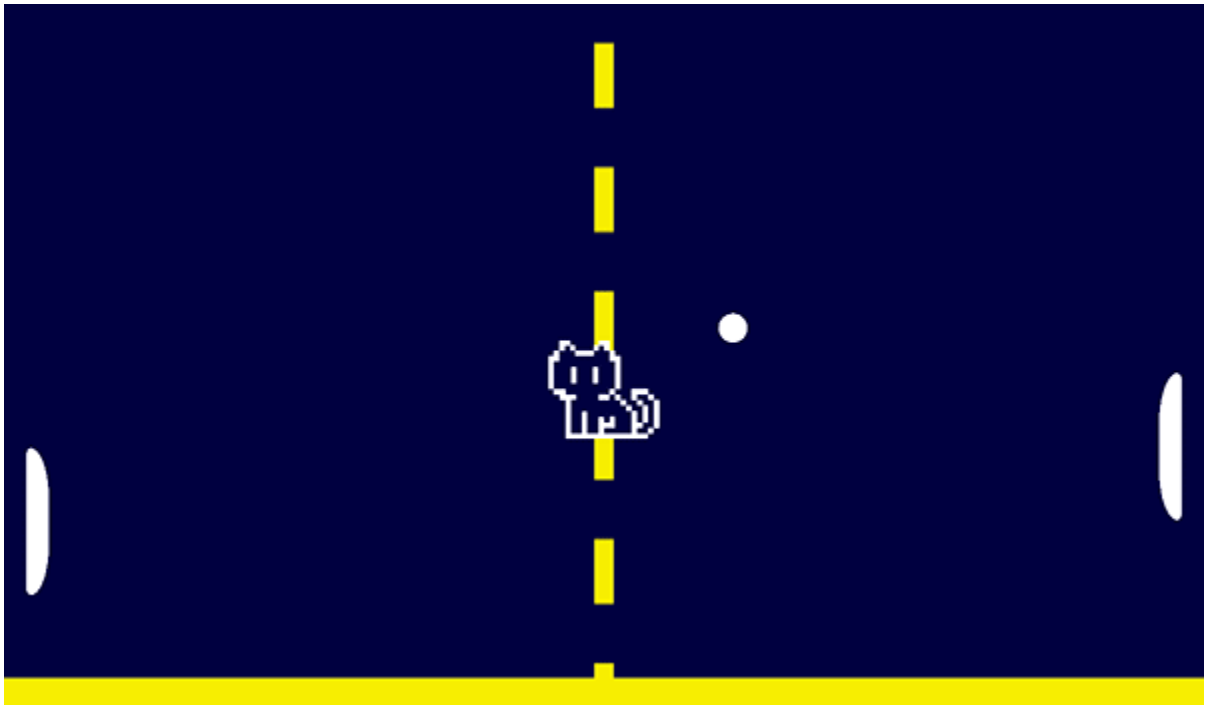
HOMEWORK 02:
Mode 3 – Pong



Purpose: To build a simple game in Mode 3 to further your understanding of: inputs to the GBA, collision detection and reaction, C basics, and drawing in Mode 3.

Instructions:

For this homework, you will recreate the game *Pong* (with a twist) in Mode 3. Review the **Requirements** section of this document for an explicit list of what we expect as the base requirements. The basic idea of original *Pong* is as follows:

There are two different paddles on opposite sides of the screen that can be controlled through user input. A ball moves around the screen and bounces off of the paddles. If a player misses the ball, the opposing player gets a point. The game ends when a player reaches 3 points, and that player wins.

Moreover, you are encouraged to be creative! Go outside of the requirements to add some flair to your game. **You will receive a maximum of 95 out of 100 points for meeting all of the base requirements. By adding your own flair, you *may* be awarded the 5 additional points needed to receive 100 points.**

Requirements:

Gameplay

Your *game* must have the following:

- Two paddles that can collide with the ball and move meaningfully through *intuitive* user input controls.
    - By intuitive, we mean something like buttons UP/DOWN for one paddle and A/B for the other.
- If the ball collides with a paddle or the top/bottom of the screen, it must bounce off and change direction.
    - **Note:** if you decide to change the orientation of the paddles to be at the top and bottom of the screen, then the ball should bounce off from the left and right sides of the screen.
- If the ball is missed by a paddle and hits the edge of the screen, the opposing player gets a point.
- When a paddle collides with the ball, the ball should split into two.
    - These balls should have all the behaviors that the original one had.
    - Another way to look at this is that, upon collision with the paddle, another ball is spawned at that location (where the collision occurred).
    - The two balls should be going in **opposite y directions**, meaning that they split once they move.
        - e.g. one ball goes up, the other goes down.
    - It's ok to have a max number of balls, as long as that number is higher than 10.
    - **Note:** balls do **not** have to interact with each other! The game does not need to detect two balls bouncing off each other, for example.
- The game should enter an end state once either of the two players reaches 3 points.
    - It must be clear that the game has ended, the player should not be able to play indefinitely. Just preventing the ball from moving or respawning is ok here.
- Only a minimal amount of flicker.

Code/files

Your *code* must have the following:

- Multiple `.c` files.

- At least one `.h` file.
- A **`readme.txt`** file.
    - This should contain an **instruction manual** that tells a player (but most importantly, your grading TA!) how to play your game.
        - *Include which buttons are used for controlling the paddles.*
- Good organization (see **Tips** section below).
- Meaningful comments.

Flair

Add *flair* to your game in order to receive the 5 points mentioned earlier. Some ideas:

- Add a tally to the screen to indicate how many points each player currently has.
- Draw the ball as something other than a rectangle/square.
- Change the end state to do something fun when a player wins.

Tips:

- **Start early.** Never underestimate how long it takes to make a game!
- For collision code, draw pictures. Graph paper is your friend.
- When splitting code between multiple files, put code that will be useful in multiple games in your lib file (e.g. `gba.h`), and code specific to this game in `main.c` or other files. Those other files should be specific to a concept (collision, movement, etc.).
- Organize your code into functions specific to what that code does. **Your `main()` should not be very long.** Use helper functions!
- Having `update()` and `draw()` functions that you call in `main()` is helpful.
- Make sure the order in which you call your functions takes into account waiting for vBlank at the correct times. This will help you minimize flicker.

Submission Instructions:

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

Zip up your entire project folder, including all source files, the Makefile, and everything

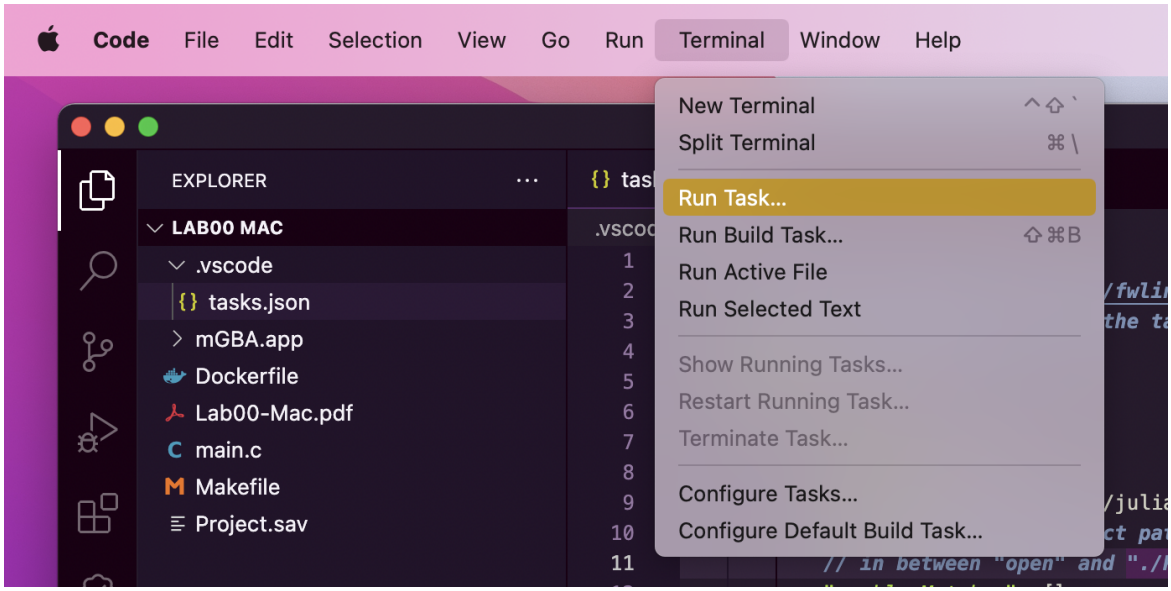
produced during compilation **(including the .gba file)**. Submit this zip on Canvas. Name your submission HW02_LastnameFirstname, for example:

> “HW02_TurnerTimmy.zip”

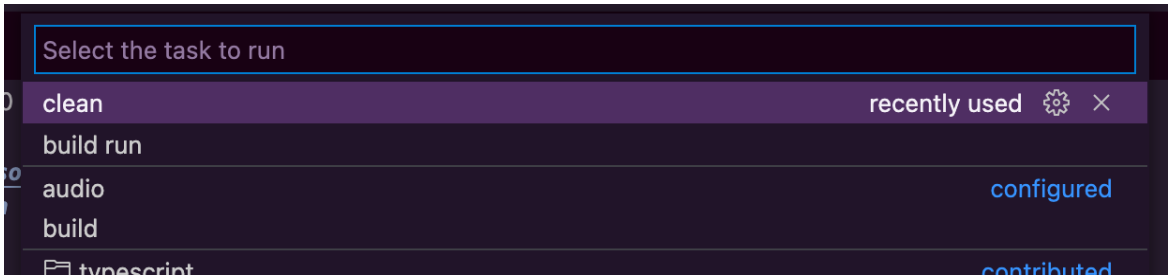
It is your responsibility to ensure that all the appropriate files have been submitted, and that your submitted zip can be opened and everything cleans, builds, and runs as expected.

How to Clean:

Navigate to the **Terminal** option at the top of your screen. Click on it, and then select **Run Task…**, as shown in the image below.



A dropdown menu will appear with the possible tasks to perform. Next, select **clean** from the options. Note that yours might be in a different order from mine, but the task should be there.



After selecting clean, something similar to the following should appear in your terminal tab (bottom of the screen).

If so, success! You have cleaned. You can now do **cmd/ctrl + shift + B** to build run (or click **Terminal**, then **Run Build Task…** or **Run Task…** and then **build run**).