Spike: Task 24
Title: Sprites and Graphics
Author: Sam Huffer, 101633177

Goals / deliverables:

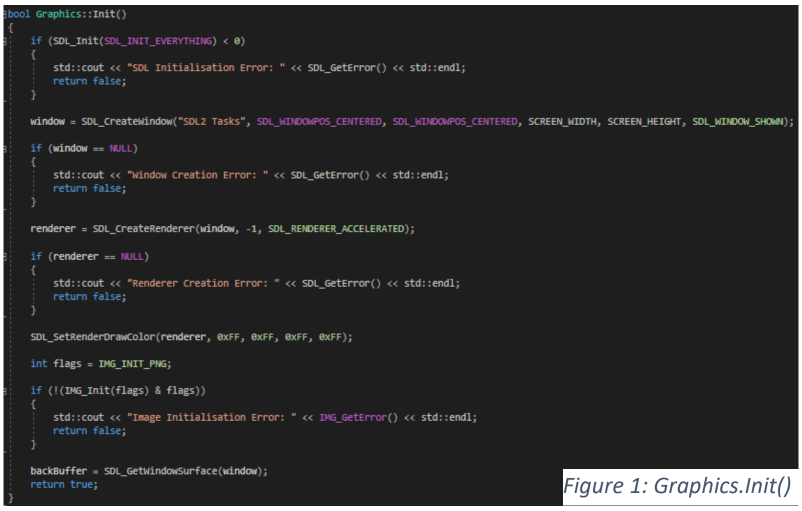
- Display a background image that can be toggled on or off with the 0 key.
- Load another image with three identifiable sub-regions (tiles) within it to serve as a sprite sheet.
- Define three rectangles that specify the sub-region for each tile's image
- Display each tile's image to a unique random 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:

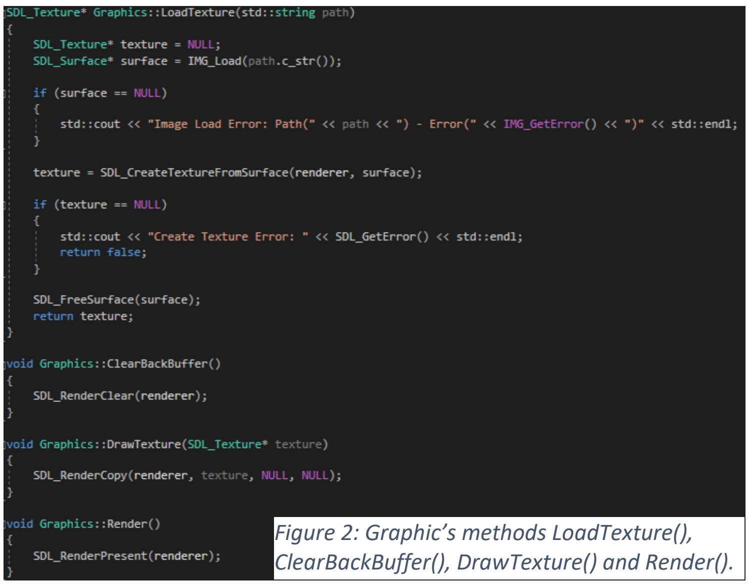
- I copied the task 23 spike report into the task folder, stripping out the spike report's original content and replacing it with goals and resources pertaining to the task at hand.
- I found an image on my computer to use as the window background for this task, and several other images that could comprise 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.



```
bool Graphics::Init()
{
    if (SDL_Init(SDL_INIT_EVERYTHING) < 0)
    {
        std::cout << "SDL Initialisation Error: " << SDL_GetError() << std::endl;
        return false;
    }

    window = SDL_CreateWindow("SDL2 Tasks", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_SHOWN);

    if (window == NULL)
    {
        std::cout << "Window Creation Error: " << SDL_GetError() << std::endl;
        return false;
    }

    renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);

    if (renderer == NULL)
    {
        std::cout << "Renderer Creation Error: " << SDL_GetError() << std::endl;
        return false;
    }

    SDL_SetRenderDrawColor(renderer, 0xFF, 0xFF, 0xFF, 0xFF);

    int flags = IMG_INIT_PNG;

    if (!(IMG_Init(flags) & flags))
    {
        std::cout << "Image Initialisation Error: " << IMG_GetError() << std::endl;
        return false;
    }

    backBuffer = SDL_GetWindowSurface(window);
    return true;
}
```

Figure 1: Graphics.Init()

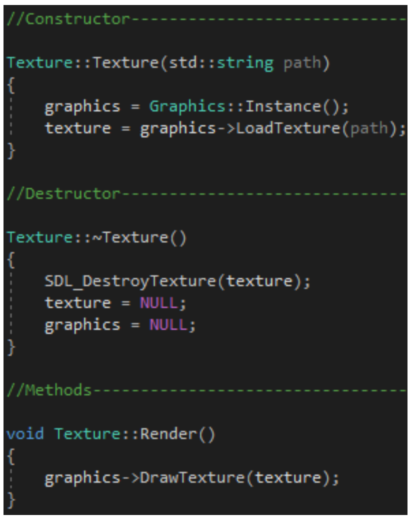
- I had a look at the YouTube playlist I got the SDL_Mixer tutorial from to see what I could find that was relevant for this task, and found several tutorials for handling images. However, when I looked at it, it had some established classes where I wasn't sure if I would need some of their content later 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.



```
SDL_Texture* Graphics::LoadTexture(std::string path)
{
    SDL_Texture* texture = NULL;
    SDL_Surface* surface = IMG_Load(path.c_str());

    if (surface == NULL)
    {
        std::cout << "Image Load Error: Path(" << path << ") - Error(" << IMG_GetError() << ")" << std::endl;
    }

    texture = SDL_CreateTextureFromSurface(renderer, surface);

    if (texture == NULL)
    {
        std::cout << "Create Texture Error: " << SDL_GetError() << std::endl;
        return false;
    }

    SDL_FreeSurface(surface);
    return texture;
}

void Graphics::ClearBackBuffer()
{
    SDL_RenderClear(renderer);
}

void Graphics::DrawTexture(SDL_Texture* texture)
{
    SDL_RenderCopy(renderer, texture, NULL, NULL);
}

void Graphics::Render()
{
    SDL_RenderPresent(renderer);
}
```

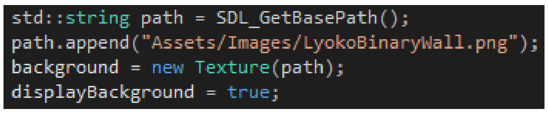
Figure 2: Graphic's methods LoadTexture(), ClearBackBuffer(), DrawTexture() and Render().

- 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.
- 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.
- 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



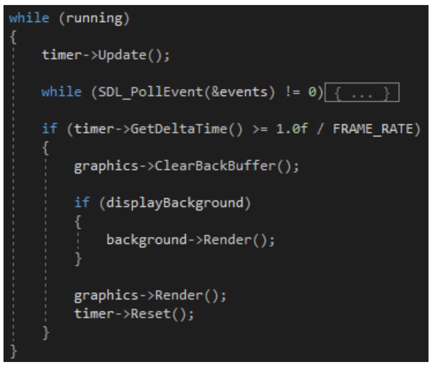
```
//Constructor---------------------------

Texture::Texture(std::string path)
{
    graphics = Graphics::Instance();
    texture = graphics->LoadTexture(path);
}

//Destructor----------------------------

Texture::~Texture()
{
    SDL_DestroyTexture(texture);
    texture = NULL;
    graphics = NULL;
}

//Methods-------------------------------

void Texture::Render()
{
    graphics->DrawTexture(texture);
}
```

Figure 3: Early implementation of the Texture class.



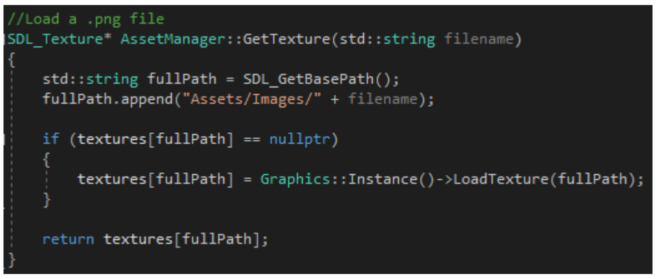
```
std::string path = SDL_GetBasePath();
path.append("Assets/Images/LyokoBinaryWall.png");
background = new Texture(path);
displayBackground = true;
```

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



```
while (running)
{
    timer->Update();

    while (SDL_PollEvent(&events) != 0) { ... }

    if (timer->GetDeltaTime() >= 1.0f / FRAME_RATE)
    {
        graphics->ClearBackBuffer();

        if (displayBackground)
        {
            background->Render();
        }

        graphics->Render();
        timer->Reset();
    }
}
```

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

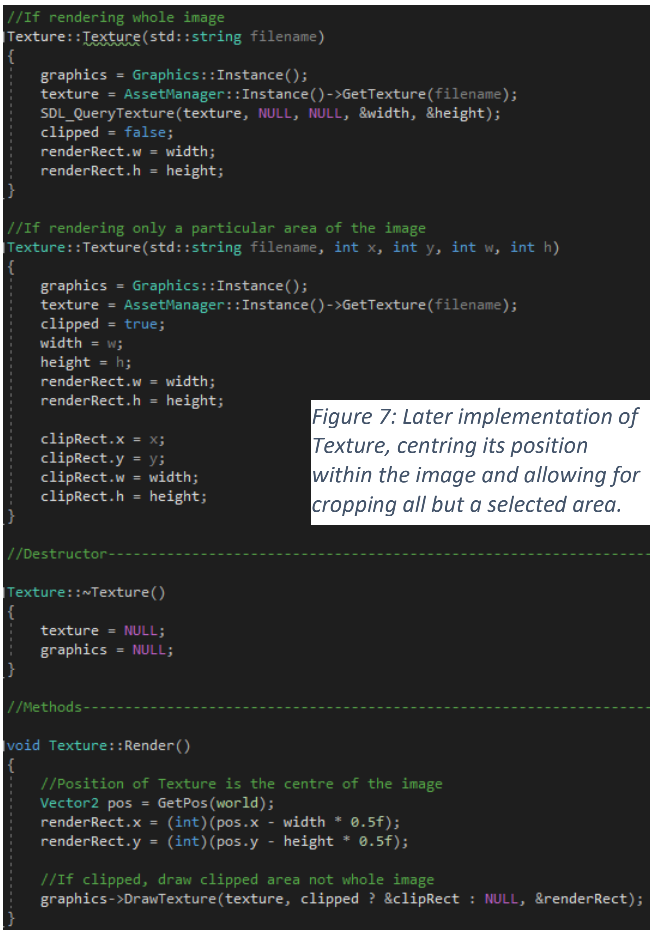


```
//Load a .png file
SDL_Texture* AssetManager::GetTexture(std::string filename)
{
    std::string fullPath = SDL_GetBasePath();
    fullPath.append("Assets/Images/" + filename);

    if (textures[fullPath] == nullptr)
    {
        textures[fullPath] = Graphics::Instance()->LoadTexture(fullPath);
    }

    return textures[fullPath];
}
```

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

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.

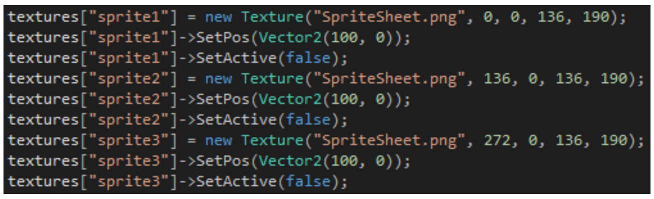
- Following res. 6 and the task's objectives, I updated Texture such that its position is in its centre (fig. 7, Render()), by default its width and height is the width and height of the image in pixels (fig. 7, first Texture()), and that you can select a given portion of the image to render, cropping 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.



```
//If rendering whole image
Texture::Texture(std::string filename)
{
    graphics = Graphics::Instance();
    texture = AssetManager::Instance()->GetTexture(filename);
    SDL_QueryTexture(texture, NULL, NULL, &width, &height);
    clipped = false;
    renderRect.w = width;
    renderRect.h = height;
}

//If rendering only a particular area of the image
Texture::Texture(std::string filename, int x, int y, int w, int h)
{
    graphics = Graphics::Instance();
    texture = AssetManager::Instance()->GetTexture(filename);
    clipped = true;
    width = w;
    height = h;
    renderRect.w = width;
    renderRect.h = height;

    clipRect.x = x;
    clipRect.y = y;
    clipRect.w = width;
    clipRect.h = height;
}

//Destructor---------------------------------------------------------------

Texture::~Texture()
{
    texture = NULL;
    graphics = NULL;
}

//Methods------------------------------------------------------------------

void Texture::Render()
{
    //Position of Texture is the centre of the image
    Vector2 pos = GetPos(world);
    renderRect.x = (int)(pos.x - width * 0.5f);
    renderRect.y = (int)(pos.y - height * 0.5f);

    //If clipped, draw clipped area not whole image
    graphics->DrawTexture(texture, clipped ? &clipRect : NULL, &renderRect);
}
```

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.



```
textures["sprite1"] = new Texture("SpriteSheet.png", 0, 0, 136, 190);
textures["sprite1"]->SetPos(Vector2(100, 0));
textures["sprite1"]->SetActive(false);
textures["sprite2"] = new Texture("SpriteSheet.png", 136, 0, 136, 190);
textures["sprite2"]->SetPos(Vector2(100, 0));
textures["sprite2"]->SetActive(false);
textures["sprite3"] = new Texture("SpriteSheet.png", 272, 0, 136, 190);
textures["sprite3"]->SetPos(Vector2(100, 0));
textures["sprite3"]->SetActive(false);
```

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

- 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.

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.