LAB 03:

User Input & Collisions

Provided Files

- `Makefile`
- `print.h`
- `print.c`

Files to Edit/Add

- `.vscode`
    - `tasks.json`
- `main.c`
- `gba.c`
- `gba.h`

Instructions

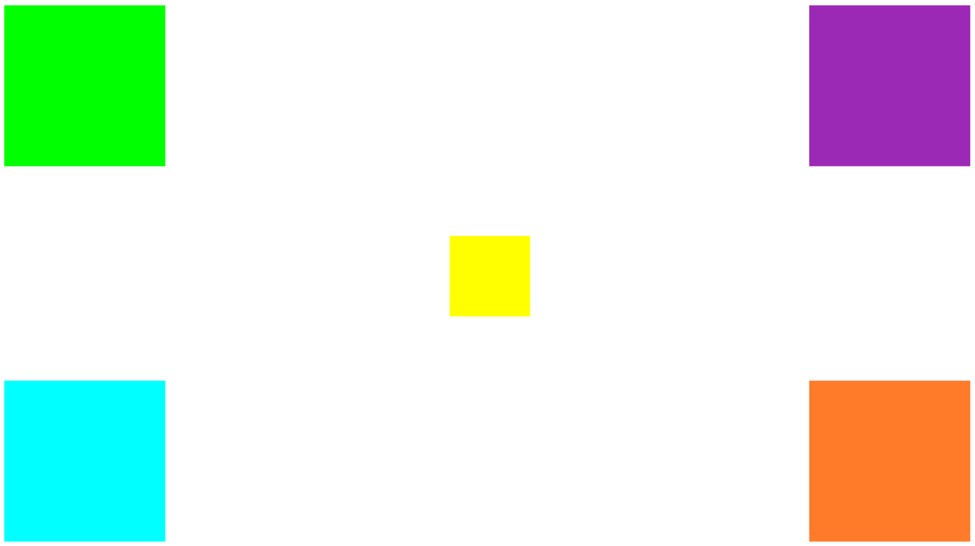
You will create a rectangle that has the ability to "paint" the screen.

TODO 1

TODO 1.0

- `void fillScreen(unsigned short color)`
    - In `gba.c`, find `fillScreen` (right beneath `drawRect`) and complete it.
    - You may only use a single loop (one for-loop) to complete this one.
        - **Hint:** the GBA screen has a total of 240 x 160 pixels which is equal to 38400.

Build and run your program. At this stage, your program should look like this:



TODO 2

This one requires a lot of moving parts to work, so it is broken into three parts.

TODO 2.0

- Open `gba.h`, find the **button macros**, and replace the 0 with the appropriate expressions.
- For `BUTTON_PRESSED`, assume that `buttons` and `oldButtons` have been set up appropriately.

TODO 2.1

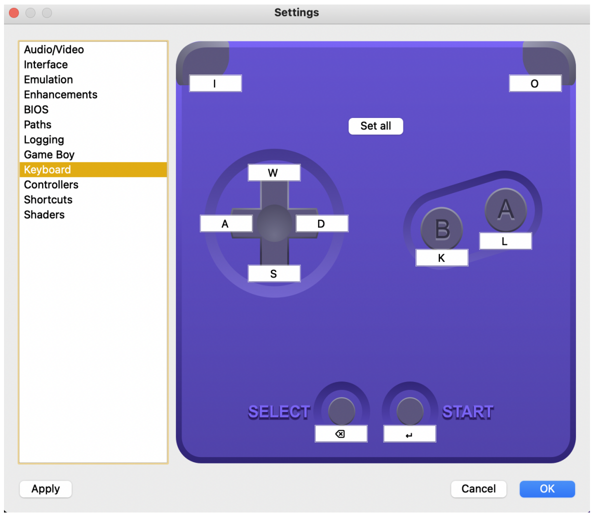
- Since `oldButtons` and `buttons` *have not* been set up correctly, head back to `main.c` and initialize them in the `initialize` function.
- They have already been declared in `main.c` (and in `gba.h` as extern variables), so you don't have to worry about that part this time, but you will when you code things yourself for your homeworks.

TODO 2.2

- The buttons still won't do anything unless you update `buttons` and `oldButtons` each frame, so do that in the main while-loop.

Run it, and now you can press button A or button B to toggle between yellow and black.

- **Note:** Everyone's emulated buttons can be mapped to different keys on their keyboard. To check what your GBA buttons are mapped to, go to the "Keyboard" tab in your mGBA settings/preferences.



Holding down the A or B button for a long time should have the same effect as just tapping it a single time. If that is not the case, you did one of these three TODOs incorrectly.

TODO 3

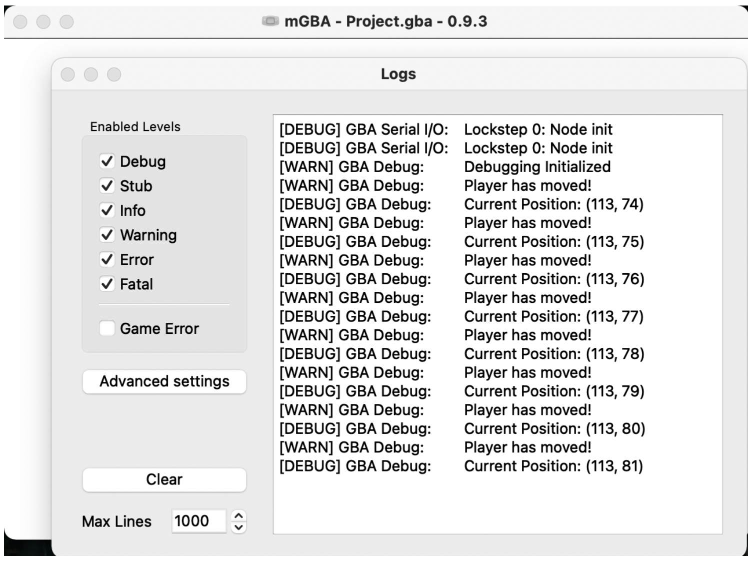
<u>TODO 3.0</u>

- Now that you can take button input, find TODO 3.0 in the `update` function and complete it so that the painter's rectangle can move up, down, left, and right if you hold the corresponding arrow key.
    - This time, it should move for **as long as the key is held**, meaning that if you tap it once quickly, it will barely move, but if you press it for several seconds, it will move a lot. If not, fix that.

Build and run, and you should be able to move the painter rectangle anywhere on the screen.

<u>TODO 3.1</u>

- Find TODO 3.1 in `update()`. When the player moves, use `mgba_printf_level()` to log “`Player has moved`” on the level Debug (`MGBA_LOG_DEBUG`). Then, use `mgba_printf()` to log their new position in the format “`Current Position: (x, y)`”. The default level is Warning.
- You must check the Debug box to see "Player has moved!". You should see that when you move your painting rectangle, the logs will update with the current position on the screen.



```
[DEBUG] GBA Serial I/O:  Lockstep 0: Node init
[DEBUG] GBA Serial I/O:  Lockstep 0: Node init
[WARN] GBA Debug:        Debugging Initialized
[WARN] GBA Debug:        Player has moved!
[DEBUG] GBA Debug:       Current Position: (113, 74)
[WARN] GBA Debug:        Player has moved!
[DEBUG] GBA Debug:       Current Position: (113, 75)
[WARN] GBA Debug:        Player has moved!
[DEBUG] GBA Debug:       Current Position: (113, 76)
[WARN] GBA Debug:        Player has moved!
[DEBUG] GBA Debug:       Current Position: (113, 77)
[WARN] GBA Debug:        Player has moved!
[DEBUG] GBA Debug:       Current Position: (113, 78)
[WARN] GBA Debug:        Player has moved!
[DEBUG] GBA Debug:       Current Position: (113, 79)
[WARN] GBA Debug:        Player has moved!
[DEBUG] GBA Debug:       Current Position: (113, 80)
[WARN] GBA Debug:        Player has moved!
[DEBUG] GBA Debug:       Current Position: (113, 81)
```

- Printing will be a useful tool for you throughout the course.

TODO 4

TODO 4.0

- At the bottom of `gba.c`, implement the `collision` function. It takes in the positions and dimensions of two rectangles, and returns a 1 if they are colliding, or a 0 if not.
    - **Hint:** using graph paper, or drawing a grid yourself, is extremely useful. This function should work regardless of the size or velocity of the rectangle (if one is moving).

TODO 4.1

- Now that you have `collision` implemented, use it in the `update()` function in `main.c`. You will need to check if the painter's rectangle has collided with any of the colorful rectangles. If the painter rectangle collides with any of the other colorful rectangles, the painter rectangle should become that color. For example, if the painter's rectangle hits the red rectangle, it should become red.
    - **Note:** After a collision, you should still be able to flip between two colors by pressing A or B. However, the two colors will be different from before since we're flipping the yellow bits with the XOR.
    - **Hint:** Since there are four separate rectangles for which you need to detect a collision, you'll need to use the collision function four separate times. You can find all the relevant variables for the color, height, width, x, and y of all of the colorful rectangles at the beginning of `main.c`.

TODO 4.2

- Now you will add the "painting" functionality. Find TODO 4.2 in the `draw()` function. Add a condition that stops the previous painter rectangle location from being erased if the select button is held.
    - **Hint:** Another way to think about this would be to only let the previous painter's rectangle location be erased if the select button is **not** held.
    - **Note:** If you are not holding select, the painter rectangle should "erase", or make any pixels it touches white. You should not be able to change the color of any of the original colorful rectangles.

Compile and run. Verify that it runs correctly by comparing to the example, and then you are done.

Checklist

Ensure that you can:

- Move the painter rectangle using the arrow keys.
- See player position update in the Logging window
- Press A or B to make the painter rectangle toggle between black and yellow if you have not collided with another color rectangle (otherwise it should simply toggle between two colors).
- Make the painter's rectangle collide with any of the colorful squares, causing the painter's rectangle to take the color of that square.
- Hold down the select button to make the painter rectangle "paint", or change the pixels it touches to the same color as the painter rectangle itself.
- Move the painter's rectangle without holding select to "erase" the colorful pixels it touches (other than the original colorful rectangles -- those should always stay the same).

To see what a completed version of the project looks like, see `Example.gba`.

Tips

- Start early, and attend recitation.
- Use graph paper and draw pictures to conceptualize what is happening.
    - This is especially helpful for collisions.
- Follow each TODO in order, and only move forward if everything is correct.

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 Lab03_LastnameFirstname. For example:

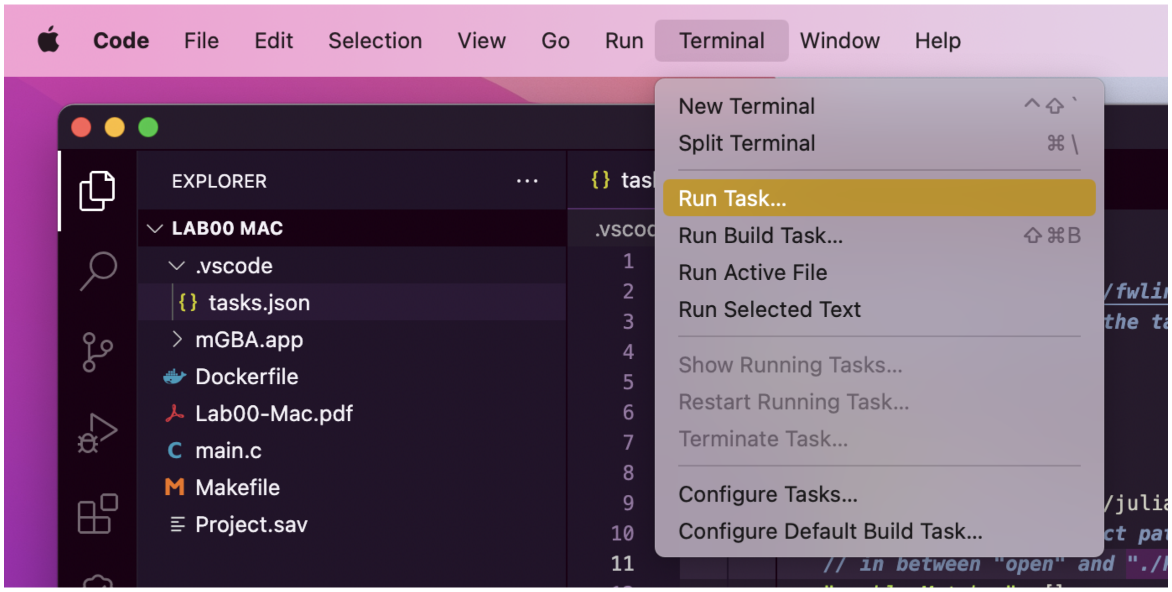
"Lab03_PicassoPablo.zip"

It is *your* responsibility to ensure that:

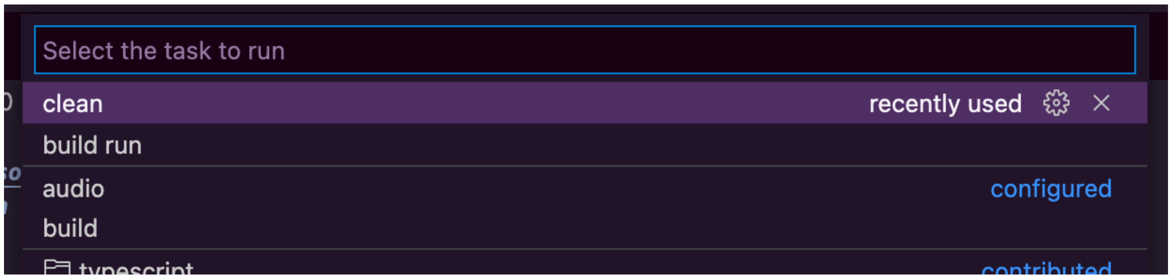
- The ZIP file is named correctly
- All the appropriate files have been submitted
- Your submitted ZIP can be opened and everything cleans, builds, and runs

How to Clean

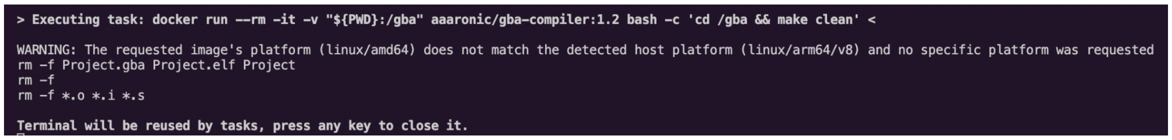
Navigate to the **Terminal** option at the top of your screen. Click on it, and then select **Run Task…**, as shown in the image below.



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).



```
> Executing task: docker run --rm -it -v "${PWD}:/gba" aaaronic/gba-compiler:1.2 bash -c 'cd /gba && make clean' <

WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested
rm -f Project.gba Project.elf Project
rm -f
rm -f *.o *.i *.s

Terminal will be reused by tasks, press any key to close it.
```

If so, success! You have cleaned. You can now do **cmd/ctrl + shift + B** to build and run.