LAB 09:
Collision maps + Complex Movement

Provided Files

- `main.c`
- `gba.c, gba.h`
- `mode0.c, mode0.h`
- `sprites.c, sprites.h`
- `game.c, game.h`
- `spritesheet.bmp, spritesheet.c, spritesheet.h`
- `collisionmap.bmp`
- `print.c, print.h`

Files to Edit / Add

- `main.c`
- `gba.c`
- `collisionmap.c, collisionmap.h`
- `Makefile`
- `.vscode`
    - `tasks.json`

Instructions

In this lab you will be completing several different instructions, which will add complex sprite movement and sprite animation. Your code may not compile until you complete an entire TODO block, at which point the game should compile with a new component of the final outcome.

TODO 1 - Sprite animation

We've already exported the spritesheet .c and .h file for you. Take a moment to look at `spritesheet.bmp`, and notice how sprites are laid out here. Once you're done, navigate to `main.c`.

TODO 1.0

- Inside `update()`, set the player's direction depending on which button was pressed.

TODO 1.1

- Still in `update()`, handle the animation if the player is moving.
    - This lab contains a generic `SPRITE` struct, but in the future, feel free to create one with the variables you need for your sprites. You can see the struct in `sprites.h`.
    - `player.isMoving` is a variable for tracking if any of the directional buttons were pressed. We are setting it to 0 at the beginning of update() and setting it to 1 if any of the directional buttons are pressed.
    - `player.timeUntilNextFrame` holds how much time (in game frames) we have until the next frame of animation. In this lab, your sprite should change animation frames once every 10 game loops (game frames). Therefore, `player.timeUntilNextFrame` is initialized to 10.
    - `player.frame` holds what frame of animation we're currently displaying.
    - `player.numOfFrames` holds how many total frames our animation has.
- We want to update how much time is left until our next frame.
    - Reminder: the variable is already initialized to 10!
- When it is time for the next frame, set the frame variable to the next frame and reset how much time is left to 10. Make sure that it loops once it has reached the last frame.

TODO 1.2

- Now, handle the animation if the player is *not* moving.
- The first thing we need to do here is set the player's frame to the first frame, so our player doesn't "freeze" looking like they're mid-step.
- We also need to reset the time we're waiting until the next frame, so that the next time the player moves it will start counting down from the beginning.

TODO 1.3

- At the bottom of update(), we need to finish setting up our sprite's shadowOAM.
    - For attribute 0, use `ATTR0_Y` to load the player's ***screen y position***.
    - For attribute 1, use `ATTR1_X` to load the player's ***screen x position***.
        - Note that for these two, the OAM wants the sprite's *screen position*. This is not necessarily the same as where the sprite is in the background.
        - Think about how the screen is mapped onto the background, and what variables we use to keep track of this!
    - For attribute 2, use `ATTR2_TILEID` to load the index of the top left tile of our sprite.
        - Since we're animating our sprite, this is going to change at runtime! `player.direction` and `player.currentFrame` will be useful here, along with `spritesheet.bmp`.

        - *Hint:* How many tiles on the spritesheet should we move in each direction when those variables change?
- Copying the shadowOAM into the OAM during vBlank is already handled for you in `draw()`.

Build and run your program. You should see the Brendan sprite, not animating, on the center of the map and the screen. While a directional button is pressed, it should animate through each of the frames for that direction, and then loop. The sprite should not move yet!

TODO 2 – Complex movement

In this section, we're going to make the screen follow the sprite around the map, centered when possible.

TODO 2.0

- In the button checks in `update()`, complete the code to move the player. The player should only move if their entire sprite will remain within the map.
    - The dimensions of the map (in pixels) are in the macros `MAPWIDTH` and `MAPHEIGHT`.

TODO 2.1

- Now that you've updated the sprite's x and y coordinates, use those to center the screen on the player.
    - Use the surrogate variables `hOff` and `vOff` to control the screen's position. We are already setting `REG_BG0HOFF` and `REG_BG0VOFF` to these in `draw()` for you.
    - The dimensions of the screen (in pixels) are in the macros `SCREENWIDTH` and `SCREENHEIGHT`.
- Don't forget to follow the cases where centering the screen of the player makes the screen show pixels beyond the edge of the map (and the background would wrap)!

Build and run your program. You should see the same thing as before but, now, when you press the directional buttons you can see the princess moving around the map. The screen should be centered on the sprite showing past the edges of the background.

TODO 3 – Collision map

In this section, we're going to add our collision map so that Brendan's movement will react to the environment.

TODO 3.0

- Boot up Usenti and export `collisionmap.bmp` as an 8bpp bitmap.
    - Make sure the color of the obstacles is in index 0, and the color of the free space is in index 1.

- At the top of `main.c`, include the .h from your export.

TODO 3.1

- Complete `colorAt()`. It's an inline function, you don't need to worry much about this – we're using it here for speed. If you would like to read more about inline functions read this: (link) This function takes in an `x` and `y`, and returns the palette index (`unsigned char`) at that location in the collision map bitmap.
    - It might be helpful to cast the `collisionmapBitmap` from Usenti as an unsigned char pointer, and then indexing into it using `OFFSET`.

TODO 3.2

- Edit the code you wrote for TODO 2.0. Now, only move the player if it is within bounds, and the location in the collision map where the sprite is attempting to move into is not using palette index 0.
    - For each direction, there are two corners of the sprite that we must check (check out the code comments). The position you should be checking is where the sprite is trying to move into, not where it currently is.
        - We have defined variables for the top, bottom, left, and right coordinates of the sprite. You don't need to use them, but it will help make your code more readable!
    - The `colorAt()` function returns 0 (false) when the palette index at the passed in location is 0. It returns true for all other palette indices.

Build and run your program. Your completed version should behave exactly like the `Example.gba` provided.

Submission Instructions

Ensure that **cleaning** and building/running your project still gives the expected results. If you are reading this, nice job. By now you probably ignore this section.You get a cookie.

Zip up your entire project folder, including all source files, the Makefile, and everything produced during compilation **(including the .gba file)**. Submit this zip on Canvas. Name your submission Lab08_LastnameFirstname. For example:

> “Lab09_PokemonKing.zip”

It is *your* responsibility to ensure that:

- The ZIP file is named correctly.
- All the appropriate files have been submitted.
- Your submitted ZIP can be opened and everything cleans, builds, and runs.