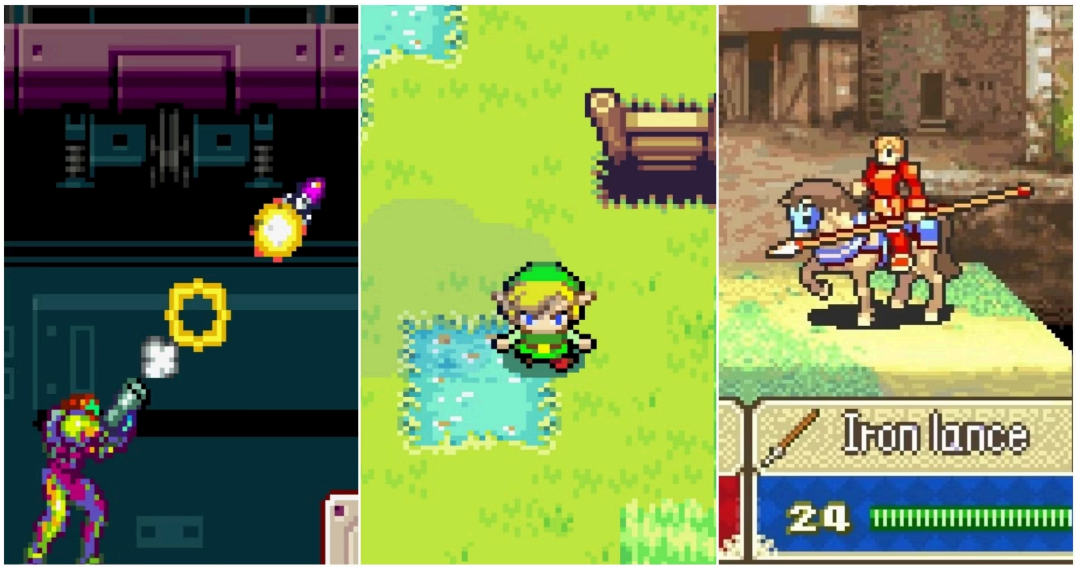
HOMEWORK 04:
Mode 4 Game



Purpose: To build a more complex game in Mode 4 to further your understanding of: 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.

<u>Note on the scaffold:</u> 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.

Base 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 Lab06!
- At least **three buttons** used for input.
- 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)
            - Make sure you're using **GBA buttons** to describe your controls instead of keyboard inputs!
        - 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).

Code/files

Your *code* must have the following:

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

Extra Credit Considerations

Make sure to include any extra credit considerations you've implemented in your README.md file!

Animation

You can earn up to 3 points of extra credit by implementing this feature!

- The animation must have **at least three frames**, each of which should be a separate image (not a rectangle!).
- The animation does not have to loop to receive credit.

Analog Sound

You can earn up to 2 points of extra credit by implementing this feature!

- The HW04 scaffold includes a set of files called `analogSound.c` and `analogSound.h`, which contains a library of sound effects you can use in your project!
- These sound effects should occur **when something happens** (collision with enemies, shooting bullets, losing a life, etc.).
- You must meaningfully use at least **two** of the analog sound functions.

Above and Beyond Gameplay Mechanic

You can earn up to 5 points of extra credit by implementing **any one** of these extra mechanics in your game (you don't need to do them all to earn all 5!):

- *Palette animation*. **Modify the palette during runtime** so that at least one of the colors in your palette cycles at a consistent rate between **at least two different RGB values**. This color-changing state should be triggered by a specific in-game event (taking damage, picking up a collectable, etc.) and make sense within the context of your game. Do **not** use DMA or change the palette indices stored in individual pixels to fulfill this requirement.
- *A high score system*. Your game should be able to keep track of your score throughout the game, and the high score for your game **should persist between runs** and be shown in **a separate SCOREBOARD state** in your state machine. You can show the high score in any state as long as it's also in the SCOREBOARD state.
- *Random seeding*. Some element(s) of your game should be randomized using the `rand()` and `srand()` functions in the standard C library. This randomness should be observable in your gameplay and **generate different results during separate playthroughs of your game**.

Alternatively, you can create your own additional gameplay mechanic **and** describe why you think it goes above and beyond the standard gameplay in your README.md to receive credit for this consideration.

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 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_MishimaKazuya.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.