Spike: Spike No.18
Title: Task 18 – Sprites and Graphics

Author: Khang Trinh - 102118468

Goals / Deliverables:

The goal of this spike is to teach the developer how to load and display images and sprites.

Technologies, Tools, and Resources used:

- Visual Studio 2017

Useful Links:

1. How to load and animate sprites in a sprite sheet
   https://youtu.be/2NVgHrOFneg
2. Intro to sprite sheet animation
   https://gamedevelopment.tutsplus.com/tutorials/an-introduction-to-spritesheet-animation--gamedev-13099
3. Why use a sprite sheet?
   https://www.codeandweb.com/what-is-a-sprite-sheet-performance

Tasks undertaken:

I. Texture handling

Step 1: Load files

To load an image into the game, you need to perform 2 smaller steps

- Load the image into a surface
- Create the texture from that surface and assign the rendering context to the renderer

```
SDL_Texture *LoadTexture(string filepath, SDL_Renderer *renderer) {
    SDL_Texture *texture = nullptr;
    SDL_Surface *surface = SDL_LoadBMP(filepath.c_str());
    texture = SDL_CreateTextureFromSurface(renderer, surface);

    SDL_FreeSurface(surface);

    return texture;
}
```

```
image1 = LoadTexture("image1.bmp", renderer);
```

Fig 1. How loading textures in SDL looks like

Step 2: Display images

```
SDL_RenderClear(renderer);
SDL_RenderCopy(renderer, image1, NULL, NULL);
SDL_RenderPresent(renderer);
```

Fig 2. Always clear the screen before displaying a new image

II. Sprites handling

For this section, we'll be using the image below. Each "frame" of the sheet is a stripe of colour. In other words, this sheet has 3 frames.

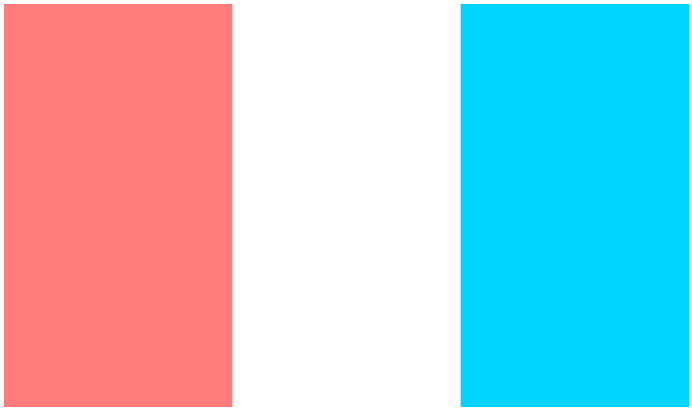

Fig 1. The sprite sheet we'll be using for this Spike

Step 1: Define width and height of a frame

A sprite sheet will always come with multiple frames tightly packed next to each other. To "synthesize" them into frames, we need to first define how big a frame is in the sheet.

```
int frameWidth, frameHeight;
int textureWidth, textureHeight;
SDL_QueryTexture(image2, NULL, NULL, &textureWidth, &textureHeight);
```

Fig 2. How to get the size of the entire sheet

```
frameWidth = textureWidth / 3;
frameHeight = textureHeight;
```

Fig 3. The sheet is 3 frames wide and 1 frame high, so we'll divide the size accordingly

Step 2: Create a Rect

We've defined the dimensions of a frame, now we need to create a rect - something that can use this dimension data. Think of this like a viewport that only shows an area of a texture.

```
SDL_Rect imageRect;
imageRect.x = imageRect.y = 0;
imageRect.w = frameWidth;
imageRect.h = frameHeight;
```

Fig 4. How to create a rect

Step 3: Assign a frame to the rect

We can then make the rect display a particular frame by moving it using the frame unit we've defined earlier (ie. Move 1 frame to the right, move 5 frames downward, etc.).
NOTE: The first frame is frame 0, not frame 1.

```
imageRect.x = frameWidth * 2;
```

Fig 5. This will display frame 3 in the sheet we're using

Step 4: Display the frame

Displaying a frame of a sprite sheet is handled in a similar way as displaying an image, the only difference is you need to pass in the rect along with the sprite sheet, and any transform you want to apply to the sprite (scale, position, etc.)

```
SDL_RenderClear(renderer);
SDL_RenderCopy(renderer, image2, &imageRect, GetRectDisplayInfo(imagePos));
SDL_RenderPresent(renderer);
```

Fig 6. How to display a frame of a sprite sheet

```
SDL_Rect *GetRectDisplayInfo(SDL_Rect &rect) {
    rect.x = rand() % (800 - 170);
    rect.y = rand() % (600 - 300);
    rect.w = 170;
    rect.h = 300;

    return &rect;
}
```

Fig 7. For this example, a frame will have a size of 170x300 and displayed at a random location on the screen

III. Clean up

Last, but most important, remember to always release the resources before quitting the application.

```
SDL_DestroyWindow(window);
SDL_DestroyTexture(image1);
SDL_DestroyTexture(image2);
SDL_DestroyRenderer(renderer);

window = nullptr;
image1 = nullptr;
image2 = nullptr;
renderer = nullptr;
SDL_Quit();
```

Fig 8. Releasing resources before quitting application

What we found:

- A sprite can be any size, but to match the hardware's constraints, it needs to be a square image with dimensions matching powers of 2. This means if a sprite wasn't that size to start with, it would be padded to match the constraints, in other words, become a waste of memory. By packing multiple sprites closely together into a sprite sheet, we can reduce this overhead.
- A GPU needs to know 3 things when drawing a sprite:
    - Which sprite to draw
    - Which part of the sprite to draw
    - Which part of on the screen to draw it on
- If there were many sprites to draw at the same time, it would drastically increase the draw time, because the same number of operations would need to be repeated for

each sprite. By using a sprite sheet, we only need to set the reference to one massive sprite, and if the sprite is laid out in a grid, figuring out which part of the sprite atlas to draw is much faster since the calculation for the sizes would've already been done before gameplay starts.

- What's not to do with sprite sheets is loading them during runtime, since it eats a massive chunk in loading time. If there's a need to do this, it’s recommended to load individual images to save loading time (if that’s your concern).