Tutorial 2 – Opening a Window

Before starting this tutorial, you must complete the last one and have it working. Failure to do so will result in not being able to complete these steps.

In this tutorial will we be going through the steps needed to open a simple window using SDL. The code will be supplied with explanations. It is advised you create a new branch in your GitHub repo before starting out as you will be changing code over time and it's always good to have a reference of how something was done in future work. Branches being a snapshot of the project at any given time. (If making a branch, name it something appropriate, e.g. "WindowOpeningTut2").

1. As games will tend to accumulate a lot of constants, we will need to create a file to hold these.
    a. Create a "constants.h" file (ensure it is in the Header Files Folder) and input the following code:

    ```
    #pragma once
    //screen Dimensions
    #define SCREEN_WIDTH 512
    #define SCREEN_HEIGHT 416
    ```

    constants are typically written in this format and do not require any semi colon after the statement.
2. Back I the main/source file include the constants.h file at the top under your SDL includes, unlike the SDL includes it will require double quotation marks instead of chevrons (remember back to classes from semester 1).
3. We will be using iostream in our project for outputting information to console, so include this and add the using namespace std if you so wish.
4. Next, we need to add some Global variables. After your using namespace add the following code:

    ```
    //Globals
    SDL_Window* g_window = nullptr;
    ```

    here we are creating a pointer of type SDL_Window and initializing it to nullptr, you can also use NULL if you wish but nullptr is the new standard.

5. We will be splitting the code up in to separate functions. This will give us bite size chunks to work through. Add the following function prototypes to your file:

    ```
    //Function prototypes
    bool InitSDL();
    void CLoseSDL();
    ```

6. Then below the main function add the function bodies, easiest way is to copy the prototypes, paste below, remove the semi colon and add opening and closing braces. To use any SDL functions, we must initialise it first.
    a. In the InitSDL function scope we will add a simple if statement to set it up which will return false should something go wrong. Add the following:

```
//Setup SDL
if(SDL_Init(SDL_INIT_VIDEO) < 0)
{
	cout << "SDL did not initialise. Error: " << SDL_GetError();
	return false;
}
```

The SDL_INIT_VIDEO flag passed into the SDL_Init function is to specify that we are using the video subsystems.

The other flags available are:

SDL_INIT_TIMER	The Timer subsystem.
SDL_INIT_AUDIO	The Audio subsystem.
SDL_INIT_CDROM	The CDRom subsystem.
SDL_INIT_JOYSTICK	The Joystick subsystem
SDL_INIT_EVERYTHING	All of the above
SDL_INIT_NOPARACHUTE	Prevents SDL from catching fatal signals
SDL_INIT_EVENTTHREAD	Runs the event manager in a separate thread.

Multiple flags can be set by using the | operator.

For example: SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO)

SDL_Init will return -1 to signify the initialisation failed and 0 on success. On failure we need to inform the user what went wrong. Using the SDL_GetError() function allows us to detail what the error was.

If setup goes well, we need to create a window. This will use the dimensions we set in the Constants header file.

```
else
{
	//setup passed so create window
	g_window = SDL_CreateWindow("Games Engine Creation",
		SDL_WINDOWPOS_UNDEFINED,
		SDL_WINDOWPOS_UNDEFINED,
		SCREEN_WIDTH,
		SCREEN_HEIGHT,
		SDL_WINDOW_SHOWN);
	//did the window get created?
	if(g_window == nullptr)
	{
		//window failed
		cout << "Window was not created. Error: " << SDL_GetError();
		return false;
	}
}
```

The SDL_CreateWindow() function will create us a window and return a pointer to a SDL_Window. The parameters are as follows:

- The first is a string that will be displayed in the bar at the top of the window.
- The next two are screen positions, X and Y. SDL_WINDOW_UNDEFINED allows us to not set a particular position.
- The fourth is an integer to define the width of the new window.
- The fifth is also an integer, but this one specifies the height of the new window.
- The final is a flag to state whether the window should be instantly shown.

The other available window flags are:

SDL_WINDOW_FULLSCREEN	Fullscreen window
SDL_WINDOW_FULLSCREEN_DESKTOP	Fullscreen window at current desktop resolution
SDL_WINDOW_OPENGL	Window usable with OpenGL context
SDL_WINDOW_HIDDEN	Window is not visible
SDL_WINDOW_BORDERLESS	No window decoration
SDL_WINDOW_RESIZABLE	Window can be resized
SDL_WINDOW_MINIMIZED	Window is minimized
SDL_WINDOW_MAXIMISED	Window is maximised
SDL_WINDOW_INPUT_GRABBED	Window has grab input focus
SDL_WINDOW_INPUT_FOCUS	Window has input focus
SDL_WINDOW_MOUSE_FOCUS	Window has mouse focus
SDL_WINDOW_FORIEGN	Window not created by SDL
SDL_WINDOW_ALLOW_HIGHDPI	Window should be created with High-DPI format (if supported)

Back to the code, we make a check to see if the window was created. It the pointer returned was NULL there was an error. We need to catch this error and output the error to the user.

b. If the code progresses through the function without returning false, we simply need add return true before the last closing brace of the scope.

7. No to sort our CloseSDL function. This is where we free up the memory used and close the SDL subsystems that we created.

a. First free the memory from the window:

```
//release the window
SDL_DestroyWindow(g_window);
g_window = nullptr;
```

b. Next, call the SDL functions to quit the subsystems:

```
//quit SDL subsystems
IMG_Quit();
SDL_Quit();
```

ensure to use the correct SDL_Quit, there is also an SDL_QUIT

8. Now return to the main so we can use our new functions. Add the following:

```
//check if sdl was setup correctly
if(InitSDL())
{
	SDL_Delay(5000);
}

CLoseSDL();

return 0;
```

What we are doing here is if InitSDL is true, we run a delay which will open a window for 5 seconds (SDL_Delay takes in milliseconds as a parameter) and then the window and program will end.

9. Build and run your program to ensure everything is working.

In the next tutorial we will be looking at SDL events to allow us to close the window using the X in the window.