CÓDIGO DEL FICHERO MAIN.CPP INICIAL

#include <stdlib.h>

#include "Application.h"

#include "Globals.h"

#include "SDL/include/SDL.h"

#pragma comment( lib, "SDL/libx86/SDL2.lib" )

#pragma comment( lib, "SDL/libx86/SDL2main.lib" )

enum main\_states

{

MAIN\_CREATION,

MAIN\_START,

MAIN\_UPDATE,

MAIN\_FINISH,

MAIN\_EXIT

};

Application\* App = NULL;

int main(int argc, char \*\* argv)

{

int main\_return = EXIT\_FAILURE;

main\_states state = MAIN\_CREATION;

while (state != MAIN\_EXIT)

{

switch (state)

{

case MAIN\_CREATION:

LOG("Application Creation --------------");

App = new Application();

state = MAIN\_START;

break;

case MAIN\_START:

LOG("Application Init --------------");

if (App->Init() == false)

{

LOG("Application Init exits with error -----");

state = MAIN\_EXIT;

}

else

{

state = MAIN\_UPDATE;

LOG("Application Update --------------");

}

break;

case MAIN\_UPDATE:

{

int update\_return = App->Update();

if (update\_return == UPDATE\_ERROR)

{

LOG("Application Update exits with error -----");

state = MAIN\_EXIT;

}

if (update\_return == UPDATE\_STOP)

state = MAIN\_FINISH;

}

break;

case MAIN\_FINISH:

LOG("Application CleanUp --------------");

if (App->CleanUp() == false)

{

LOG("Application CleanUp exits with error -----");

}

else

main\_return = EXIT\_SUCCESS;

state = MAIN\_EXIT;

break;

}

}

delete App;

LOG("Bye :)\n");

return main\_return;

}

CÓDIGO DEL FICHERO APPLICATION.H INICIAL

#ifndef \_\_Application\_h\_\_

#define \_\_Application\_h\_\_

#include<list>

#include "Globals.h"

#include "Module.h"

class ModuleRender;

class ModuleWindow;

class ModuleTextures;

class ModuleInput;

class Application

{

public:

Application();

~Application();

bool Init();

bool CleanUp();

update\_status Update();

public:

ModuleRender\* renderer = nullptr;

ModuleWindow\* window = nullptr;

ModuleTextures\* textures = nullptr;

ModuleInput\* input = nullptr;

private:

std::list<Module\*> modules;

};

extern Application\* App;

#endif // \_\_Application\_h\_\_

CÓDIGO DEL FICHERO APPLICATION.CPP INICIAL

#ifndef \_\_Application\_cpp\_\_

#define \_\_Application\_cpp\_\_

#include "Application.h"

#include "ModuleWindow.h"

#include "ModuleRender.h"

#include "ModuleTextures.h"

#include "ModuleInput.h"

using namespace std;

Application::Application()

{

// Order matters: they will Init/start/update in this order

modules.push\_back(window = new ModuleWindow());

modules.push\_back(renderer = new ModuleRender());

modules.push\_back(textures = new ModuleTextures());

modules.push\_back(input = new ModuleInput());

// TODO 7: Create a new "scene" module that loads a texture and draws it on the screen

// Homework: Create a new module to handle music and sound effects

}

Application::~Application()

{

// TODO 6: Free module memory and check the result in Dr. Memory

}

bool Application::Init()

{

bool ret = true;

for(list<Module\*>::iterator it = modules.begin(); it != modules.end() && ret; ++it)

ret = (\*it)->Init();

return ret;

}

// TODO 4: We need to have three updates, add them: PreUpdate Update PostUpdate

update\_status Application::Update()

{

update\_status ret = UPDATE\_CONTINUE;

for(list<Module\*>::iterator it = modules.begin(); it != modules.end() && ret == UPDATE\_CONTINUE; ++it)

ret = (\*it)->Update();

return ret;

}

bool Application::CleanUp()

{

bool ret = true;

for(list<Module\*>::reverse\_iterator it = modules.rbegin(); it != modules.rend() && ret; ++it)

ret = (\*it)->CleanUp();

return ret;

}

#endif // #ifndef \_\_Application\_cpp\_\_

CÓDIGO DEL FICHERO GLOBALS.CPP INICIAL

*#ifndef \_\_Globals\_h\_\_*

*#define \_\_Globals\_h\_\_*

*#include <windows.h>*

*#include <stdio.h>*

*#define LOG(format, ...) log(\_\_FILE\_\_, \_\_LINE\_\_, format, \_\_VA\_ARGS\_\_);*

*void log(const char file[], int line, const char\* format, ...);*

*enum update\_status*

*{*

*UPDATE\_CONTINUE = 1,*

*UPDATE\_STOP,*

*UPDATE\_ERROR*

*};*

*// Configuration -----------*

*#define SCREEN\_WIDTH 640*

*#define SCREEN\_HEIGHT 480*

*#define FULLSCREEN false*

*#define VSYNC true*

*#define TITLE "Super Awesome Game"*

*#define SPRITE\_FILE "sprites.png"*

*#endif // #ifndef \_\_Globals\_h\_\_*

CÓDIGO DEL FICHERO log.CPP INICIAL

#ifndef \_\_log\_cpp\_\_

#define \_\_log\_cpp\_\_

#include "Globals.h"

void log(const char file[], int line, const char\* format, ...)

{

static char tmp\_string[4096];

static char tmp\_string2[4096];

static va\_list ap;

// Construct the string from variable arguments

va\_start(ap, format);

vsprintf\_s(tmp\_string, 4096, format, ap);

va\_end(ap);

sprintf\_s(tmp\_string2, 4096, "\n%s(%d) : %s", file, line, tmp\_string);

OutputDebugString(tmp\_string2);

}

#endif // #ifndef \_\_log\_cpp\_\_

CÓDIGO DEL FICHERO MODULES.H INICIAL

#ifndef \_\_Modules\_h\_\_

#define \_\_modules\_h\_\_

class Application;

class Module

{

public:

Module()

{

}

virtual bool Init()

{

return true;

}

// TODO 3: Create a new method "Start()" that should be calling all modules just before the first frame

virtual update\_status Update()

{

return UPDATE\_CONTINUE;

}

virtual bool CleanUp()

{

return true;

}

};

#endif // ifndef \_\_Modules\_h\_\_

CÓDIGO DEL FICHERO MODULEINPUT.H

#ifndef \_\_ModuleInput\_h\_\_

#define \_\_ModuleInput\_h\_\_

#include "Module.h"

#include "Globals.h"

typedef unsigned \_\_int8 Uint8;

class ModuleInput : public Module

{

public:

ModuleInput();

~ModuleInput();

bool Init() override = 0;

bool CleanUp() override = 0;

update\_status Update() override = 0;

private:

const Uint8 \*keyboard = NULL;

};

#endif // #ifndef \_\_ModuleInput\_h\_\_

CÓDIGO DEL FICHERO MODULEINPUT.CPP INICIAL

#include "ModuleInput.h"

#include "Application.h"

#include "Globals.h"

#include "SDL/include/SDL.h"

ModuleInput::ModuleInput()

{}

// Destructor

ModuleInput::~ModuleInput()

{}

// Called before render is available

bool ModuleInput::Init()

{

LOG("Init SDL input event system");

bool ret = true;

SDL\_Init(0);

if (SDL\_InitSubSystem(SDL\_INIT\_EVENTS) < 0)

{

LOG("SDL\_EVENTS could not initialize! SDL\_Error: %s\n", SDL\_GetError());

ret = false;

}

return ret;

}

// Called every draw update

update\_status ModuleInput::Update()

{

SDL\_PumpEvents();

keyboard = SDL\_GetKeyboardState(NULL);

// TODO 1: Make the application properly close when ESC is pressed (do not use exit())

// Homework: Make the application close up when pressing “X” button of the window

return UPDATE\_CONTINUE;

}

// Called before quitting

bool ModuleInput::CleanUp()

{

LOG("Quitting SDL input event subsystem");

SDL\_QuitSubSystem(SDL\_INIT\_EVENTS);

return true;

}

CÓDIGO DEL FICHERO MODULEWINDOW.H INICIAL

#ifndef \_\_ModuleWindow\_h\_\_

#define \_\_ModuleWindow\_h\_\_

#include "Module.h"

#include "SDL/include/SDL.h"

class Application;

class ModuleWindow

{

public:

ModuleWindow();

// Destructor

virtual ~ModuleWindow();

// Called before quitting

bool Init() override = 0;

// Called before quitting

bool CleanUp() override = 0;

public:

//The window we'll be rendering to

SDL\_Window\* window = NULL;

//The surface contained by the window

SDL\_Surface\* screen\_surface = NULL;

};

#endif // \_\_ModuleWindow\_h\_\_

CÓDIGO DEL FICHERO MODULEWINDOW.CPP INICIAL

#include "ModuleWindow.h"

#include "Application.h"

#include "Globals.h"

ModuleWindow::ModuleWindow()

{

}

// Destructor

ModuleWindow::~ModuleWindow()

{

}

// Called before render is available

bool ModuleWindow::Init()

{

LOG("Init SDL window & surface");

bool ret = true;

if (SDL\_Init(SDL\_INIT\_VIDEO) < 0)

{

LOG("SDL\_VIDEO could not initialize! SDL\_Error: %s\n", SDL\_GetError());

ret = false;

}

else

{

//Create window

int width = SCREEN\_WIDTH;

int height = SCREEN\_HEIGHT;

Uint32 flags = SDL\_WINDOW\_SHOWN;

if (FULLSCREEN == true)

{

flags |= SDL\_WINDOW\_FULLSCREEN;

}

// TODO 2: Create options for RESIZABLE, SDL\_WINDOW\_BORDERLESS, SDL\_WINDOW\_RESIZABLE,

// SDL\_WINDOW\_FULLSCREEN\_DESKTOP (same way as with FULLSCREEN)

window = SDL\_CreateWindow(TITLE, SDL\_WINDOWPOS\_UNDEFINED, SDL\_WINDOWPOS\_UNDEFINED, width, height, flags);

if (window == NULL)

{

LOG("Window could not be created! SDL\_Error: %s\n", SDL\_GetError());

ret = false;

}

else

{

//Get window surface

screen\_surface = SDL\_GetWindowSurface(window);

}

}

return ret;

}

// Called before quitting

bool ModuleWindow::CleanUp()

{

LOG("Destroying SDL window and quitting all SDL systems");

//Destroy window

if (window != NULL)

{

SDL\_DestroyWindow(window);

}

//Quit SDL subsystems

SDL\_Quit();

return true;

}

CÓDIGO DEL FICHERO MODULERENDER.H INICIAL

#ifndef \_\_ModuleRender\_h\_\_

#define \_\_moduleRender\_h\_\_

#include "Module.h"

#include "Globals.h"

struct SDL\_Texture;

struct SDL\_Renderer;

struct SDL\_Rect;

class ModuleRender : public Module

{

public:

ModuleRender();

~ModuleRender();

bool Init() override = 0;

bool CleanUp() override = 0;

bool Blit(SDL\_Texture\* texture, int x, int y, SDL\_Rect\* section) = 0;

update\_status Update() override = 0;

public:

SDL\_Renderer\* renderer = NULL;

};

#endif // \_\_ModuleRender\_h\_\_

CÓDIGO DEL FICHERO MODULERENDER.CPP INICIAL

#include "ModuleRender.h"

#include "Application.h"

#include "Globals.h"

#include "ModuleWindow.h"

#include "SDL/include/SDL.h"

ModuleRender::ModuleRender()

{

}

// Destructor

ModuleRender::~ModuleRender()

{

}

// Called before render is available

bool ModuleRender::Init()

{

LOG("Creating Renderer context");

bool ret = true;

Uint32 flags = 0;

if (VSYNC == true)

{

flags |= SDL\_RENDERER\_PRESENTVSYNC;

}

renderer = SDL\_CreateRenderer(App->window->window, -1, flags);

if (renderer == NULL)

{

LOG("Renderer could not be created! SDL\_Error: %s\n", SDL\_GetError());

ret = false;

}

return ret;

}

// Called every draw update

update\_status ModuleRender::Update()

{

SDL\_RenderClear(renderer);

// TODO 5: Now that we have PreUpdate/PostUpdate/Update move things around so we can render :)

SDL\_RenderPresent(renderer);

return UPDATE\_CONTINUE;

}

// Called before quitting

bool ModuleRender::CleanUp()

{

LOG("Destroying renderer");

//Destroy window

if (renderer != NULL)

{

SDL\_DestroyRenderer(renderer);

}

return true;

}

// Blit to screen

bool ModuleRender::Blit(SDL\_Texture\* texture, int x, int y, SDL\_Rect\* section)

{

bool ret = true;

SDL\_Rect rect;

rect.x = x;

rect.y = y;

if (section != NULL)

{

rect.w = section->w;

rect.h = section->h;

}

else

{

SDL\_QueryTexture(texture, NULL, NULL, &rect.w, &rect.h);

}

if (SDL\_RenderCopy(renderer, texture, section, &rect) != 0)

{

LOG("Cannot blit to screen. SDL\_RenderCopy error: %s", SDL\_GetError());

ret = false;

}

return ret;

}

CÓDIGO DEL FICHERO MODULETEXTURES.H INICIAL

#ifndef \_\_ModuleTextures\_h\_\_

#define \_\_moduleTextures\_h\_\_

#include<list>

#include "Module.h"

#include "Globals.h"

struct SDL\_Texture;

class ModuleTextures

{

public:

ModuleTextures();

~ModuleTextures();

bool Init() override = 0;

bool CleanUp() override = 0;

SDL\_Texture\* const Load(const char\* path) = NULL;

private:

std::list<SDL\_Texture\*> textures;

};

#endif // \_\_ModuleTextures\_h\_\_

CÓDIGO DEL FICHERO MODULETEXTURES.CPP INICIAL

#include "ModuleTextures.h"

#include "Application.h"

#include "Globals.h"

#include "ModuleRender.h"

#include "ModuleTextures.h"

#include "SDL/include/SDL.h"

#include "SDL\_image/include/SDL\_image.h"

#pragma comment( lib, "SDL\_image/libx86/SDL2\_image.lib" )

using namespace std;

ModuleTextures::ModuleTextures()

{

}

// Destructor

ModuleTextures::~ModuleTextures()

{

}

// Called before render is available

bool ModuleTextures::Init()

{

LOG("Init Image library");

bool ret = true;

// load support for the PNG image format

int flags = IMG\_INIT\_PNG;

int init = IMG\_Init(flags);

if ((init & flags) != flags)

{

LOG("Could not initialize Image lib. IMG\_Init: %s", IMG\_GetError());

ret = false;

}

return ret;

}

// Called before quitting

bool ModuleTextures::CleanUp()

{

LOG("Freeing textures and Image library");

for (list<SDL\_Texture\*>::iterator it = textures.begin(); it != textures.end(); ++it)

SDL\_DestroyTexture(\*it);

textures.clear();

return true;

}

// Load new texture from file path

SDL\_Texture\* const ModuleTextures::Load(const char\* path)

{

SDL\_Texture\* texture = NULL;

SDL\_Surface\* surface = IMG\_Load(path);

if (surface == NULL)

{

LOG("Could not load surface with path: %s. IMG\_Load: %s", path, IMG\_GetError());

}

else

{

texture = SDL\_CreateTextureFromSurface(App->renderer->renderer, surface);

if (texture == NULL)

{

LOG("Unable to create texture from surface! SDL Error: %s\n", SDL\_GetError());

}

else

{

textures.push\_back(texture);

}

SDL\_FreeSurface(surface);

}

return texture;

}