Spike: 9
Title: Sprites and Graphics

Author: Ryan Chessum, 102564760

Goals / deliverables:

- Code, see /17 – Spike – Sprites and Graphics /Task 17/
- Spike Report

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Visual Studio 2019
- C plus plus reference (https://www.cplusplus.com/reference/)
- SDL2 (https://www.libsdl.org/)
- SDL2 Image (https://www.libsdl.org/projects/SDL_image/)

Tasks undertaken:

- Download and install Visual Studio
- Create a new C++ project
- Download SDL development libraries
- Link SDL Library to project
- Download SDL2 Image development libraries
- Link SDL Image to project
- Draw background and spritesheet to use as images
- Load renderer
- Load the images as textures
- Set up inputs in game loop
- Set up rendering in game loop

What we found out:

SDL2 by itself does not support common image formats such as png and jpg. Much like with sound effects, we need to download an add on library called SDL2 image.

First you must download and link both development libraries for SDL2 and SDL image.

To make a simple program using images I first drew a few sprite images on ms paint. I made one for the background and another with 3 sprites in it to use as a sprite sheet.

1

2

3

Background

Image

Ryan C

102564760

To display our images on the screen we are going to use a Renderer which is a structure in SDL used for rendering things to the screen. Using this we can render SDL shapes to the screen.

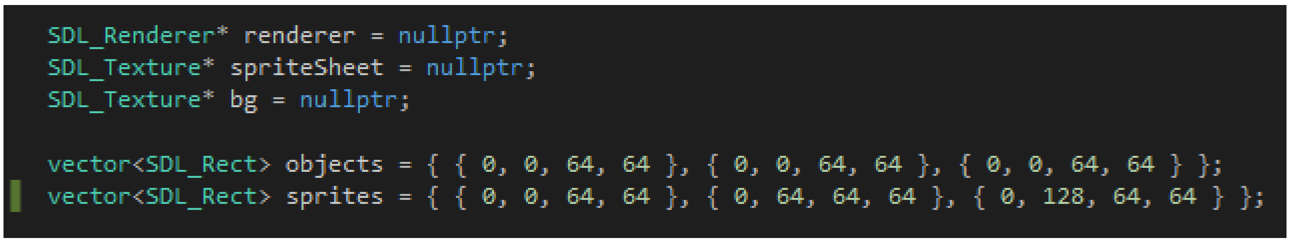
To load images we can use IMG_Load() and pass in a file directory. This will return a pointer to the image as a surface.

SDL_Rect defines rectangle shapes with an x and y coordinate aswell as a width and a height. These can be drawn to the screen by the renderer.

An SDL_Texture is pixel data. We can create a texture using a surface by calling SDL_CreateTextureFromSurface.

Then using a function called SDL_RenderCopy() we can pass in the renderer, a texture, a rect describing what part of the texture we want to use and the rect we want to render. This will render the rect with the part of the texture we specified on it to the screen.

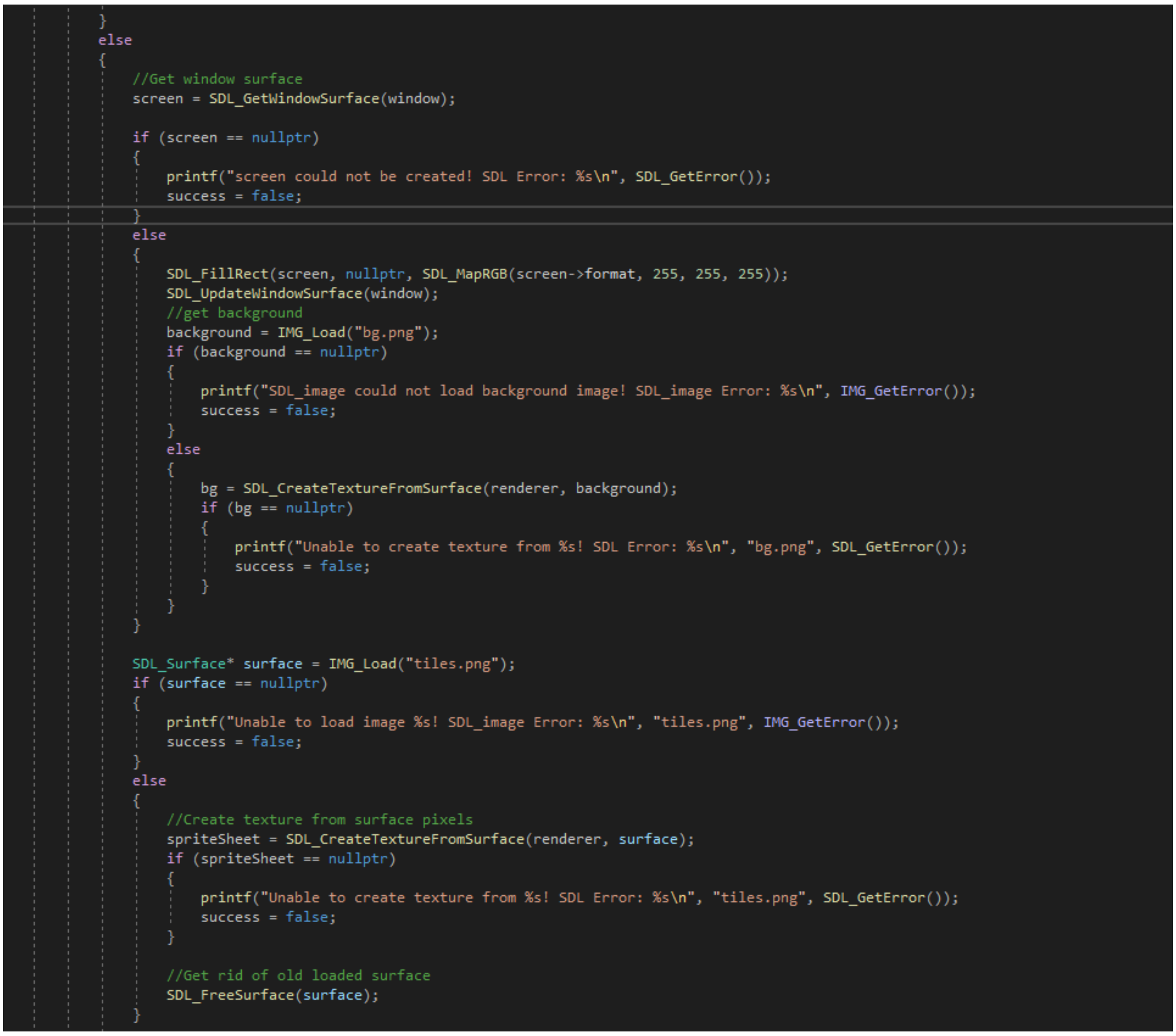
In my program I first set up the local variables to do this.



```
SDL_Renderer* renderer = nullptr;
SDL_Texture* spriteSheet = nullptr;
SDL_Texture* bg = nullptr;

vector<SDL_Rect> objects = { { 0, 0, 64, 64 }, { 0, 0, 64, 64 }, { 0, 0, 64, 64 } };
vector<SDL_Rect> sprites = { { 0, 0, 64, 64 }, { 0, 64, 64, 64 }, { 0, 128, 64, 64 } };
```

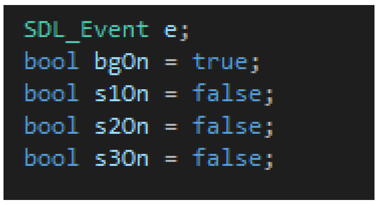
There is a renderer, two textures, one for each image and tow vectors holding rects. The objects rect vector is for showing the sprites on the screen. The other sprites rect vector holds the sizes for the sprite cuts in the second image.

When we initialiase we need to use SDL_CreateRenderer to create a renderer and set our renderer pointer to it.



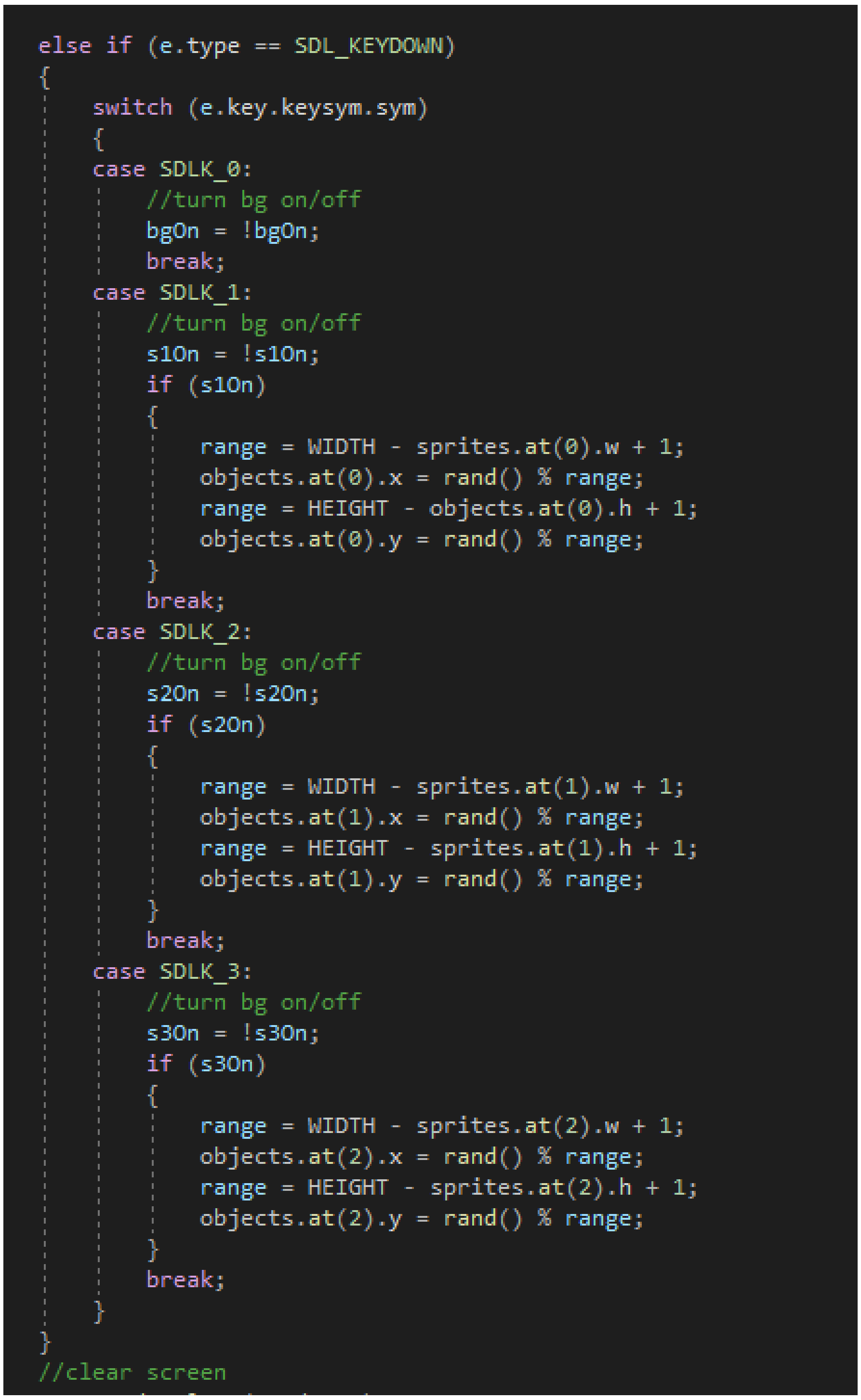
```
        }
        else
        {
            //Get window surface
            screen = SDL_GetWindowSurface(window);

            if (screen == nullptr)
            {
                printf("screen could not be created! SDL Error: %s\n", SDL_GetError());
                success = false;
            }
            else
            {
                SDL_FillRect(screen, nullptr, SDL_MapRGB(screen->format, 255, 255, 255));
                SDL_UpdateWindowSurface(window);
                //get background
                background = IMG_Load("bg.png");
                if (background == nullptr)
                {
                    printf("SDL_image could not load background image! SDL_image Error: %s\n", IMG_GetError());
                    success = false;
                }
                else
                {
                    bg = SDL_CreateTextureFromSurface(renderer, background);
                    if (bg == nullptr)
                    {
                        printf("Unable to create texture from %s! SDL Error: %s\n", "bg.png", SDL_GetError());
                        success = false;
                    }
                }
            }

            SDL_Surface* surface = IMG_Load("tiles.png");
            if (surface == nullptr)
            {
                printf("Unable to load image %s! SDL_image Error: %s\n", "tiles.png", IMG_GetError());
                success = false;
            }
            else
            {
                //Create texture from surface pixels
                spriteSheet = SDL_CreateTextureFromSurface(renderer, surface);
                if (spriteSheet == nullptr)
                {
                    printf("Unable to create texture from %s! SDL Error: %s\n", "tiles.png", SDL_GetError());
                    success = false;
                }

                //Get rid of old loaded surface
                SDL_FreeSurface(surface);
            }
```

Next, we initialise a screen for the white background behind the background image if we turn it off. Then we use IMG_Load() to get the background image and use the surface returned to use to create a texture using SDL_CreateTextureFromSurface(). We do the same for the spritesheet image.



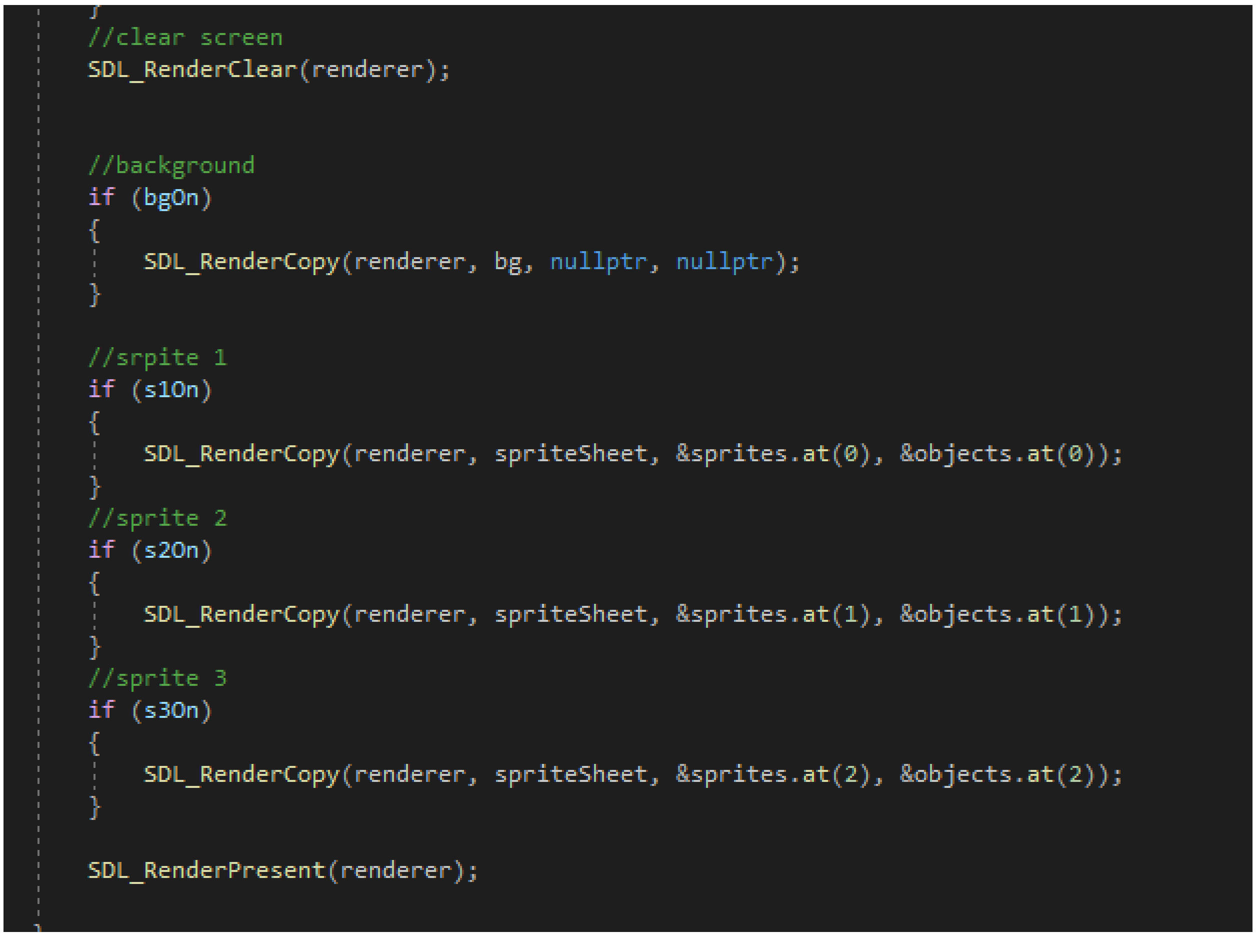
```
SDL_Event e;
bool bgOn = true;
bool s1On = false;
bool s2On = false;
bool s3On = false;
```

Before the gameloop I declare some Boolean variables to use for turning on/off the images.



```
else if (e.type == SDL_KEYDOWN)
{
    switch (e.key.keysym.sym)
    {
    case SDLK_0:
        //turn bg on/off
        bgOn = !bgOn;
        break;
    case SDLK_1:
        //turn bg on/off
        s1On = !s1On;
        if (s1On)
        {
            range = WIDTH - sprites.at(0).w + 1;
            objects.at(0).x = rand() % range;
            range = HEIGHT - objects.at(0).h + 1;
            objects.at(0).y = rand() % range;
        }
        break;
    case SDLK_2:
        //turn bg on/off
        s2On = !s2On;
        if (s2On)
        {
            range = WIDTH - sprites.at(1).w + 1;
            objects.at(1).x = rand() % range;
            range = HEIGHT - sprites.at(1).h + 1;
            objects.at(1).y = rand() % range;
        }
        break;
    case SDLK_3:
        //turn bg on/off
        s3On = !s3On;
        if (s3On)
        {
            range = WIDTH - sprites.at(2).w + 1;
            objects.at(2).x = rand() % range;
            range = HEIGHT - sprites.at(2).h + 1;
            objects.at(2).y = rand() % range;
        }
        break;
    }
}
//clear screen
```

In the event handler if a key is pressed, it switches the bool to the opposite value. For the sprites, if the bool has just been turned back on, the object rect is set to a random position.



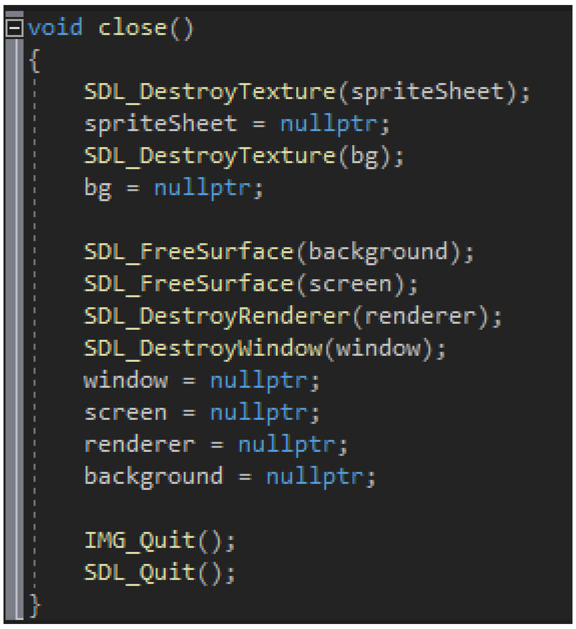
```
    //clear screen
    SDL_RenderClear(renderer);


    //background
    if (bgOn)
    {
        SDL_RenderCopy(renderer, bg, nullptr, nullptr);
    }

    //srpite 1
    if (s1On)
    {
        SDL_RenderCopy(renderer, spriteSheet, &sprites.at(0), &objects.at(0));
    }
    //sprite 2
    if (s2On)
    {
        SDL_RenderCopy(renderer, spriteSheet, &sprites.at(1), &objects.at(1));
    }
    //sprite 3
    if (s3On)
    {
        SDL_RenderCopy(renderer, spriteSheet, &sprites.at(2), &objects.at(2));
    }

    SDL_RenderPresent(renderer);
```

Then we render. First the renderer is cleared. Then for each image we want to display, if they are ‘on’ we call SDL_RenderCopy() and pass in the necessary parameters for each.

For the background, we want it to cover the whole screen and use the whole image. So we use nullptr instead of passing a rect defining the image size. It will just use the whole texture and fill the window.

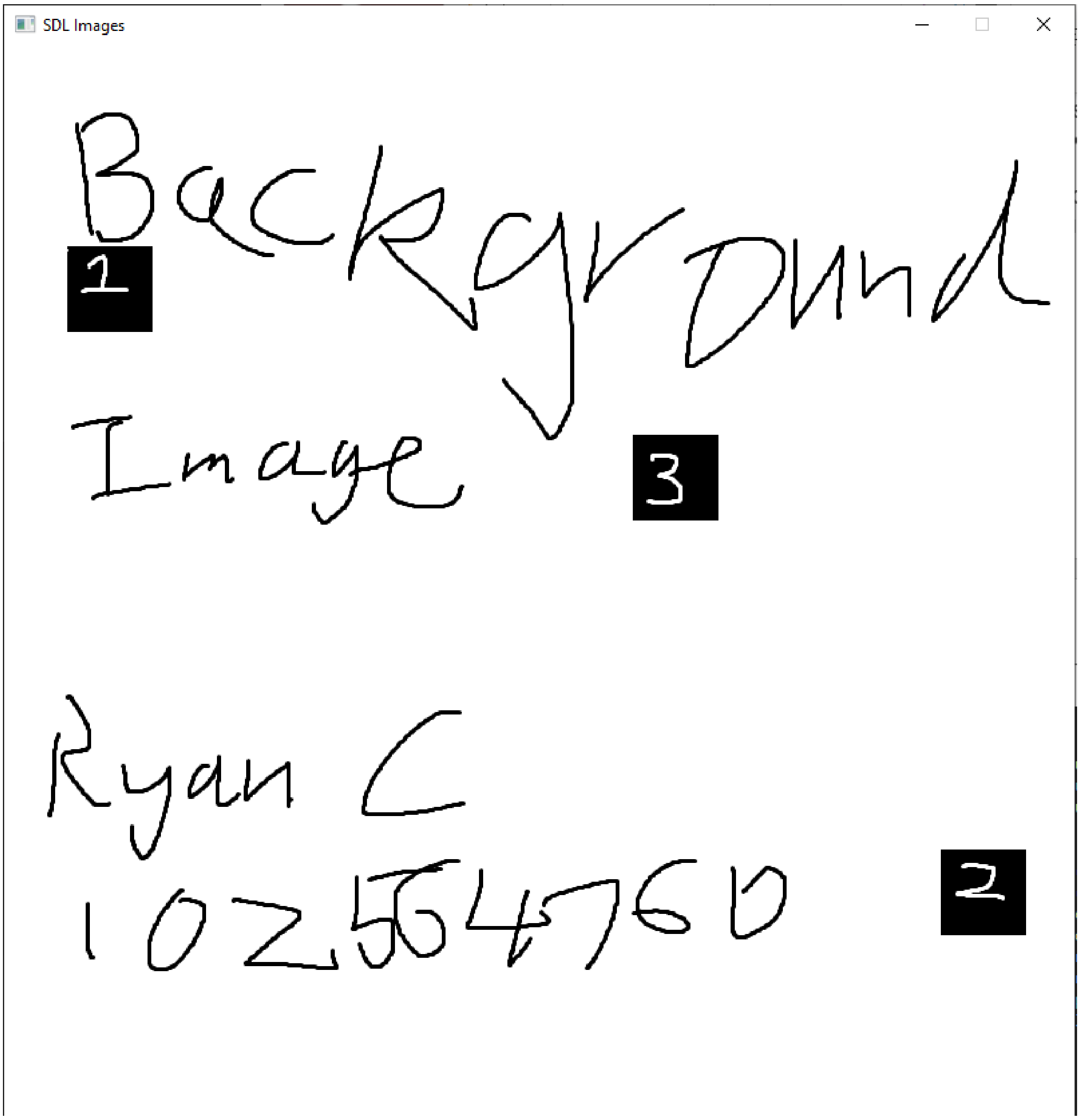
For the sprite sheet we pass in the spritesheet as well as the sprites rect and objects rect. The sprites rect will grab the part of the texture we want. Then that part of the texture will fill the objects rect.

Lastly at the end of the main function make sure to deallocate and destroy everything as usual.



```
void close()
{
    SDL_DestroyTexture(spriteSheet);
    spriteSheet = nullptr;
    SDL_DestroyTexture(bg);
    bg = nullptr;

    SDL_FreeSurface(background);
    SDL_FreeSurface(screen);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    window = nullptr;
    screen = nullptr;
    renderer = nullptr;
    background = nullptr;

    IMG_Quit();
    SDL_Quit();
}
```

Then the program should work.


SDL Images
Background
Image
1
3
Ryan C
102564760
2