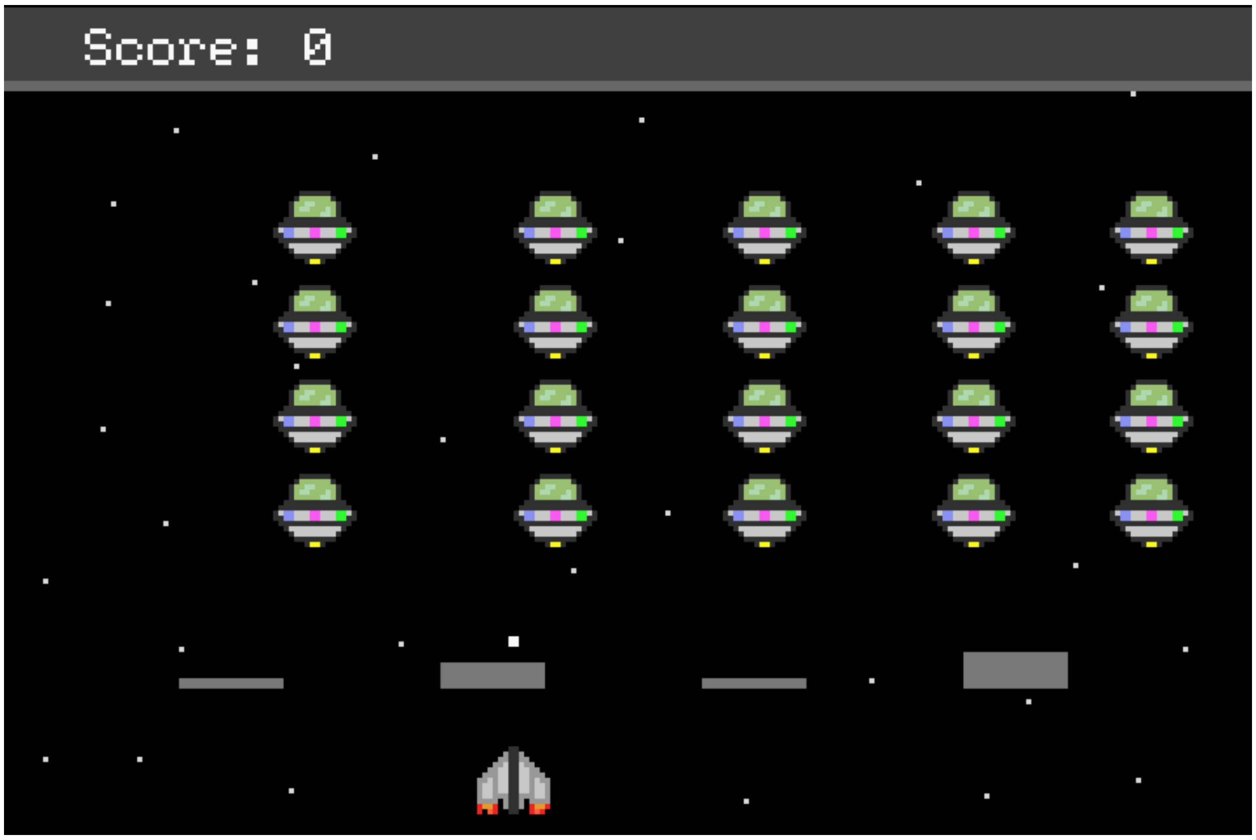
HOMEWORK 04

Mode 4 Game



Purpose: To build a 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:

For this homework, you will create a complex game in Mode 4. The complexity expectation is the same as the last homework assignment, but this time, images are required. You will need to complete Lab05 before this assignment so you can have the necessary functions for Mode 4. **Use the scaffold provided to you. You may NOT extend a game you already made for a previous assignment; you MUST create something new. You also may not extend a previous lab to fit the requirements of this homework.**

You are free to choose one of the examples we provide, but you are also free to create your own original game (*if you choose this option, please speak with a TA first* so that we can ensure it is on the expected difficulty level).

Requirements:

Your *game* must have the following:

- At least **one struct**
- At least **one array**
- **Pooling**
- Meaningful **text**
    - It needs to be relevant to the game
- **Non-static text**
    - This means the text changes while you are looking at it (erased then redrawn)
        - E.g. score that visibly updates during the game
- 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!
- **DMA used correctly for fillScreen4(), drawRect4(), drawImage4(), and drawFullscreenImage4()**
- At least **one non-fullscreen image**
- At least **one fullscreen image**
- At least **five moving objects**
- 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.)
- **Collision** that matters (i.e. *something* must happen whenever two different objects hit each other)
- 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
        - Anything that is buggy or not completed
    - You should have experience with Markdown syntax from HW03, but feel free to reference that pdf for more info!
- **No flicker**

Your *code* must have the following:

- **Be entirely written in Mode 4**
    - Having the Mode 3 functions alongside the others in HW04Lib.c is fine, as long as you **never** call them.
- **Multiple .c** files (more than just main.c and HW04Lib.c)
- At least **two .h files**
- Good organization (see tips below)
- Meaningful comments

Examples of games at the complexity level we expect:

- Tanks
- Space invaders (at least 3 rows of 5 blocks/aliens)
- Simple flash games (ex. http://www.ferryhalim.com/orisinal/)
- Simple Neopets games (http://www.neopets.com/games/)
- Old Atari games, like Asteroids (http://www.freeasteroids.org)

Tips:

- **Start early**. Never underestimate how long it takes to make a game!
- **Start from the scaffold or from scratch. Do not copy your last game's code and change it**.
- Do not draw text every frame. Text takes a while to draw, so only update it when it needs to be updated (ex: only redraw score when it has changed from the previous frame).
- When splitting code between multiple files, put code that will be useful in multiple games in HW04Lib.c, and code specific to this game in main.c or other files. Those other files should be specific to a concept (collision, etc.).
- Organize your code into functions specific to what that code does. **Your main method should not be very long.**
    - Having update() and draw() functions that you call in main() is helpful.
    - Make sure the order takes into account waiting for vblank at the correct times to minimize flicker.

- 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_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.