LAB 10:

Tilemap Editing and Sound

Files to Edit/Add

- main.c
- game.c
- gameSong.wav
- gameSong.c
- gameSong.h
- loseSong.wav
- loseSong.c
- loseSong.h
- winSong.wav
- winSong.c
- winSong.h
- rowClearSound.wav
- rowClearSound.c
- rowClearSound.h
- startSound.wav
- startSong.c
- startSong.h
- sound.c
- .vscode
    - tasks.json

Instructions

In this lab, we give you a nearly complete version of *Tetris* and leave you in charge of two things:

- Managing the background tiles to place, check for, and clear tetriminos
- Adding sound for the start, game, and lose states, as well as when a row of tetriminos is cleared

To support this, you will need to edit sound.c to create some basic sound functions like playing, pausing, and looping sound tracks.

TODO 0 — Familiarize Yourself With the Lab

Before you make modifications to the provided code, skim through it briefly and see how things are set up. You do not need to understand how all of the code works, but you should have some idea how it is structured.

While you're at it, play the completed version (Example.gba) and see how the game should look once you are finished.

Note: If you struggle to actually clear rows in Tetris, check the game function in main.c. You will see that we have set up controls for you to go to the win and lose state by pressing the left and right trigger buttons. In addition, once you get to the sound portion of the lab, you can add a call to playSound to play the row clear sound when select is pressed.

TODO 1 — Tilemap Modification

TODO 1.0

- In game.c, complete the checkIfRowIsFilled function.
    - This function should return 1 is the row is filled with tetriminos and 0 otherwise.

TODO 1.1

- In game.c, complete the clearRow function.
    - This function takes in a row index. It then moves up each row of the board and sets it to the row beneath it.
        - Since each row gets overwritten by the row above it, we effectively erase the passed in row and move all other rows down to replace it.

TODO 1.2

- In game.c, complete the addPiecesToBackground function.
    - This function should iterate through the 4x4 area of the tetrimino and, if that index of the tetrimino collision map is not 0, sets the corresponding tile in gameboard to the same index at that spot in the collision map.
        - You only need to do the actual setting of the tilemap entries. The provided code handles iterating through the area and checking for non-zero entries.
        - Use currentTetreminoRow and currentTetreminoCol for the tile coordinates of the top left corner of the tetremino we're moving to the background.
        - You don't need to worry about how the tetremino collision maps are setup, though feel free to look in collision_maps.c if you'd like. To get a particular entry of the current tetremino, use:

          ```
          collisionMaps[(currentTetreminoType * 4)
          + currentTetreminoRotation][4 * j + i];
          ```

          where j and i are the row and col within the tetremino, respectively.
        - The corresponding row in the gameboard is currentTetreminoRow + j. The corresponding col is similar.

TODO 2 - 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.

TODO 2.0

Please note that these steps may slightly vary for different versions of Audacity.

- Exporting the .wav file
  - Open .wav audio files in “OriginalTracks” folder in Audacity.
  - Select the track.
  - Select Tracks -> Mix -> Mix Stereo Down 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 -> 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 “WAV (Microsoft).
    - If there is a button for “options,” press the button.
    - You should see a dropdown menu for “Encoding”.
    - 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***.

TODO 2.1

- Include the sound .h files produced in the previous step at the top of main.c and game.c.
  - game.c only needs rowClearSound.h, main.c needs all of the others
- Build your project. No sound should be playing (since we haven't implemented that yet), but you should be able to play game like before.

- 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 3 - Complete the playSound functions

TODO 3.1

- 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 3.2

- 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 4: Complete the interrupt handler

TODO 4.1

- 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 tetrisLib.h and should point to the interruptHandler function.

TODO 4.2

- 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 4.3

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

TODO 4.4

- Call the two setup functions for sounds and interrupts in main.c. Also in main.c, uncomment the line in goToStart to actually play the start song. 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 5 - 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 5.1

- Complete the pauseSound function. To pause a sound, we want to set soundA and soundB playing to false and stop their timers.

TODO 5.2

- 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 5.3

- 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 6 - Set up more sounds!

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

TODO 6.1

- 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 6.2

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

TODO 6.3

- Play the loseSong in goToLose and winSong in goToWin
    - Call stopSounds first in case any songs are currently playing.

TODO 6.4

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

TODO 6.5

- Play the rowClearSound when a row of tetriminos is cleared.
- This sound should not loop.
- Make sure that the sfxSound does not interrupt the gameSong. (It should play on a different channel!)

At this point, you should be done! Check your lab against the Example.gba file provided!

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

Ensure that **cleaning** and building/running your project still gives the expected results. **Please reference the last page of Lab02.pdf for instructions on how to perform a "clean" command.**

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_LastnameFirstname, for example:

> “Lab10_FunIsTetris.zip”

It is your responsibility to ensure that all the appropriate files have been submitted, and that your submitted zip can be opened and everything cleans, builds, and runs as expected.