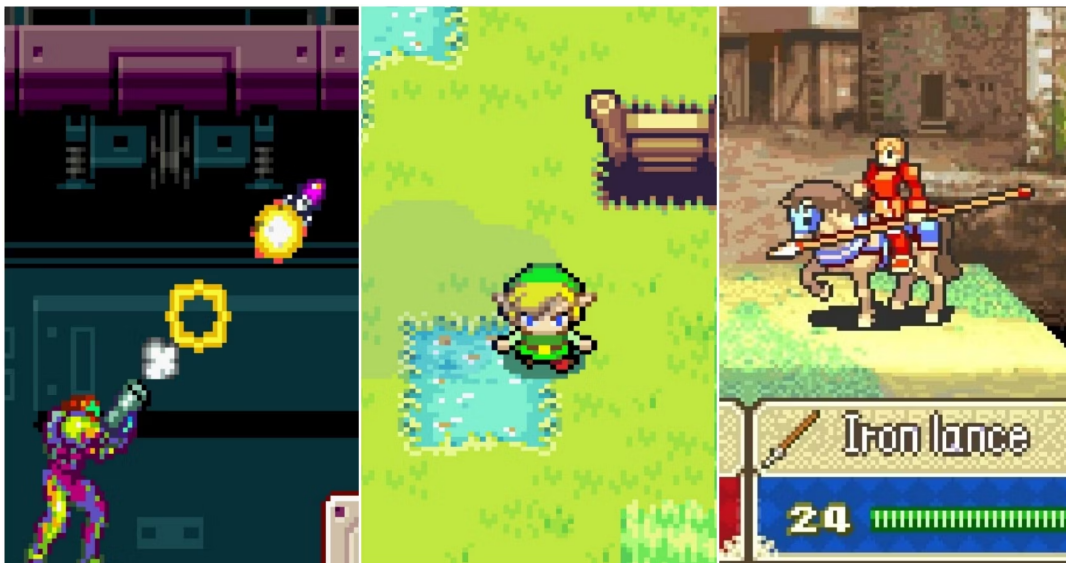




HOMEWORK 04:

Mode 4 Game



Purpose: To build a more complex game in Mode 4 to further your understanding of: analog sound, double buffering, palettes in Mode 4, text, DMA, and images in Mode 4.

Instructions

In this homework, you will create a complex game in Mode 4. This must be something different than what you implemented for a previous assignment or any of the labs; **you must create something new**. Also, you must use the provided scaffold as the basis for your homework.

Note on the scaffold: The scaffold released with this HW is purposefully incomplete, as Mode 4 functions are a part of this week's lab. After completing the lab, feel free to paste the content of the functions you wrote into the correct spots in the scaffold. We will also provide a completed version after the submissions for the lab have closed.

The complexity expectation is the same as the last homework assignment but, this time, images are required.



Requirements

Minimum Game Features

Your *game* must have the following:

- At least **two structs**.
- At least **one array**.
- **Object pooling**.
- A **state machine** including at least the following states:
 - Start,
 - Pause,
 - Game,
 - Win and/or Lose.
 - It is ok if your game is a survival based game, and therefore only has a lose state!
- You must be able to **navigate between the states** in some way (e.g. pressing the button START while on the Start state takes you to the Game state, winning the game while on Game states takes you to the Win state, etc.). Your game must also be **restartable** after reaching the Win/Lose state via transition back to the Start state.
 - If you have both Win and Lose, the game should be restartable from both states!
 - Using the emulator's reset button (ctrl/cmd + R) *does not meet this requirement*.
- **Meaningful text**.
 - It needs to be relevant to the game.
 - e.g. labeling states, or "lives:" or "score:" next to the non-static text for those, etc.
- **Non-static text**.
 - This means the text changes while you are looking at it (erased then redrawn).
 - e.g. score or lives that visibly update during the game, etc.
- At least **two types of moving objects**.
 - These must use two different structs. They must also *look and behave* differently (e.g. green player controlled by buttons and red enemy chasing player).
- **Collision that matters** (i.e. *something* must happen whenever two different objects touch each other – think bullet+enemy, enemy+player, tomato+face, etc.).
- At least **one non-fullscreen image**.
- At least **one fullscreen image**.



- e.g. a nice Start screen, like in Lab05!
- At least **three buttons** used for input.
- At least **two different sound effects using analog sound**.
 - These sound effects should occur *when something happens* (e.g. collision with enemies, shooting bullets, losing a life, etc.).
 - You must meaningfully use at least **two** of the analog sound channels.
 - Using channel 1 and channel 2 does not meet this requirement if you do not use sweep with channel 1, since they would have the same functionality.
- A **README.md** file.
 - An instruction manual (of sorts) that tells a player how to play your game, including things such as:
 - What each button does (controls),
 - How to navigate your state machine,
 - How to win and/or lose your game,
 - Which additional required mechanics you implemented and where to see them in your code
 - Anything that is buggy or not completed.
 - If you have little to no experience with Markdown syntax, here is a [cheat sheet](#) and a [more thorough explanation](#).
 - There are plenty of other resources online! Feel free to reference as many as you like.
 - You must use at **least one form of Markdown formatting** (header, bold, italics, etc.) and this use must make some sort of logical sense.
- **No flicker at all.**
 - Framerate should stay reasonably high (20fps or higher).

Additional Required Mechanics

You must implement **at least two** of the following mechanics in your game:

- At least **one game object/entity must be an image** *and* it must **switch between at least two images** at some point in the game.
 - For example, it could change images while the player has some sort of power-up (like when Mario collects the star or fire flower). Another example is the player hitting a block and switching to an image showing the block close to breaking.
- You must have a dodge mechanic for the player. If you press a button the player will enter a dodge state. We recommend using the **L/R shoulder buttons** for this. While in the dodge state, the player is invulnerable to any damage. The dodge must have a **cooldown** to prevent spamming it!



- For example, if a bullet is heading toward the player, they can dodge through the bullet and not take any damage.
- Hint: You can fulfill the first additional mechanic at the same time as this one by making a **visible** change to the player's sprite to indicate that the player is dodging.
- You must have some sort of **score counter** and **the high score must persist between restarts** (you must store and display at least the highest score, but you may choose to store and display multiple, such as the top 3, if you wish.) **The score must be displayed inside of its own state called Scoreboard.** This must be added to your state machine and must be **accessible from either the Start or the Pause state.**
 - *Note:* when you build and run again or re-open the emulator, your score will be cleared and that is fine! However, it must persist when we return to the START state.

You may implement all three of these mechanics to receive **up to 10pts of extra credit** to your grade for this assignment.

Code/files

Your *code* must have the following:

- Be entirely written in **Mode 4**.
- At least **six .c files**.
 - The scaffold contains `main.c`, `gba.c`, `mode4.c`, `font.c`, and `print.c`. This means you have to create at least one more source file for this assignment.
- At least **five .h files**.
 - The scaffold contains `gba.c`, `mode4.c`, `font.c`, and `print.c`. This means you have to create at least one more header file for this assignment.
- Good organization (see tips below).
- Meaningful comments.

Tips:

- **Start early.** Never underestimate how long it takes to make a game!
- **Start from the scaffold. Do not copy your last game's code and change it.**
- When splitting code between multiple files, put code specific to this game in `main.c` or other files. Those other files should be specific to a concept (response to collision, a specific state of the state machine, etc.).



- Do not draw text every frame. Text takes a while to draw, so only update it when it needs to be updated (e.g. only redraw score when it has changed from the previous frame).
 - Organize your code into functions specific to what that code does. **Your main function should not be very long at all!**
 - Having `update()` and `draw()` functions that you call directly in `main()` or within another function being called in `main()` is helpful.
 - Make sure the order of calling your functions takes into account waiting for `vBlank` at the correct times to minimize flicker.
 - Reference the Recitations and Lectures files on Canvas to review concepts, and feel free to reach out to Aaron or the TAs if you have any questions!
-

Submission Instructions:

Ensure that **cleaning** and building/running your project still gives the expected results.

Please reference previous assignments for instructions on how to perform a "clean" command if you need clarification.

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 **HW04_LastnameFirstname**, for example:

"HW04_PeachPrincess.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.