**Spike:** Task 24

**Title:** Sprites and Graphics

**Author:** Sam Huffer, 101633177

# Goals / deliverables:

* Display a background image that can be toggled on or off with the 0 key.
* Load another image with three identifiable sub-regions (tiles) within it to serve as a sprite sheet.
* Define three rectangles that specify the sub-region for each tile’s image
* Display each tile’s image to a unique random location using a toggle on or off in response to the 1, 2 and 3 number keys.

# Technologies, Tools, and Resources used:

* Visual Studio 2019
* Microsoft Word
* Paint
* SDL2
* Online Resources
  1. Prerequisite game management content: <https://www.youtube.com/watch?v=ATa_joa6Gzg&list=PLhJr2LOK-xwxQlevIZ97ZABLw72Eu9he7&index=1>
  2. Prerequisite timer content: <https://www.youtube.com/watch?v=z9U-Jif4RVU&list=PLhJr2LOK-xwxQlevIZ97ZABLw72Eu9he7&index=2>
  3. Prerequisite game entity content: <https://www.youtube.com/watch?v=DI6q73p3rEI&list=PLhJr2LOK-xwxQlevIZ97ZABLw72Eu9he7&index=3>
  4. Rendering images: <https://www.youtube.com/watch?v=bKiejuOaJtU&list=PLhJr2LOK-xwxQlevIZ97ZABLw72Eu9he7&index=4>
  5. Handling images with asset manager: <https://www.youtube.com/watch?v=UPZol-0fn08&list=PLhJr2LOK-xwxQlevIZ97ZABLw72Eu9he7&index=5>
  6. Using sprite sheets: <https://www.youtube.com/watch?v=k70aBlef-20&list=PLhJr2LOK-xwxQlevIZ97ZABLw72Eu9he7&index=6>
  7. Random numbers in C++: <http://www.cplusplus.com/reference/cstdlib/rand/>

# Tasks undertaken:

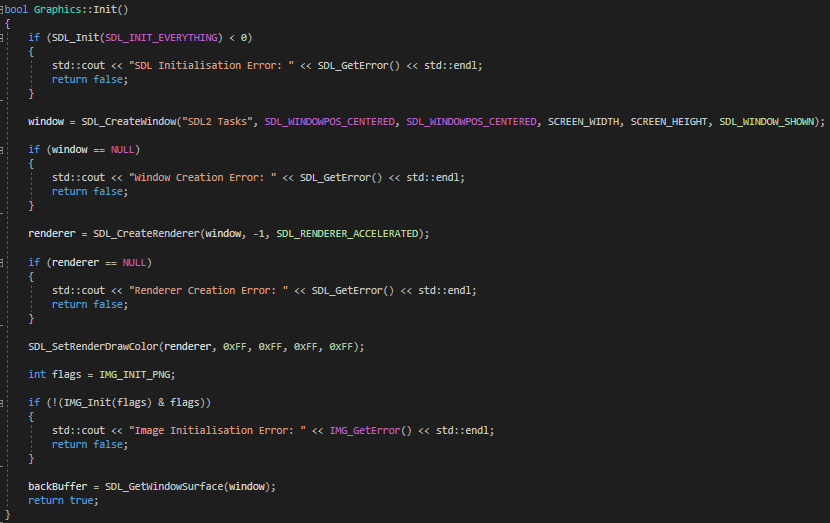
* I copied the task 23 spike report into the task folder, stripping out the spike report’s original content and replacing it with goals and resources pertaining to the task at hand.
* I found an image on my com­puter to use as the window background for this task, and several other images that could comprise a sprite sheet. For the former, I opened it up in paint, then saved it in “As­sets/Images” as a .png file. For the latter, I lined them all up in Word, then selected them all and copied them to Paint, be­fore saving them as a sin­gle .png in the same folder.

Figure : Graphics.Init()

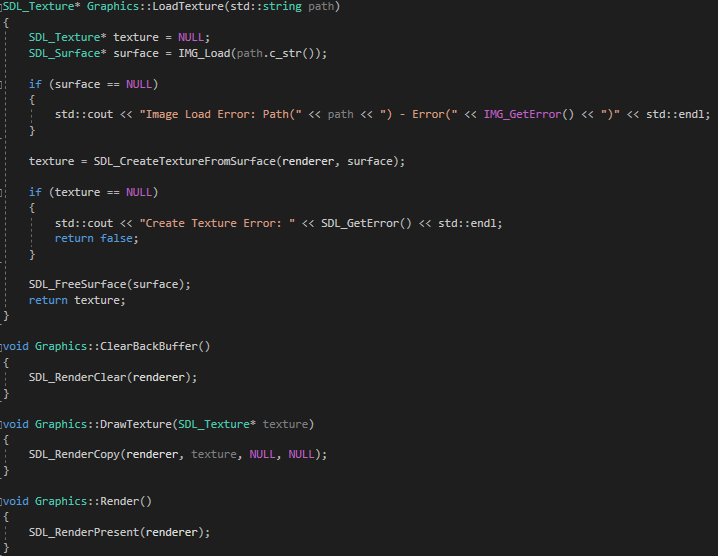
* I had a look at the YouTube playlist I got the SDL\_Mixer tutorial from to see what I could find that was rele­vant for this task, and found several tutorials for handling images. How­ever, when I looked at it, it had some established classes where I wasn’t sure if I would need some of their content later on or where I should put the image-related con­tent in my current structure. There­fore, I had a look at some earlier tu­torials in the series and reorganised and added basic game manage­ment (res. 1), time management (res. 2) and game entity (res. 3) functionality in line with them to get my VS project in line with where the example project is for the image rendering tutorials while also retaining existing functionality in the events polling loop for playing audio files from keystrokes.

Figure : Graphic’s methods LoadTexture(), ClearBackBuffer(), DrawTexture() and Render().

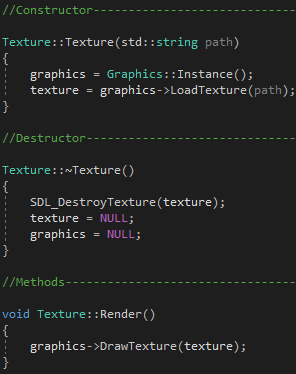
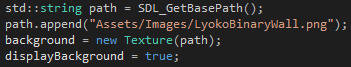
* As part of the prerequisite tutorials, I created a Graphics class to handle loading and rendering graphics, but so far it hadn’t really done anything beyond configure the window when it was initial­ised. Here, I updated it to also initialise a renderer and the image library as part of its own initialisation (fig. 1).
* Following res. 4 and this task’s objectives, I added to Graphics a LoadTexture() method to allow the loading of images into the program from a file, with appropriate error checking, as well as ClearBackBuffer(), DrawTexture() and Render() to properly render loaded images to the screen.

Figure 4: the lines in GameManager.GameManager() for loading the background.

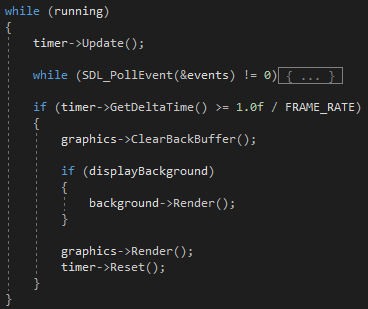
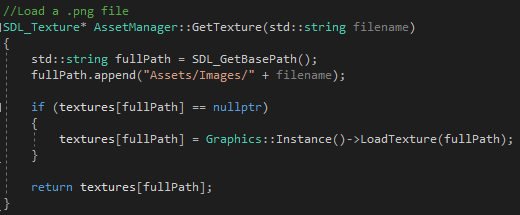
* I added a Texture class that encapsulates SDL\_Texture, and can make calls to Graphics to load and draw itself (fig. 3).
* I added to GameManager a Texture\* field background and updated its constructor to load up an image to be the background (fig. 4). I then added in the rendering section of GameManager.Run()’s game loop a call to background.Render() encapsulated in an if statement checking if the background is active (fig. 5). I then added to the keyboard input switch a case for the 0 key to toggle background.active, changing the keys mapped to other cases in that switch to make room for it.

Figure 5: the game loop of GameManager.Run(), focusing on the rendering section.

Figure 3: Early implementation of the Texture class.

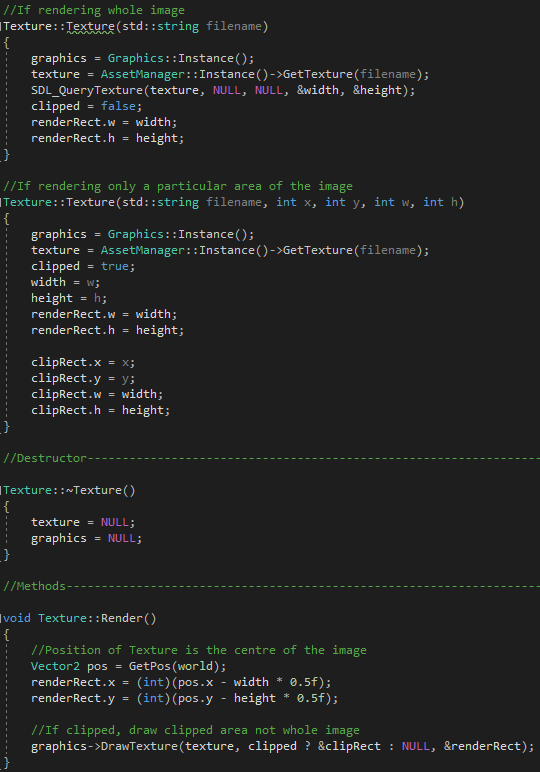
* Following res. 5 and this task’s objectives, I added to AssetManager the method GetTexture() to load / retrieve textures if given a file name (fig. 6). I then updated Texture’s constructor to make a call to AssetManager.GetTexture() instead of Graphics.LoadTexture(), and updated the call to Texture’s constructor in GameManager.Run() for instantiating the background, as Texture now loading itself using AssetManager.GetTexture() only requires the name of the file rather than a longer path name.

Figure 6: AssetManager.GetTexture(). If an image has previously been loaded, it returns its pointer rather than loading it again.

* Following res. 6 and the task’s objectives, I updated Texture such that its position is in its centre (fig. 7, Render()), by default its width and height is the width and height of the image in pixels (fig. 7, first Texture()), and that you can select a given portion of the image to render, cropping out the rest (fig. 7, second Texture()). I also updated Graphics.DrawTexture() to accept a Tex­ture’s clipped and whole SDL\_Rects as pa­rameters, passing them to its call to SDL\_RenderCopy() in that order instead of passing NULL and NULL.

Figure 7: Later implementation of Texture, centring its position within the image and allowing for cropping all but a selected area.

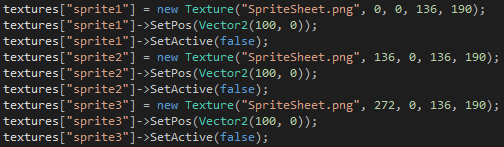
* I removed from GameManager the back­ground Texture\* field and instead added a map of Texture\*s, with strings as keys, add­ing the background Texture\* to it. I then added several sprites to the map, each loading a different area of SpriteSheet.png (fig. 8), and modified the render section of the game loop to render each active Tex­ture\* in textures rather than make individ­ual calls for each pair.
* I added a method ToggleSprite() to toggle whether a sprite passed to it was active or not, and if it became active, to give it a ran­dom position (following res. 7). I then added cases to the keyboard input switch to pass each sprite to Tog­gleSprite() when its corresponding key is pressed.
* Up until now the background image had only been rendering at its own size rather than the size of the window, so I added to Texture getter and setter methods for its width and height, and assigned to the background the width and height of the window.

Figure 8: sprites that will render a different area of SpriteSheet.png.

# What we found out:

* More about how child game object local and world positions and rotations are calculated.
* More about how time scale is calculated.
* How to load image files into a game using SDL.
* How to render images to the screen using SDL.
* How to treat the middle of an image as its position.
* How to render portions of an image rather than a whole image using SDL.
* How to generate random numbers in C++.
* Using the “inline” keyword in a .h file when declaring a method or operator seems to allow you to implement it there rather than in a .cpp file.
* When scaling the background, I had to apply width and height changes to rendRect.w and .h as well as the width and height fields; otherwise, it’d result in weird positioning issues. If I wanted to change the width and height of clipped sprites, I’d also need to test how the current getter and setter meth­ods interact with them considering they use clipRect in their rendering as well.

# Screenshot

Figure 9: The program running and rendering the background and randomly-positioned sprites.