LAB 10:
Sound

Provided Files

- Makefile
- tasks.json
- main.c
- myLib.c
- myLib.h
- game.c
- game.h
- lose.c
- lose.h
- start.c
- start.h
- sound.c
- sound.h
- spritesheet.c
- spritesheet.h
- originalMP3s
    - gameSong.mp3
    - loseSong.mp3
    - sfxSound.mp3
    - startSong.mp3

Files to Edit/Add

- main.c
- gameSong.wav
- gameSong.c
- gameSong.h
- loseSong.wav
- loseSong.c
- loseSong.h
- sfxSound.wav
- sfxSound.c
- sfxSound.h
- startSound.wav

- startSong.c
- startSong.h
- sound.c
- .vscode
    - tasks.json

Instructions

In this lab, you will be modifying the attached project to play sounds correctly. You will be editing the code that pauses, stops, and loops sound. *Please make sure to check the recitation video from 4/6 for a tutorial on how to convert .mp3 to .wav files.*

TODO 1 - Convert sound files

You will need to convert the attached sound files to be the correct .wav format using Audacity. Then, you will convert the .wav files to .c and .h files by building your project with the new Makefile and tasks.json file.

- **Formating the .wav file**
  Please note that these steps may slightly vary for different versions of Audacity.
    - Open .wav audio files in Original folder in Audacity
    - Select the track
    - Select Tracks -> Stereo Track to Mono
    - Set the Project Rate (Hz) in the lower-left corner of the screen to 11025
    - Go to Tracks -> Resample and choose 11025 as the new sample rate
    - Select File -> Export Audio
    - Be sure to save it in your lab folder where the rest of your .c and .h files are
    - Under the “Save as type” dropdown, select Other uncompressed files
        - If there is a button for “options,” press the button
        - You should see dropdown menus for “Header” and “Encoding”
        - Set Header to “WAV (Microsoft)”
        - Set Encoding to “Unsigned 8-bit PCM”
        - Type “.wav” after your file name (for example, gameSong.wav)
        - When the Edit Metadata screen pops up, press the Clear button then hit OK

- **Converting the .wav file**
    - Ensure you’re using the new Makefile and tasks.json provided in the lab zip. **MOVING FORWARD, you will need to use the new Makefile and**

tasks.json in ALL FUTURE ASSIGNMENTS.

- As in Lab00, update your tasks.json on line 9 to include your emulator exact path.
  - This will be the same path you have used for previous projects
- Given that your .wav files you created from the previous step are in the SAME directory as your Makefile and other .c files, BUILD YOUR PROJECT NOW. It will not actually compile, which is fine, but you should see .c and .h files created for your .wav files once the build command finished.

- TODO 1.1
  - Include the sound .h files produced in the previous step at the top of main.c.
  - Build your project. No sound should be playing (since we haven't implemented that yet), but you should see Rosie from Animal Crossing taking a nap on a bench :)
    - If you run into any issues when you build your project, you can "clean" your project, then build again. To do this in VSCode, go to Terminal > Run Task and select "clean." Then, build as normal.

TODO 2 - Complete the playSound functions

- TODO 2.1
  - Navigate to the playSoundA function in sound.c. This is the function we will use to tell our game to start playing a sound, so let's complete it.
  - Assign all of soundA's appropriate struct values.
    - soundA is declared in sound.h. You can find the SOUND struct in myLib.h.
    - data, length, and loops are assigned values passed in through the function.
    - isPlaying should be equivalent to "true," since when we call this function we want a sound to start playing.
    - The duration formula is `((VBLANK_FREQ*length) / SOUND_FREQ)`
    - We can ignore priority, as it is not necessary for this lab.
    - vBlankCount should start at 0.

- TODO 2.2
  - Complete playSoundB with the same logic as playSoundA (these functions will be almost identical).
  - At this point, your code should build, but you won't hear any sound.

TODO 3: Complete the interrupt handler

- TODO 3.1
    - Navigate to the setupInterrupts function in sound.c. Set up the interrupt handler register. The interrupt handler register is a macro that can be found in myLib.h and should point to the interruptHandler *function*.

- TODO 3.2
    - Handle soundA playing in the interruptHandler function.
        - This is where we want to determine if we need to stop playing soundA or not.
        - First, increment the sound's vBlankCount.
        - Then, if the sound's vBlankCount is greater than the song's duration, we know we've reached the end of the song. You need to handle two cases here:
            - If the sound loops, restart the song.
            - If the sound does not loop, set the sound playing to false, turn off the DMA channel the sound is using, and turn off the timer the sound is using. Looking at the playSoundA function will be helpful here.

- TODO 3.3
    - Handle soundB playing in the interruptHandler function. This will be extremely similar to TODO 3.2.

- TODO 3.4
    - Call the two setup functions for sounds and interrupts in main.c. At this point, your code should build and start playing and looping music on the start screen! If it does not, take a closer look at your sound and interrupt functions.
    - Wait for the start song to loop and make sure you do not hear "Cornerface screaming at you", aka your game playing random bits of memory that will make a screeching noise. This means you are not correctly stopping your sounds in the interruptHandler function.

TODO 4 - Pausing, unpausing, and stopping music

We want to be able to control when our music plays and when it doesn't, so we have three functions to handle this in sound.c.

- TODO 4.1
    - Complete the pauseSound function. To pause a sound, we want to set soundA

and soundB playing to false and stop their timers.

- TODO 4.2
    - Complete the unpauseSound function. To unpause a sound, we want to reverse the changes made in TODO 4.1.
        - HINT: check out the timer flags in myLib.h.

- TODO 4.3
    - Complete the stopSound function. Completely stop soundA and soundB. This should be very similar to some of the code you wrote in TODO 3.2 and TODO 3.3.
    - At this point your code should build, but as we have not called any of these functions yet, you will not notice any changes.

TODO 5 - Set up more sounds!

Now that we have everything in place to play sounds, we can add more to our game.

- TODO 5.1
    - Play the gameSong music when the player presses START to transition from the start to game state.
        - Call stopSounds first, in case any songs are currently playing.
        - Make sure that the gameSong loops!

- TODO 5.2
    - Pause the music when transitioning from game to pause screen.
        - HINT: We wrote a function that does exactly this.

- TODO 5.3
    - Play the sfxSound and loseSong in goToLose
        - Call stopSounds first, in case any songs are currently playing.
        - Make sure that the loseSong loops and that the sfxSound do NOT loop!
        - Make sure that the sfxSound does not interrupt the loseSong (should play on a different channel)

- TODO 5.4
    - Unpause the music when transitioning from pause to game screen.
        - Hint: We wrote another function that also does exactly this.

At this point, you should be done! Check that your lab meets all of the requirements below.

You will know if it runs correctly if you:

- Have the following behavior in each state:
    - START
        - A looping start song plays
    - GAME
        - A looping game song plays
        - The game song is paused when going to pause screen
        - The game song is stopped sound when going back to the start screen and the start song begins to play
    - PAUSE
        - Sprites from the game state are hidden
        - The game song is unpaused when returning to the game screen
    - LOSE
        - A looping lose song plays
        - A NON-looping sfx sound plays
        - Both sounds are stopped when going back to the start screen and the start song begins playing

Tips

- Review recitation materials: Canvas > Recitation Materials
- Review the videos on how to convert .mp3 files to .wav files: Canvas > Pages > Lecture/Recitation Recordings (Monday Lecture and Tuesday Recitation)

Submission Instructions

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 Lab10_FirstameLastname, for example: “Lab10_DaisyChain.zip”.