnomeTexture)

{

this->creatureCurrentTexture = this->creautureTrollTexture;

this->creatureFrame = 0;

}

else if (this->creatureFrame >= this->creatureSheetWidth && this->creatureCurrentTexture == this->creautureTrollTexture)

{

this->creatureCurrentTexture = this->creatureGnomeTexture;

this->creatureFrame = 0;

}

this->creatureSourceRect->left = this->creatureFrameWidth \* this->creatureFrame;

this->creatureSourceRect->right = this->creatureFrameWidth \* (this->creatureFrame + 1);

this->creatureSourceRect->bottom = this->creatureFrameHeight;

this->creatureSourceRect->top = 0;

}

if ( creatureChanged )

{

this->creatureLastFrameChange = tickTotal;

}

//For tracking number of ticks

this->trackTicks = std::to\_wstring( tickTotal );

this->trackTrigger = std::to\_wstring (gamepad.GetTrigger());

//For updating the position of the troll

int spawnWindowWidth = this->windowBounds.Width;

int spawnWindowHeight = this->windowBounds.Height;

srand(time(0));

bool checker = false;

int randomTime;

randomTime = rand() % 400 + 20;

if (checker == false)

{

interval++;

if( interval >= randomTime)

{

this->trollPosition.x = rand() % spawnWindowWidth;

this->trollPosition.y = rand() % spawnWindowHeight;

interval = 0;

}

}

//For changing the background colour redder via gamepad

float colorAway = 0.001f;

GameController oldGamepad = this->gamepad; //Make a local copy of the old gamepad state

this->gamepad.Update(); //Get the current gamepad state

if(this->gamepad.IsConnected())

{

// Level Detection

if (this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_B) && !this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_LEFT\_SHOULDER))

{

float subtractGBackground = this->clearToThisColor.G() - colorAway;

this->clearToThisColor.G(subtractGBackground);

float subtractBBackground = this->clearToThisColor.B() - colorAway;

this->clearToThisColor.B(subtractBBackground);

}

// Edge Detection

if (!this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_RIGHT\_SHOULDER) && oldGamepad.IsButtonDown(XINPUT\_GAMEPAD\_RIGHT\_SHOULDER))

{

this->clearToThisColor = DirectX::Colors::White.v;

}

//Multiple Buttons

if (this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_LEFT\_SHOULDER) && this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_B))

{

float addGBackground = this->clearToThisColor.G() + colorAway;

this->clearToThisColor.G(addGBackground);

float addBBackground = this->clearToThisColor.B() + colorAway;

this->clearToThisColor.B(addBBackground);

}

//Calculate displacement using leftThumbstick vector as velocity during current time period

Vector2 leftThumbstick = gamepad.GetThumbstick(XINPUT\_GAMEPAD\_LEFT\_THUMB);

this->movingDiscPosition = this->movingDiscPosition + (leftThumbstick \* this->movingDiscSpeed \* secondsDelta);

//Stop paddle moving off screen

if ( this->movingDiscPosition.y >= windowBounds.Height - this->movingDiscboundingBox.Height )

{

this->movingDiscPosition.y = windowBounds.Height - this->movingDiscboundingBox.Height;

}

if ( this->movingDiscPosition.y <= 0.0f )

{

this->movingDiscPosition.y = 0.0f;

}

if ( this->movingDiscPosition.x >= windowBounds.Width - this->movingDiscboundingBox.Width)

{

this->movingDiscPosition.x = windowBounds.Width - this->movingDiscboundingBox.Width;

}

if ( this->movingDiscPosition.x <= 0.0f )

{

this->movingDiscPosition.x = 0.0f;

}

//Calculate the colour change if the left trigger is pressed

//Left Trigger

float addColour = 0.1f;

float stopAdd = 1.0f;

float stopColour = 0.0f;

if (this->gamepad.IsTriggerDown(TRUE) & !this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_B))

{

float removeGreen = this->clearToThisColor.G() - addColour;

this->clearToThisColor.G(removeGreen);

float removeRed = this->clearToThisColor.R() - addColour;

this->clearToThisColor.R(removeRed);

if (this->clearToThisColor.R() <= this->addBlue.R() & !this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_B))

{

this->clearToThisColor.G(stopColour);

this->clearToThisColor.R(stopColour);

}

}

else if (!this->gamepad.IsTriggerDown(TRUE) && !this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_B) && this->clearToThisColor.B() >= changeColor.B())

{

float removeGreen = this->clearToThisColor.G() + addColour;

this->clearToThisColor.G(removeGreen);

if (this->clearToThisColor.G() >= this->wormColor.G())

{

this->clearToThisColor.G(stopAdd);

}

float removeRed = this->clearToThisColor.R() + addColour;

this->clearToThisColor.R(removeRed);

if (this->clearToThisColor.R() >= this->wormColor.R())

{

this->clearToThisColor.R(stopAdd);

}

}

//Adjust the bounding box

this->movingDiscboundingBox.X = this->movingDiscPosition.x;

this->movingDiscboundingBox.Y = this->movingDiscPosition.y;

if(this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_DPAD\_LEFT))

{

this->gamepad.Vibrate(65535, 0);

}

if(this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_DPAD\_RIGHT))

{

this->gamepad.Vibrate(0, 65535);

}

if(this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_DPAD\_DOWN))

{

this->gamepad.Vibrate(65535, 65535);

}

if(this->gamepad.IsButtonDown(XINPUT\_GAMEPAD\_DPAD\_UP))

{

this->gamepad.Vibrate();

}

}

}

void SpriteGame::Render()

{

//First, clear the back buffer of everything you drew to it last time

this->d3dDeviceContext->ClearRenderTargetView(

this->backBufferTarget.Get(),

clearToThisColor

);

//Next, set the back buffer as our current target for rendering to

this->d3dDeviceContext->OMSetRenderTargets(

1,

this->backBufferTarget.GetAddressOf(),

nullptr

);

//Finally, render your sprites to the backbuffer

this->spriteBatch->Begin(SpriteSortMode\_Deferred, commonStates->NonPremultiplied());

//Draw the rock sprite

this->spriteBatch->Draw(this->rockTexture.Get(), rockPosition, this->rockSourceRect.get(), rockColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects\_None, 0.0f);

//Draw the cabinet sprite

this->spriteBatch->Draw(this->cabinetTexture.Get(), cabinetPosition, this->cabinetSourceRect.get(), cabinetColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the grass animation sprite

this->spriteBatch->Draw(this->grassTexture.Get(), grassPosition, this->grassSourceRect.get(), grassColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the almighty square

this->spriteBatch->Draw(this->squareTexture.Get(), squarePosition, this->squareSourceRect.get(), squareColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the set of barrels

this->spriteBatch->Draw(this->barrelTexture.Get(), barrelPosition, this->barrelSourceRect.get(), barrelColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the set of flipped barrels

this->spriteBatch->Draw(this->flippedTexture.Get(), flippedPosition, this->flippedSourceRect.get(), flippedColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_FlipHorizontally, 0.0f);

//Draw the set of partial cabinets

this->spriteBatch->Draw(this->partialTexture.Get(), partialPosition, this->partialSourceRect.get(), partialColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the piece of trophy

this->spriteBatch->Draw(this->trophyTexture.Get(), trophyPosition, this->trophySourceRect.get(), trophyColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the piece of cupboard

this->spriteBatch->Draw(this->cupboardTexture.Get(), cupboardPosition, this->cupboardSourceRect.get(), cupboardColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the piece of glass

this->spriteBatch->Draw(this->glassTexture.Get(), glassPosition, this->glassSourceRect.get(), glassColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the reversed animation

this->spriteBatch->Draw(this->reverseTexture.Get(), reversePosition, this->reverseSourceRect.get(), reverseColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 0.5f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the moving disc

this->spriteBatch->Draw(this->discTexture.Get(), discPosition, this->discSourceRect.get(), discColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the plasma rock sprite

this->spriteBatch->Draw(this->plasmaTexture.Get(), plasmaPosition, this->plasmaSourceRect.get(), plasmaColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the worm

this->spriteBatch->Draw(this->wormTexture.Get(), wormPosition, this->wormSourceRect.get(), wormColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the gnome/troll

this->spriteBatch->Draw(this->creatureCurrentTexture.Get(), creaturePosition, this->creatureSourceRect.get(), creatureColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Draw the text "Hello game world"

this->spriteFont->DrawString(this->spriteBatch.get(), (L"Hello Game World" + this->trackTrigger).c\_str(), Vector2(500, 10), DirectX::Colors::Yellow);

//Draw the text "Tick Total =" followed by displaying the number of ticks gone by in the game

this->eleFont->DrawString(this->spriteBatch.get(), (L"Tick Total =" + this->trackTicks).c\_str(), Vector2(500, 100), DirectX::Colors::Black);

//Draw the controlled Disc object

this->spriteBatch->Draw(this->movingDiscTexture.Get(), this->movingDiscPosition, this->movingDiscColor);

//Draw the random trolls

this->spriteBatch->Draw(this->trollTexture.Get(), trollPosition, this->trollSourceRect.get(), trollColor, 0.0f, SimpleMath::Vector2(0.0f, 0.0f), 1.0f, SpriteEffects::SpriteEffects\_None, 0.0f);

//Further calls to this->spriteBatch->Draw() to draw any more sprites

//When all sprites are drawn ...

this->spriteBatch->End();

}

void SpriteGame::Present()

{

DXGI\_PRESENT\_PARAMETERS parameters = {0};

parameters.DirtyRectsCount = 0;

parameters.pDirtyRects = nullptr;

parameters.pScrollRect = nullptr;

parameters.pScrollOffset = nullptr;

HRESULT hr = this->swapChain->Present1(1, 0, &parameters);

this->d3dDeviceContext->DiscardView(this->backBufferTarget.Get());

ThrowIfFailed(hr);

}

float SpriteGame::ConvertDipsToPixels(float dips)

{

static const float dipsPerLogicalInch = 96.0f;

return floor(dips \* DisplayProperties::LogicalDpi / dipsPerLogicalInch + 0.5f); // Round to nearest integer.

}