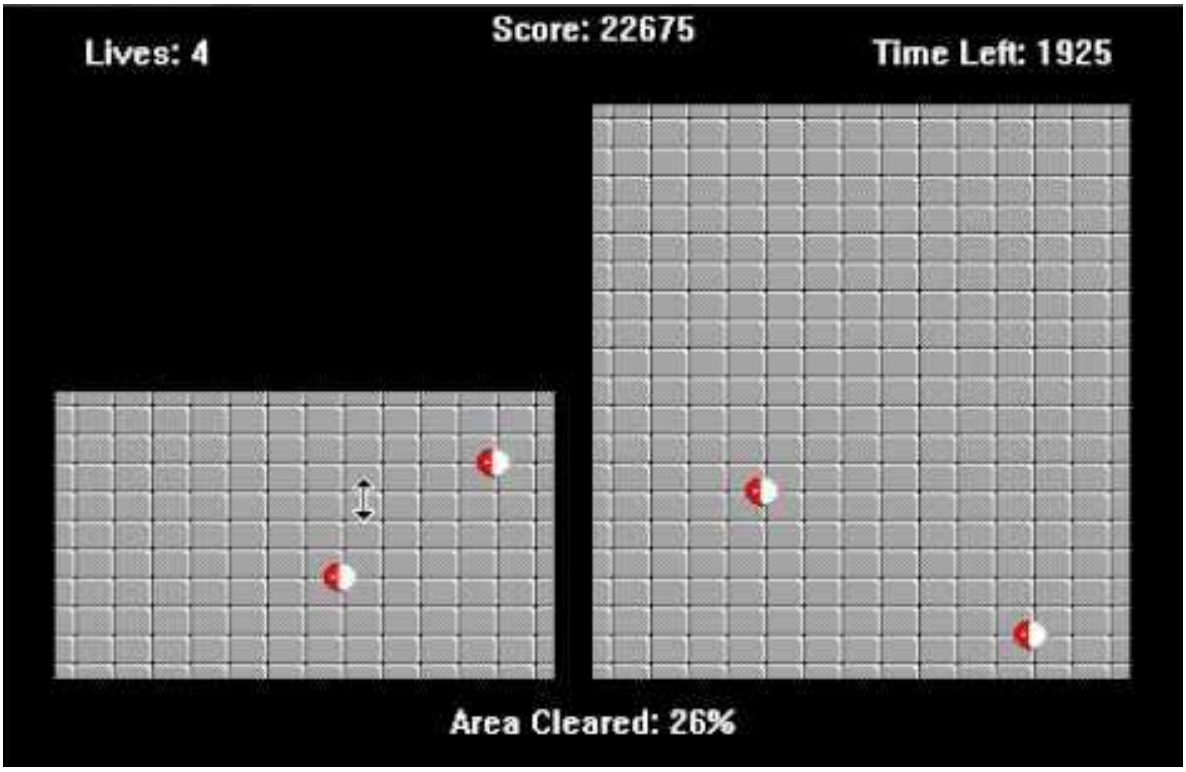
HOMEWORK 02:

JezzBall



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 create a simplified version of the game *JezzBall* in Mode 3. If you are unfamiliar with the game, you can watch this playthrough of JezzBall. Review the **Requirements** section of this document for an explicit list of what we expect as the base requirements. The basic idea of our version of *JezzBall* is as follows:

The player will try to trap a bouncing atom. At the start of the game, the atom can bounce anywhere on the screen and is only bound by the screen's borders. The player can draw vertical or horizontal lines across the screen: these lines will divide the screen and restrict the atom's movement. When the player draws a line through the bounding

box, the box will be split in two: the part of the box where the atom is currently bouncing will become the new bounding box, and the other part of the original box will become part of the background. The player wins if the atom is trapped in a small enough box. The player loses if they lose all lives. They lose a life if a vertical or horizontal line is drawn directly through the atom.

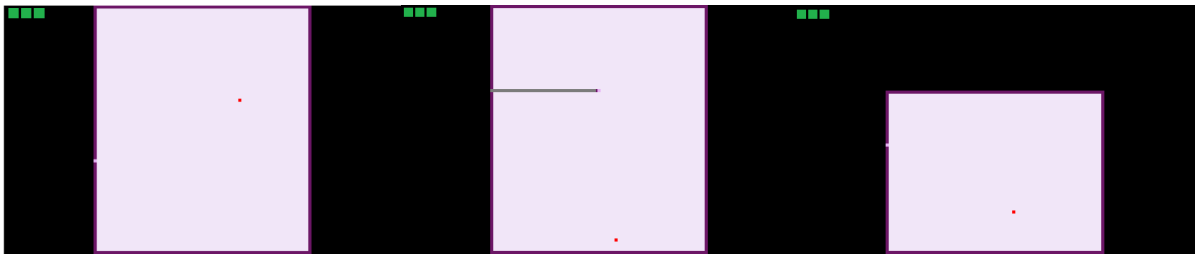
Moreover, you are encouraged to be creative! Go outside of the requirements to add some flair to your game. **You will receive a maximum of 95 out of 100 points for meeting all of the base requirements. By adding your own flair, you *may* be awarded the 5 additional points needed to receive 100 points.**

Requirements

Gameplay

Your *game* must have the following:

- The player can move meaningfully and draw horizontal and vertical lines through *intuitive* user input controls.
    - The player rectangle must constantly move around the perimeter of the bounding box. The player can press a button to draw a line in a specific direction (UP for upward line, LEFT for line to the left, etc.).
    - If the player attempts to draw an invalid line (e.g. drawing a line UP while on the left/right side) nothing should happen.
    - The lines drawn by the player should be visible (the color should be different from that of the bounding box).



Here, the player rectangle is light pink, moving around the perimeter of the bounding box. The player presses RIGHT to begin drawing a line to the right. Then, a new bounding box is formed.

- The bouncing atom should be no smaller than 2x2 pixels and should always move throughout the bounding box (no faster than 3px/frame). If the atom collides with the top/bottom or sides of the bounding box, it must bounce off and change direction.

- The player should draw lines that move to the opposite side of the bounding box. When a line reaches the other side, the current bounding box should be split along that line, into two areas:
    - The part of the box where the atom is currently bouncing will become the new bounding box. For example, if the player draws a horizontal line and the atom is below the line when the line reaches the other side of the bounding box, the bottom section of the original box will become the new bounding box.
    - The part of the original box where the atom is *not* bouncing will become part of the background. In the earlier example, the top section of the original box will become part of the background.
    - Make sure to choose a drawing speed (how fast the player moves when drawing lines) that ensures the game is winnable but not too easy.
- The bounding box should be visible (the color should be different from that of the background).
- The player should have 3 lives, visually indicated by a tally on the top-left corner of the screen. When the player loses all three lives, they should lose the game.
    - There should be visual representation of when a life is lost, meaning that one of the boxes/rectangles in the tally should disappear.
- The player should lose a life if a (vertical or horizontal) line is drawn directly through the atom. This would happen if the player rectangle itself collides with the atom, or if the atom collides with a partially drawn line (before it becomes a bounding box).
    - If a line isn't successfully drawn (the atom collides while drawing), the player rectangle should return to its starting position and the line should be erased (and a life indicator should also be erased).
- The player should win if the "playable area" shrinks to be less than **625px** in area.
    - You'll need to calculate the width and height of the area and multiply them to get the area in pixels. 625px is equivalent to a 25x25px area.
- The game should enter an end state once the player wins or loses.
    - It must be clear that the game has ended; the player should not be able to play indefinitely. Preventing the player and atom from moving is okay here.
- Only a minimal amount of flicker.

Code/files

Your *code* must have the following:

- **You must use the provided HW02 scaffold!**
- 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.
- 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 multiple enemy atoms.
- Change the end state to do something fun when a player wins.
- Do something different for the win state and the lose state.
- Make your game look extra nice!

If you have questions about whether a specific flair will earn 5 points, please ask Julia about it!

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.

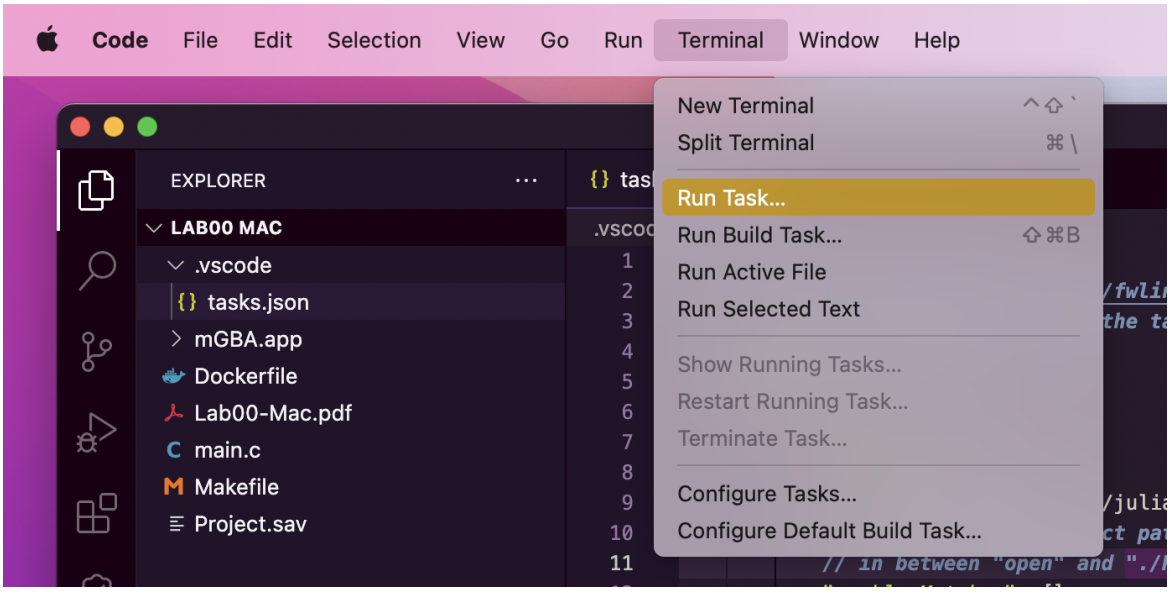
Zip up your entire project folder, including all source files, the Makefile, and everything produced during compilation **(including the .gba file)**. Submit this zip on Canvas. Name your submission HW02_LastnameFirstname, for example:

"HW02_TurnerTimmy.zip"

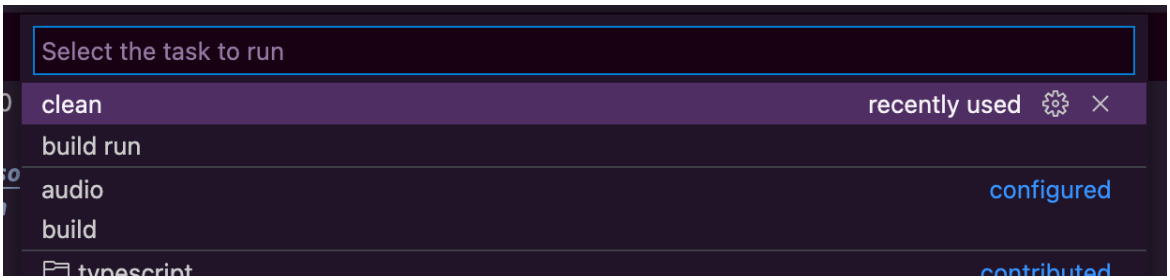
It is your responsibility to ensure that all the appropriate files have been submitted, and that your submitted zip can be opened and everything cleans, builds, and runs as expected.

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**).