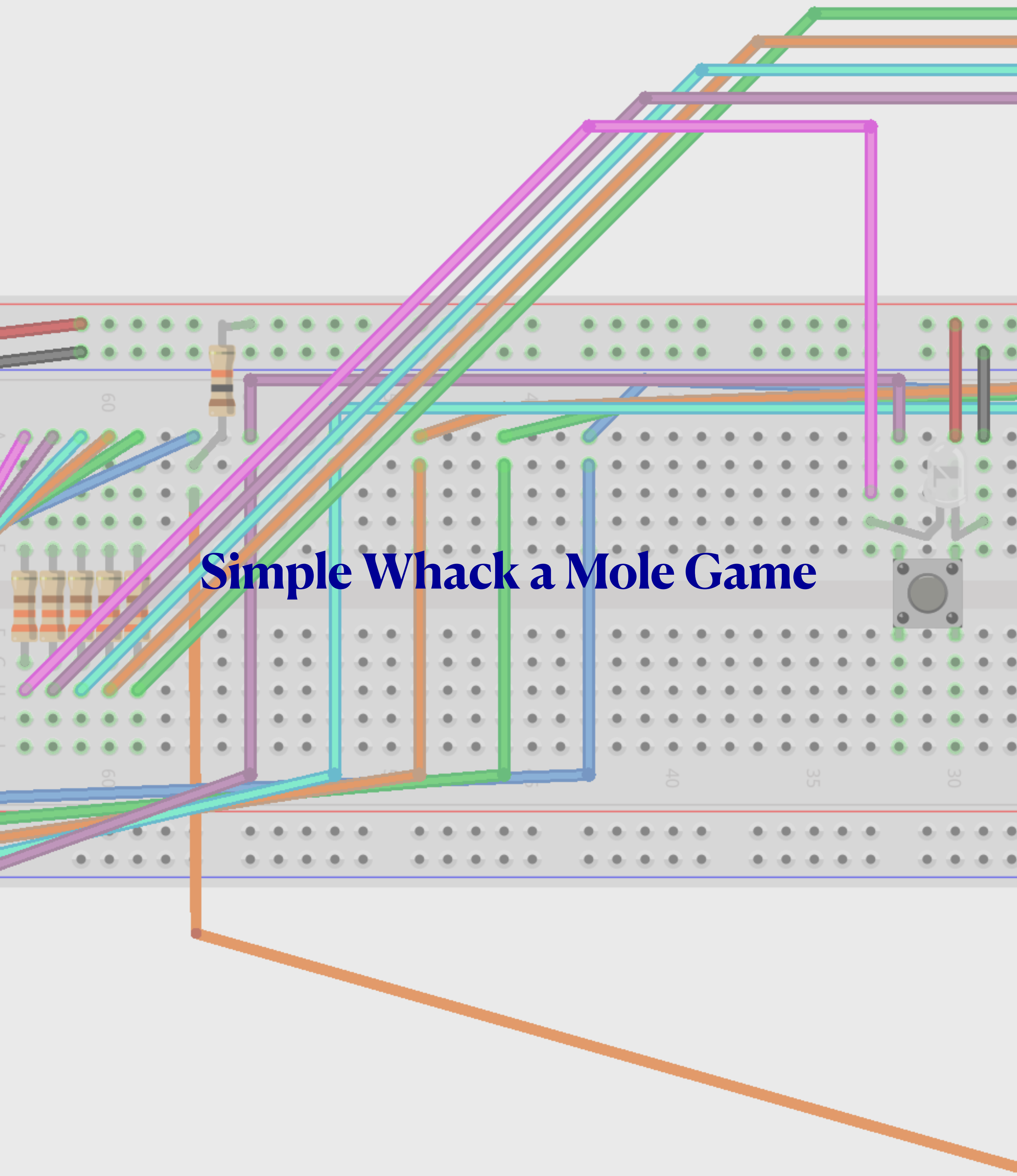
Simple Whack a Mole Game



Abstract

This brief report is to supplement the C++ code for a whack a mole game created using a Nucleo-G071RB microcontroller and an Uno R3 project starter kit. It explains how to set up the hardware, and summarises the software design, planning, and debugging process.

1. Setting up the hardware

1.(1) Schematic

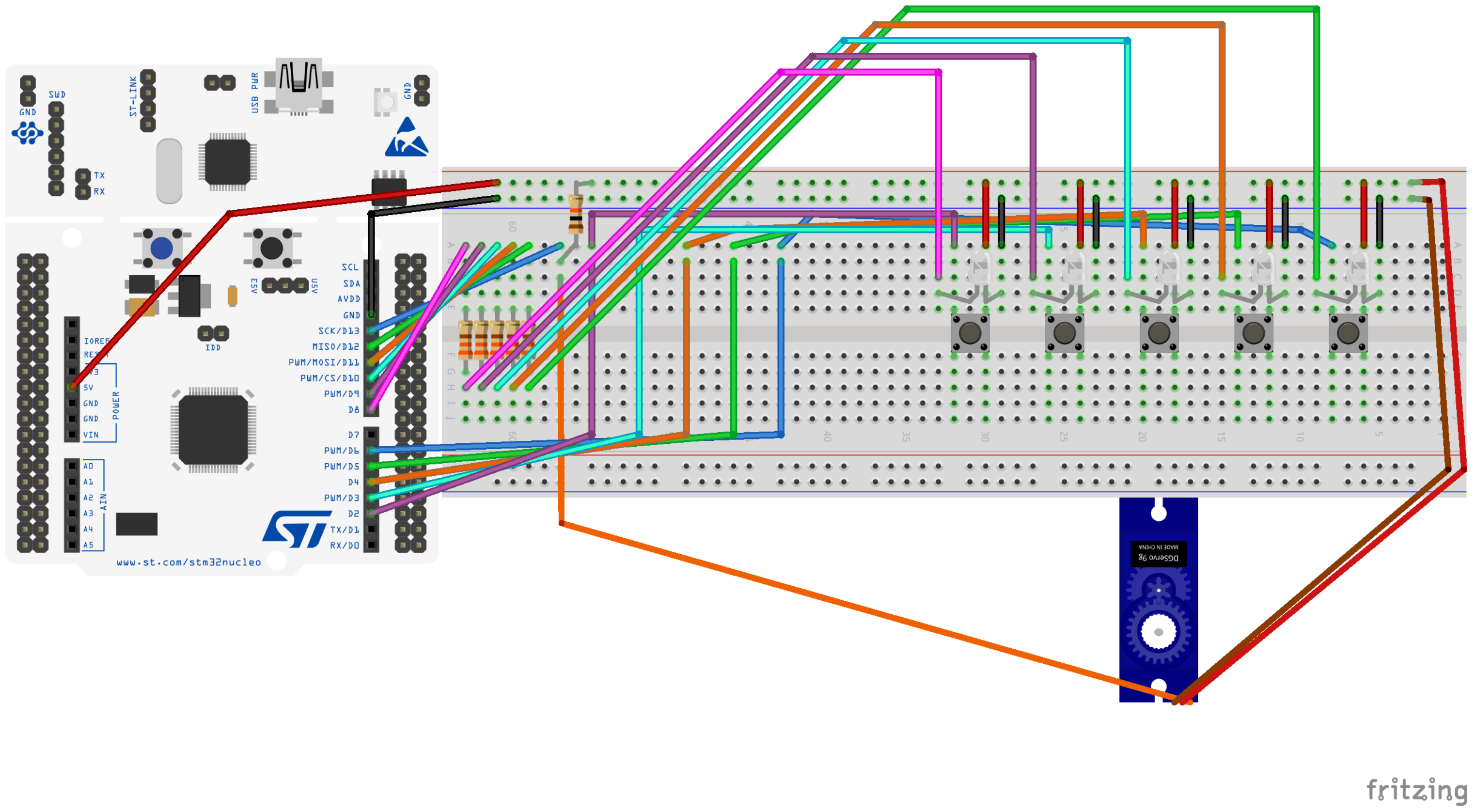


Figure 1 is the breadboard schematic for the hardware and was built using https://fritzing.org/

- Attach 5v and ground to the top rails.
- Attach pins D8, D9, D10, D11, and D12 to the anode of one of each of the five LEDs, in the order specified by the wire colours, with a 330 ohm resistor between each LED and pin.
- Attach the cathode of each LED to the ground power rail.
- Attach pins D2, D3, D4, D5, and D6, to one side of each of the five buttons, in the order specified by the wire colours.
- Attach the other side of each button to the 5v power rail.
- Attach the red Vcc wire from the SG90 servo to the 5v power rail.
- Attach the brown ground wire from the SG90 servo to the ground rail.
- Attach the orange signal pin from the SG90 servo to Vcc via a 10k ohm resistor, and straight to pin D13 with no resistor.

1.(2) Attaching the white servo arm

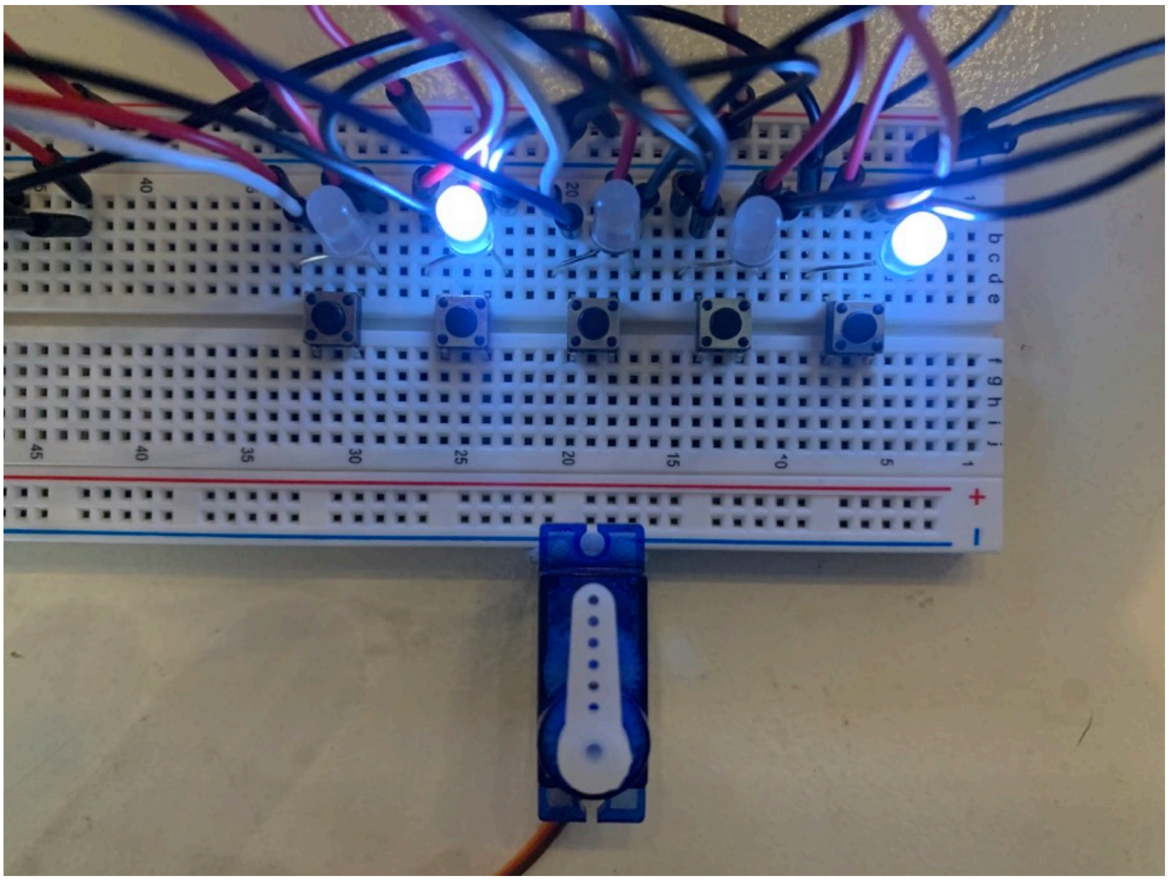


Figure 2 shows how the white servo arm should be positioned when you press the middle button on the first time playing the game.

- When the program first runs press the blue button on the nucleo board when prompted.
- Now press the middle of the five buttons on the breadboard.
- Now attach the white servo arm to the SG90 servo such that it is pointing towards the middle LED as shown in Figure 2 above.

2. How the code works

2.(1) How the code works

Please see the comments in the code itself for a description of how it works.

2.(2) Plan used to write and debug the code

Plan for coding whack a mole:

- Get a random number generator from 0-4 working, which is generated when the blue user button is released to make the sudo-random number seem actually random.
  - Use Interrupt service routines and run this in a separate thread to main(), so that it can be activated when the servo is running.
- Create an ISR that prints a random number from 0-4 at random intervals within a certain time range, in a sequence that is started by pressing the blue button: **this didn't work so I will diverge to getting it to work as soon as the program starts, then the user can just press the black button to restart gameplay.**
  - In order to have functionality where up to 4 numbers could be output at once, use a separate thread for each number, and have each thread sleep for a random amount of time, within intervals, then print the number.
  - For debugging purposes use a separate semaphore to ensure printf is only called once instantaneously
- Now, get this working with the hardware LEDs
- Create 1 semaphore for each number LED, with one token that is only released when the LED is on:
  - Pressing the correct button attached to the LED during this time will acquire a token from that semaphore
  - Use try_acquire method of each Semaphore object so that when the LED_off_check callback function returns, it checks if a resource is available and if it is, is will acquire the token from the semaphore and return true, which will cause the LED_off_yes callback to activate turning the LED off: **this is extra scope as it involves more complex deferred execution than I thought: I will just make the lights blink for the same period of time regardless, and then have pressing the button during this period give you points.**
  - If try_acquire returns false, it is assumed that pressing the button at the correct time acquired the token from the semaphore and activated the LED_off_yes callback function.
  - The random interval generator will then iterate again and turn the LED on after a period of time
- Now implement a score which iterates each time the correct button is clicked when an LED is on:
  - **I have tried and the errors seem to be to do with the rise/fall of the pin, so I will take a step back and figure out how the logic of the pins works.**
  - Use DigitalIns, or another object, so that a signal from a button calls try_acquire for that LED's semaphore object: if it returns true, a token will be acquired from that semaphore, the LED_off_yess callback called turning the LED off, and the score iterated by one
  - If the try_acquire for that LED's semaphore returns false, no token will be available from that semaphore, and the score will be decreased by one as penalty
  - Each time the score is iterated, print it on the console
  - Whenever the button is pressed, no matter if try_acquire returns true of false, the rotate_servo.start() method will be called such that its arm points towards that LED
- Note in the user instructions that you should press the middle button and attach the servo arm at the correct angle before gameplay, then press the reset button to set the score back to zero, then press the blue button to start gameplay.

3. References

[1] https://www.youtube.com/watch?v=uxgEkgUDB0I
[2] http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
[3] https://electronics.stackexchange.com/questions/246642/power-supply-circuit-for-powering-sg90-servo-how-to-handle-high-current-require#:~:text=The rated current for a,50mA, stall 650 ±80mA.
[4] https://en.cppreference.com/w/cpp/language/class
[5] https://stackoverflow.com/questions/4184468/sleep-for-milliseconds
[6] https://github.com/ajp109/ELE00029C-exercise
[7] Lab 7 Event Driven Programming
[8] https://os.mbed.com/platforms/ST-Nucleo-G071RB/#nucleo-pinout
[9] https://www.circuitbasics.com/pull-up-and-pull-down-resistors/
[10] https://cplusplus.com/reference/cstdio/printf/