/\*

=================

main.cpp

Main entry point for the application

=================

\*/

#include "GameConstants.h"

#include "GameResources.h" //define font and window icon

#include "cD3DManager.h"

#include "cD3DXSpriteMgr.h"

#include "cD3DXTexture.h"

#include "cSprite.h" //these all handle the physical appearance of the game

#include "cExplosion.h" //handles the explosion graphic

#include "cStephen.h" //handles the player character

#include "cEnemy.h" //handles the enemy characters

#include "cXAudio.h" //handles the game audio

#include "cD3DXFont.h" //handles the game font

#include <strsafe.h>

using namespace std; //standard namespace

bool game = false; //are we in the gameplay mode?

bool gameWater = false; //are we in the water level?

bool gameLightning = false; //are we in the lightning level?

bool gameFire = false; //are we in the fire level?

bool gameSpace = false; //are we in the space level?

bool gameCity = false; //are we in the city level?

bool gameForest = false; //are we in the forest level?

bool gameIce = false; //are we in the ice level?

bool startmenu = true; //are we in the start menu?

bool endscreen = false; //are we in the end screen?

bool music = false; //is the music on?

RECT start;

HINSTANCE hInst; // global handle to hold the application instance

HWND wndHandle; // global variable to hold the window handle

// Get a reference to the DirectX Manager

static cD3DManager\* d3dMgr = cD3DManager::getInstance();

// Get a reference to the DirectX Sprite renderer Manager

static cD3DXSpriteMgr\* d3dxSRMgr = cD3DXSpriteMgr::getInstance();

float rot = 0.0f; //Stephen's rotation value

D3DXVECTOR2 stephenTrans = D3DXVECTOR2(100,375); //Stephen's position on spawn

D3DXVECTOR2 stephenScale = D3DXVECTOR2(1.0f,1.0f); //Stephen's size

D3DXVECTOR2 EnemyTrans = D3DXVECTOR2(300,300); //The enemy position

RECT clientBounds; //the boundaries of the window aka game space

int EnemyPos = clientBounds.right/2 - 200; //the enemies will be positioned here, slightly off centre

vector<cEnemy\*> aEnemy; //holds all enemies

vector<cEnemy\*>::const\_iterator iterRO;

vector<cEnemy\*>::iterator george; //iterators to point to specific enemies

vector<cEnemy\*>::iterator index;

TCHAR szTempOutput[30];

bool gHit = false; //have we hit an enemy?

int gEnemiesKilled = 0; //number of dead enemies

char gEnemiesKilledStr[50]; //number to display

D3DXVECTOR3 expPos; //position of the explosion

POINT ptStephen; //the exact point stephen makes contact

list<cExplosion\*> gExplode; //contains the explosion

list<cStephen\*> gStephen; //contains Stephen

POINT pstephenPos; //the position of the contact point

list<cStephen\*>::iterator pete = gStephen.begin(); //point iterator Pete at Stephen

cXAudio gAttackSound; //the sound effect for attacking

cXAudio gameMusic; //the background music

LPCSTR background = "Images\\menu background.png"; //the background image for the main menu

LPDIRECT3DSURFACE9 aSurface;

/\*

==================================================================

\* LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,

\* LPARAM lParam)

\* The window procedure

==================================================================

\*/

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

// Check any available messages from the queue

switch (message)

{

//if the user is pressing a key

case WM\_KEYDOWN:

{

//if the user is pressing the left arrow

if(wParam == VK\_LEFT)

{

//if stephen isn't touching the left side of the window

//move him left by ten units

if(pstephenPos.x>(clientBounds.left))

{

pstephenPos.x -=10.0f;

stephenTrans.x -= 10.0f;

return 0;

}

}

//if the user is pressing the right arrow

if(wParam == VK\_RIGHT)

{

//if stephen isn't touching the right side of the window minus 90 units

//move him right by ten units

if(pstephenPos.x<(clientBounds.right - 90))

{

pstephenPos.x += 10.0f;

stephenTrans.x += 10.0f;

return 0;

}

}

//if the user is pressing 1

if(wParam == '1')

{

//turn all other levels off except for the water level

startmenu = false;

gameLightning = false;

gameFire = false;

gameSpace = false;

gameCity = false;

gameForest = false;

gameIce = false;

game = true;

gameWater = true;

return 0;

}

//if the user is pressing 2

if(wParam == '2')

{

//turn all other levels off except for the lightning level

startmenu = false;

gameWater = false;

gameFire = false;

gameSpace = false;

gameCity = false;

gameForest = false;

gameIce = false;

game = true;

gameLightning = true;

return 0;

}

//if the user is pressing 3

if(wParam == '3')

{

//turn off all levels apart from the fire level

startmenu = false;

gameWater = false;

gameLightning = false;

gameSpace = false;

gameCity = false;

gameForest = false;

gameIce = false;

game = true;

gameFire = true;

return 0;

}

//if the user is pressing 4

if(wParam == '4')

{

//turn off all levels apart from the space level

startmenu = false;

gameWater = false;

gameLightning = false;

gameFire = false;

gameCity = false;

gameForest = false;

gameIce = false;

game = true;

gameSpace = true;

return 0;

}

//if the user is pressing 5

if(wParam == '5')

{

//turn off all levels apart from the city level

startmenu = false;

gameWater = false;

gameLightning = false;

gameFire = false;

gameSpace = false;

gameForest = false;

gameIce = false;

game = true;

gameCity = true;

return 0;

}

//if the user is pressing 6

if(wParam == '6')

{

//turn off all levels apart from the forest level

startmenu = false;

gameWater = false;

gameLightning = false;

gameFire = false;

gameSpace = false;

gameCity = false;

gameIce = false;

game = true;

gameForest = true;

return 0;

}

//if the user is pressing 7

if(wParam == '7')

{

//turn off all levels apart from the ice level

startmenu = false;

gameWater = false;

gameLightning = false;

gameFire = false;

gameSpace = false;

gameCity = false;

gameForest = false;

game = true;

gameIce = true;

return 0;

}

//if the user is pressing 8

if(wParam=='8')

{

//turn off all levels and the gameplay and take the user to the end screen

startmenu = false;

gameWater = false;

gameLightning = false;

gameFire = false;

gameSpace = false;

gameCity = false;

gameForest = false;

gameIce == false;

game = false;

endscreen = true;

return 0;

}

//if the user is pressing 9

if(wParam == '9')

{

//turn off all levels and gameplay and take them to the start menu

gameWater = false;

gameLightning = false;

gameFire = false;

gameSpace = false;

gameCity = false;

gameForest = false;

gameIce = false;

game = false;

endscreen == false;

startmenu = true;

return 0;

}

//if the user is pressing the space bar

if(wParam == VK\_SPACE)

{

//if we are in gameplay, i.e. one of the levels

if(game == true)

{

//point the iterator George at the beginning of the enemy vector

POINT mouseXY;

mouseXY.x = LOWORD(lParam);

mouseXY.y = HIWORD(lParam);

george = aEnemy.begin();

//while george is in the enemy vector and there isn't a hit

while (george != aEnemy.end() && !gHit)

{

//if an enemy's bounding rectangle is within Stephen's bounding rectangle

if ( (\*george)->insideRect((\*george)->getBoundingRect(), pstephenPos))

{

//a hit has occurred and that enemy will be eliminated

//add one to the kill counter and display this

OutputDebugString("Hit!\n");

gHit = true;

george = aEnemy.erase(george);

gEnemiesKilled++;

sprintf\_s(gEnemiesKilledStr, 50, "Kills: %d", gEnemiesKilled);

}

else

{

//if not then move on to the next enemy and check again

++george;

}

}

//play the attack sound effect whether or not there's a hit

expPos = D3DXVECTOR3((float)pstephenPos.x,(float)pstephenPos.y-23, 0.0f);

gAttackSound.playSound(L"Sounds\\dale.wav",false);

StringCchPrintf(szTempOutput, STRSAFE\_MAX\_CCH, TEXT("Mouse: lLastX=%d lLastY=%d\r\n"), LOWORD(lParam), HIWORD(lParam));

OutputDebugString(szTempOutput);

gHit = false;

return 0;

}

}

return 0;

}

case WM\_CLOSE:

{

// Exit the Game

PostQuitMessage(0);

return 0;

}

case WM\_DESTROY:

{

PostQuitMessage(0);

return 0;

}

}

// Always return the message to the default window

// procedure for further processing

return DefWindowProc(hWnd, message, wParam, lParam);

}

/\*

==================================================================

\* bool initWindow( HINSTANCE hInstance )

\* initWindow registers the window class for the application, creates the window

==================================================================

\*/

bool initWindow( HINSTANCE hInstance )

{

WNDCLASSEX wcex;

// Fill in the WNDCLASSEX structure. This describes how the window

// will look to the system

wcex.cbSize = sizeof(WNDCLASSEX); // the size of the structure

wcex.style = CS\_HREDRAW | CS\_VREDRAW; // the class style

wcex.lpfnWndProc = (WNDPROC)WndProc; // the window procedure callback

wcex.cbClsExtra = 0; // extra bytes to allocate for this class

wcex.cbWndExtra = 0; // extra bytes to allocate for this instance

wcex.hInstance = hInstance; // handle to the application instance

wcex.hIcon = LoadIcon(hInstance,MAKEINTRESOURCE(IDI\_MyWindowIcon)); // icon to associate with the application

wcex.hCursor = LoadCursor(NULL, (IDC\_ARROW));// the default cursor

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1); // the background color

wcex.lpszMenuName = NULL; // the resource name for the menu

wcex.lpszClassName = "Balloons"; // the class name being created

wcex.hIconSm = LoadIcon(hInstance,"stephen icon.ico"); // the handle to the small icon

pstephenPos.x = 100;

pstephenPos.y = 375;

RegisterClassEx(&wcex);

// Create the window

wndHandle = CreateWindow("Balloons", // the window class to use

"Stephen Stills' Fabulous Adventure", // the title bar text

WS\_OVERLAPPEDWINDOW, // the window style

CW\_USEDEFAULT, // the starting x coordinate

CW\_USEDEFAULT, // the starting y coordinate

800, // the pixel width of the window

600, // the pixel height of the window

NULL, // the parent window; NULL for desktop

NULL, // the menu for the application; NULL for none

hInstance, // the handle to the application instance

NULL); // no values passed to the window

// Make sure that the window handle that is created is valid

if (!wndHandle)

return false;

// Display the window on the screen

ShowWindow(wndHandle, SW\_SHOW);

UpdateWindow(wndHandle);

return true;

}

/\*

==================================================================

// This is winmain, the main entry point for Windows applications

==================================================================

\*/

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow )

{

// Initialize the window

if ( !initWindow( hInstance ) )

return false;

// called after creating the window

if ( !d3dMgr->initD3DManager(wndHandle) )

return false;

if ( !d3dxSRMgr->initD3DXSpriteMgr(d3dMgr->getTheD3DDevice()))

return false;

// Grab the frequency of the high def timer

\_\_int64 freq = 0; // measured in counts per second;

QueryPerformanceFrequency((LARGE\_INTEGER\*)&freq);

float sPC = 1.0f / (float)freq; // number of seconds per count

\_\_int64 currentTime = 0; // current time measured in counts per second;

\_\_int64 previousTime = 0; // previous time measured in counts per second;

float numFrames = 0.0f; // Used to hold the number of frames

float timeElapsed = 0.0f; // cumulative elapsed time

GetClientRect(wndHandle,&clientBounds);

float fpsRate = 1.0f/25.0f; //the frame rate the game will run at

D3DXVECTOR3 EnemyPos; //the position of the enemy

srand((unsigned int)time(NULL)); //seed the random number generator

int numEnemies = 5; //there will be five enemies on spawn

//a loop that stops when we run out of enemies

for(int loop = 0; loop < numEnemies; loop++)

{

//if it's one of the first three enemies

if(loop<3){

//draw the enemy at these calculated co-ordinates

EnemyPos = D3DXVECTOR3(160+loop\*160,(float)clientBounds.top+150,0);

}

else{

//draw enemies after the first three under the first three

EnemyPos = D3DXVECTOR3(160+((loop-3)\*160),(float)clientBounds.top+300,0);

}

//draw the enemy sprite

//seed another number generator

//make all the enemies fly in this pattern

aEnemy.push\_back(new cEnemy(EnemyPos,d3dMgr->getTheD3DDevice(),"Images\\ridley.png"));

srand((unsigned int)time(NULL));

aEnemy[loop]->setTranslation(D3DXVECTOR2(5.0f,(rand()%+(-5))));

}

LPDIRECT3DSURFACE9 aSurface; // the Direct3D surface

LPDIRECT3DSURFACE9 theBackbuffer = NULL; // This will hold the back buffer

cD3DXFont \* EnemyFont = new cD3DXFont(d3dMgr->getTheD3DDevice(),hInstance, "Metroid Prime: Hunters3"); //the font for the kill counter

cD3DXFont \* MenuFont = new cD3DXFont(d3dMgr->getTheD3DDevice(),hInstance, "Metroid Prime: Hunters3"); //the font for the miscellaneous messages

RECT textPos; //the position of the kill counter

SetRect(&textPos, 30, 10, 400, 100);

RECT menuTextPos; // the position of the messages

SetRect(&menuTextPos, 20, 10, 600, 1000);

D3DXVECTOR3 stephenPos = D3DXVECTOR3(600, 600, 0); //the position of Stephen

D3DXVECTOR3 aEnemyPos; //the position of the enemies

cSprite stephen (stephenPos, d3dMgr->getTheD3DDevice(), "Images\\stephen.png"); //image for Stephen's sprite

stephen.setTranslation(D3DXVECTOR2(5.0f, 0.0f));

D3DXMATRIX stephenMatrix; //stephen's transform matrix

MSG msg;

ZeroMemory( &msg, sizeof( msg ) );

// Create the background surface

aSurface = d3dMgr->getD3DSurfaceFromFile(background);

gameMusic.playSound(L"Sounds\\gabbag.wav",true); //the game's music

QueryPerformanceCounter((LARGE\_INTEGER\*)&currentTime);

//while we're still playing

while( msg.message!=WM\_QUIT )

{

// Check the message queue

if (PeekMessage(&msg, NULL, 0U, 0U, PM\_REMOVE) )

{

TranslateMessage( &msg );

DispatchMessage( &msg );

}

else

{

// Game code goes here

//if the start menu's open

if(startmenu == true && music == false)

{

//draw the background and render the text

background = "Images\\menu background.png";

aSurface = d3dMgr->getD3DSurfaceFromFile(background);

d3dMgr->beginRender();

theBackbuffer = d3dMgr->getTheBackBuffer();

d3dMgr->updateTheSurface(aSurface, theBackbuffer);

d3dMgr->releaseTheBackbuffer(theBackbuffer);

MenuFont->printText("Stephen Stills' Fabulous Adventure\n\n\n\nPress any number from 1-7 to begin! \nUse LEFT and RIGHT to move! \nSwap backgrounds! There are seven! \nPress SPACE to kill Ridleys!", menuTextPos);

d3dMgr->endRender();

}

//if the endscreen's open

if(endscreen == true)

{

//the gameplay itself isn't happening

//draw the background and render the text

//respawn another five enemies which won't be seen until we go back into a level using the same process as before

game = false;

background = "Images\\menu background.png";

aSurface = d3dMgr->getD3DSurfaceFromFile(background);

for(int loop = 0; loop < numEnemies; loop++)

{

if(loop<3){

EnemyPos = D3DXVECTOR3(160+loop\*160,(float)clientBounds.top+150,0);

}

else{

EnemyPos = D3DXVECTOR3(160+((loop-3)\*160),(float)clientBounds.top+300,0);

}

aEnemy.push\_back(new cEnemy(EnemyPos,d3dMgr->getTheD3DDevice(),"Images\\ridley.png"));

srand((unsigned int)time(NULL));

aEnemy[loop]->setTranslation(D3DXVECTOR2(5.0f,(rand()%+(-5))));

}

d3dMgr->beginRender();

theBackbuffer = d3dMgr->getTheBackBuffer();

d3dMgr->updateTheSurface(aSurface, theBackbuffer);

d3dMgr->releaseTheBackbuffer(theBackbuffer);

EnemyFont->printText(gEnemiesKilledStr,textPos);

MenuFont->printText("\nThank you for playing! \nGo back to the levels! \nThey're waiting for you...", menuTextPos);

d3dMgr->endRender();

endscreen = false;

music = false;

}

//if we're playing the actual game

if(game == true)

{

//work out the positions of Stephen and the enemies along with displaying how many kills we have

D3DXMatrixTransformation2D(&stephenMatrix, NULL,0.0f,&stephenScale,&(stephen.getSpriteCentre()),rot,&stephenTrans);

QueryPerformanceCounter((LARGE\_INTEGER\*)&currentTime);

float dt = (currentTime - previousTime)\*sPC;

EnemyPos = D3DXVECTOR3(EnemyTrans.x,EnemyTrans.y,0);

sprintf\_s(gEnemiesKilledStr, 50, "Kills: %d", gEnemiesKilled);

// Accumulate how much time has passed.

timeElapsed += dt;

/\*

==============================================================

| Update the postion of the enemies and check for collisions

==============================================================

\*/

if(timeElapsed > fpsRate)

{

//if we're in the water level draw the water background

if(gameWater == true)

{

aSurface = d3dMgr->getD3DSurfaceFromFile("Images\\water background.png");

}

//if we're in the lightning level draw the lightning background

if(gameLightning == true)

{

aSurface = d3dMgr->getD3DSurfaceFromFile("Images\\lightning background.png");

}

//if we're in the fire level draw the fire background

if(gameFire == true)

{

aSurface = d3dMgr->getD3DSurfaceFromFile("Images\\fire background.png");

}

//if we're in the space level draw the space background

if(gameSpace == true)

{

aSurface = d3dMgr->getD3DSurfaceFromFile("Images\\space background.png");

}

//if we're in the city level draw the city background

if(gameCity == true)

{

aSurface = d3dMgr->getD3DSurfaceFromFile("Images\\city background.png");

}

//if we're in the forest level draw the forest background

if(gameForest == true)

{

aSurface = d3dMgr->getD3DSurfaceFromFile("Images\\forest background.png");

}

//if we're in the ice level draw the ice background

if(gameIce == true)

{

aSurface = d3dMgr->getD3DSurfaceFromFile("Images\\ice background.png");

}

//if we're in the end screen draw that background

if(endscreen == true)

{

aSurface = d3dMgr->getD3DSurfaceFromFile("Images\\menu background.png");

}

for(george = aEnemy.begin(); george != aEnemy.end(); ++george)

{

(\*george)->update(); // update enemy

aEnemyPos = (\*george)->getSpritePos(); // get the position of the current enemy

/\*

==============================================================

| Check for boundary collision and change enemy direction

==============================================================

\*/

if((\*george)->getTranslation().x>0 && aEnemyPos.x>(clientBounds.right - 60))

{

(\*george)->setTranslation(D3DXVECTOR2(-((\*george)->getTranslation().x), (\*george)->getTranslation().y));

}

if((\*george)->getTranslation().x<0 && aEnemyPos.x<(clientBounds.left + 10))

{

(\*george)->setTranslation(D3DXVECTOR2(-((\*george)->getTranslation().x), (\*george)->getTranslation().y));

}

if((\*george)->getTranslation().y<0 && aEnemyPos.y<(clientBounds.top + 50))

{

(\*george)->setTranslation(D3DXVECTOR2(-((\*george)->getTranslation().x), -((\*george)->getTranslation().y)));

}

if((\*george)->getTranslation().y>0 && aEnemyPos.y>(clientBounds.bottom-200))

{

(\*george)->setTranslation(D3DXVECTOR2(-((\*george)->getTranslation().x), -((\*george)->getTranslation().y)));

}

for(index = aEnemy.begin(); index != aEnemy.end(); ++index)

{

if ((\*george)->collidedWith((\*george)->getBoundingRect(),(\*index)->getBoundingRect()))

{

// if a collision occurs change the direction of each enemy that has collided

OutputDebugString("Collision!!");

(\*george)->setTranslation((\*george)->getTranslation()\*(-1));

(\*index)->setTranslation((\*index)->getTranslation()\*(-1));

}

}

}

//the rest of the loop is dedicated to drawing everything

//the enemies, followed by Stephen

//then depending on the level draw text that explains what level you're in and to press 8 to exit to the main screen

d3dMgr->beginRender();

theBackbuffer = d3dMgr->getTheBackBuffer();

d3dMgr->updateTheSurface(aSurface, theBackbuffer);

d3dMgr->releaseTheBackbuffer(theBackbuffer);

d3dxSRMgr->beginDraw();

for(george = aEnemy.begin(); george!= aEnemy.end(); ++george)

{

d3dxSRMgr->setTheTransform((\*george)->getSpriteTransformMatrix());

d3dxSRMgr->drawSprite((\*george)->getTexture(),NULL,NULL,NULL,0xFFFFFFFF);

}

d3dxSRMgr->endDraw();

d3dxSRMgr->beginDraw();

d3dxSRMgr->setTheTransform(stephenMatrix);

d3dxSRMgr->drawSprite(stephen.getTexture(),NULL,NULL,NULL,0xFFFFFFFF);

d3dxSRMgr->endDraw();

//d3dMgr->endRender();

d3dxSRMgr->beginDraw();

list<cStephen\*>::iterator pete = gStephen.begin();

while(pete != gStephen.end())

{

(\*pete)->setSpritePos(stephenPos);

(\*pete)->update(dt);

d3dxSRMgr->setTheTransform((\*pete)->getSpriteTransformMatrix());

d3dxSRMgr->drawSprite((\*pete)->getTexture(),&((\*pete)->getSourceRect()),NULL,NULL,0xFFFFFFFF);

++pete;

pstephenPos.x = stephenTrans.x;

pstephenPos.y = stephenTrans.y;

stephenPos = D3DXVECTOR3(stephenTrans.x, stephenTrans.y,0);

}

list<cExplosion\*>::iterator mike = gExplode.begin();

while(mike != gExplode.end())

{

if((\*mike)->isActive() == false)

{

mike = gExplode.erase(mike);

}

else

{

(\*mike)->update(timeElapsed);

d3dxSRMgr->setTheTransform((\*mike)->getSpriteTransformMatrix());

d3dxSRMgr->drawSprite((\*mike)->getTexture(),&((\*mike)->getSourceRect()),NULL,NULL,0xFFFFFFFF);

++mike;

}

}

previousTime = currentTime;

OutputDebugString("timeElapsed > fpsRate");

timeElapsed = 0.0f;

d3dxSRMgr->endDraw();

EnemyFont->printText(gEnemiesKilledStr,textPos);

if(gameWater == true)

{

MenuFont->printText("\nBackground 1: Water \nPress 8 when you're done!", menuTextPos);

}

if(gameLightning == true)

{

MenuFont->printText("\nBackground 2: Lightning \nPress 8 when you're done!", menuTextPos);

}

if(gameFire == true)

{

MenuFont->printText("\nBackground 3: Fire \nPress 8 when you're done!", menuTextPos);

}

if(gameSpace == true)

{

MenuFont->printText("\nBackground 4: Space \nPress 8 when you're done!", menuTextPos);

}

if(gameCity == true)

{

MenuFont->printText("\nBackground 5: City \nPress 8 when you're done!", menuTextPos);

}

if(gameForest == true)

{

MenuFont->printText("\nBackground 6: Forest \nPress 8 when you're done!", menuTextPos);

}

if(gameIce == true)

{

MenuFont->printText("\nBackground 7: Ice \nPress 8 when you're done!", menuTextPos);

}

}

d3dMgr->endRender();

}

}

}

d3dxSRMgr->cleanUp();

d3dMgr->clean();

return (int) msg.wParam;

}