



# LAB 02:

## Input and Collision

### Provided Files

- main.c
- lab2Lib.c
- lab2Lib.h
- print.c
- print.h

### Files to Edit/Add

- main.c
- lab2Lib.c
- lab2Lib.h
- Makefile
- .vscode
  - tasks.json

### Instructions

You will create a rectangle that has the ability to “paint” the screen! Note: Make sure to copy over your Makefile and .vscode/tasks.json from one of your previous assignments.

### TODO 1.0

- void drawRect(int col, int row)
  - Complete this function in lab2Lib.c (right under the setPixel function). You should already know how to do this from your last lab, and it should create a filled in rectangle.
  - In main.c, find and uncomment UNCOMMENT #1 and UNCOMMENT #2. Run your program. You should see a blue rectangle in the middle of the screen. This is your “painter” rectangle. There should be a magenta rectangle near the top left of the screen, a yellow rectangle near the top right, a cyan rectangle near the bottom left, and a green rectangle near the bottom right of



the screen. These are different paint colors you will be able to color the screen with -- essentially your color palette.

## TODO 2.0

- `void fillScreen(unsigned short color)`
  - Head back to `lab2Lib.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.
  - Enter `main.c` and uncomment `UNCOMMENT #3`. The background of your game should now be white.

## TODO 3.0-3.2

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

### TODO 3.0

- Open `lab2Lib.h`, find the button macros, and complete them.
- For `BUTTON_PRESSED`, assume that buttons and `oldButtons` have been set up appropriately.

### TODO 3.1

- Since buttons and `oldButtons` *haven't* been set up correctly (yet!), head on back to `main.c` and initialize them in the `initialize` function.
- They have already been declared in `main.c` (and in `lab2Lib.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 3.2

- The buttons still won't do anything unless you update them each frame, so do that in the main while-loop.
- After you have completed these tasks, find and uncomment `UNCOMMENT #4` in the `update` function.
- Run it, and now you can press button A to make the painter's rectangle black, and B to make it blue.
- 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.



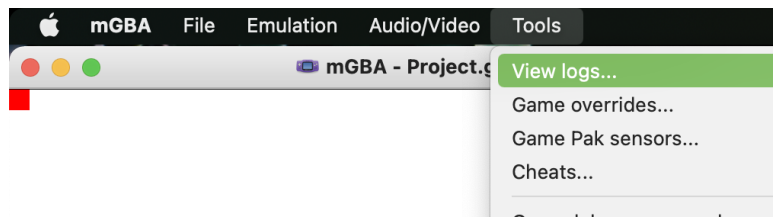
## TODO 4.0-4.1

### TODO 4.0

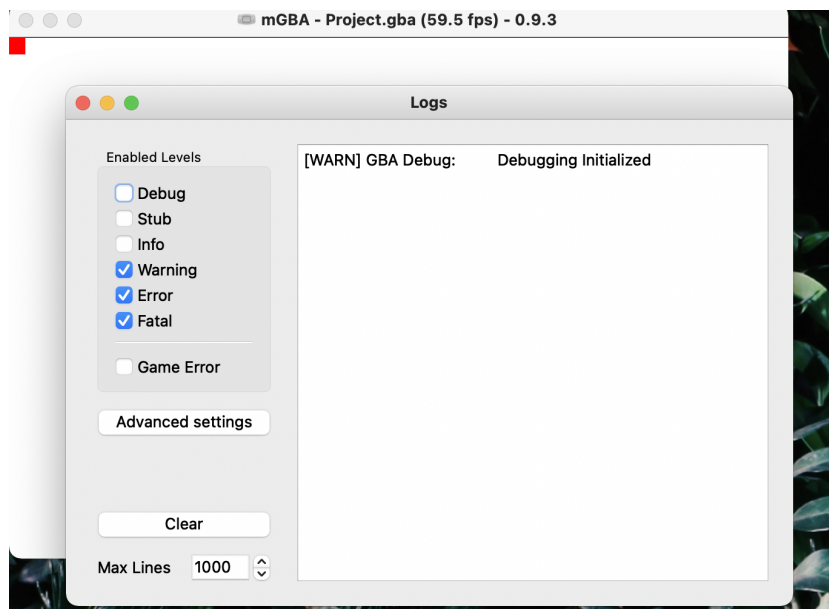
- Now that you can take button input, find TODO 4.0 in the update function and complete it so that the painter's rectangle can move up, down, left, and right if you hold the corresponding arrow key.
  - This time, it should move for as long as the key is held, meaning that if you tap it once quickly, it will barely move, but if you press it for several seconds, it will move a lot. If not, fix that.
- Compile and run, and you should be able to move the painter rectangle anywhere on the screen.

### TODO 4.1 - Console Logging / Debugging

- Uncomment UNCOMMENT #5 in “main” and “update” and compile and run.
- Once in your emulator, open “View Logs” under Tools:

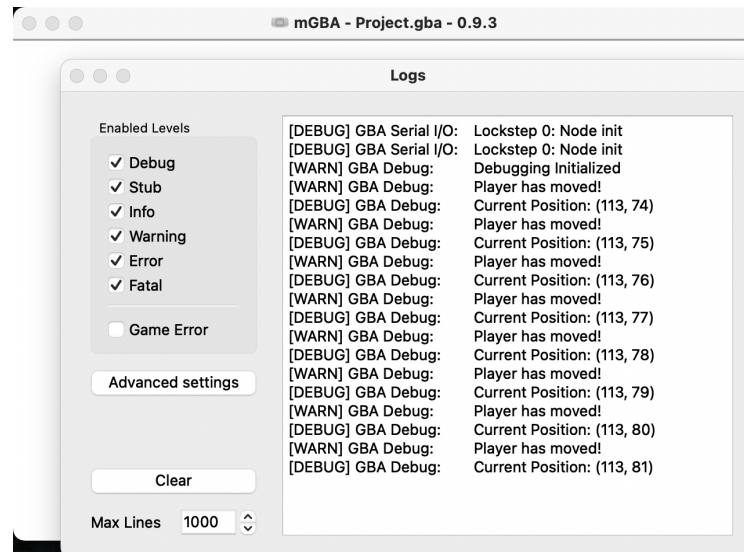


- By default it will only show “MGBA\_LOG\_FATAL,” “MGBA\_LOG\_ERROR,” and “MGBA\_LOG\_WARNING.”





- Uncomment UNCOMMENT #6 and compile and run.
  - Checking either Debug or Info will reveal “MGBA\_LOG\_DEBUG” or “MGBA\_LOG\_INFO” respectively.
  - Check “Debug” and you should see that when you move your painting rectangle, the logs will update with the current position on the screen! (relative to the top left pixel).



- **Hint:** Printing will be a useful tool for you throughout the course! Refer to the Debugging document on Canvas for more details, in case you forget.

## TODO 5.0-5.2

This is the most exhaustive part of the lab, but also the most important.

### TODO 5.0

- At the bottom of lab2Lib.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 5.1

- Now that you have the collision function 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 yellow rectangle, it should become yellow.

- **Note:** You should still be able to change the painter's rectangle back to black or blue by pressing A or B, unless the painter's rectangle is touching one of the colorful rectangles.
- **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 the variables for the color, height, width, row, and col of all of the colorful rectangles at the beginning of main.c.

### TODO 5.2

- Now you will add the “painting” function. Find TODO #5.2 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 key 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, and then you are done.

### You will know if it runs correctly if you can:

- Move the painter rectangle using the arrow keys.
- See player position update in the Logging window
- Press A to make the painter rectangle black and B to make it blue.
- 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 Select 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).



## Tips

- Start early, and attend recitation.
  - Use graph paper and draw pictures to conceptualize what is happening.
  - 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 Lab02\_FirstnameLastname, for example:

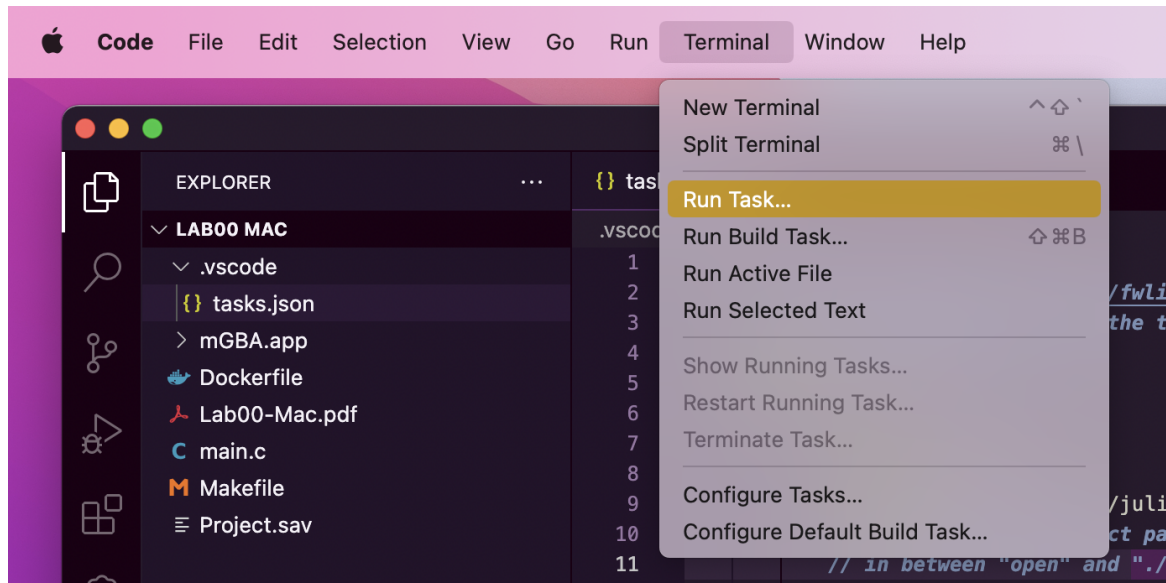
“Lab02\_KingDedede.zip”

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.

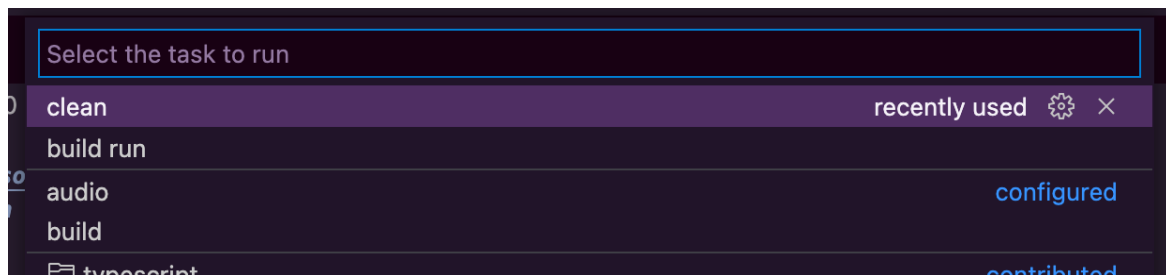


## How to Clean

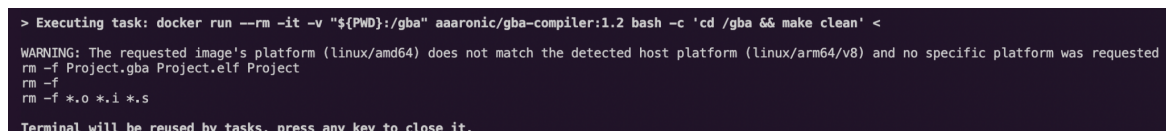
Navigate to the **Terminal** option at the top of your screen. Click on it, and then select **Run Task...**, as shown in the image below.



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**).