Arduino coursework 1 Whack a mole

Report of Development

Description

Create whack a mole on the arduino, which is where lights appear randomly and the player must gain points via clicking the button at the time the light is on.

#####

Step 1 - Check whether the button press happens at the same time as an LED

For this step I decided to use an **interrupt**. The reason for using this method was because an interrupt can be used to **pause the execution of code** in the loop() function in order to allow a defined method to run its code. This interrupt was linked to the **button** on my board, where its pin on the board that it was attached to is digital pin number 2. I could only choose number 2 or 3 as these pins support interrupt signals. To attach the interrupt to the correct pin signal , this code was necessary, notice for the button i required it to be configured to INPUT_PULLUP as it was not working otherwise:

```
attachInterrupt(digitalPinToInterrupt(2), playerOneInput, FALLING);
```

This would attach an interrupt to digital pin 2, of which the push button on my board was attached to. So every time the player pressed the pushbutton on the board, it would run the function called *playerOneInput*. It also required the designated action of *FALLING* to be stated, which essentially means run this interrupt when the signal of the button goes from 1 to 0 , so when its being pressed down essentially. Within the interrupt playerOneInput , we have now caught that the player wants to register a hit, so we simply cycle through each led in the array of led pins and check if any were on, if one was on, then we register it as a hit. An issue i encountered was the problem that during a light being on the player could hit the button multiple times in order to gain multiple points, to solve this in **playerOneInput()** there is a procedure which checks the light, however only if the *foundRed* (or for player 2 version foundBlue which has the same logic implementation) variable is false, which is to say has red been detected and clicked by the user already, if so, skip the interrupt procedure until the light turns off which is where i describe in **randomLight()** the resetting of the found variable to be enable the interrupt again to begin registering user input again. pros, A Pro of this method is the use of interrupts, as they essentially meant it does not matter when the user presses the button, it will execute the code in the procedure defined pausing execution of any other sections of code. It requires more simple code and a more structured design that is suited to how the game works. Cons , Using interrupts could lead to the execution becoming slightly cluttered and janky if there is too much code in the actual procedure defined, therefore it should be kept to a minimal if possible.

Step 2 - Add a score that is incremented once when the player presses a button when the LED turns on.

- This is essentially an addition to step 1 that detects hit, where we simple just increment a global value of either red points and blue points to 1 if they register the hit correctly the code for this is shown in the **playerOneInput()/playerTwoInput() procedures** where the points value is incremented for a correct hit, the same code will be for blue points but of course a different piezo tone and variable names.

Step 3 - When the score reaches 10, flash all LEDs to indicate the game is over.

So essentially for this section, i require a function that is looping and checking the value of red points and blue points Within this function it will have check if the points value is >= 10 to say the corresponding colour has won. I will loop an endless while loop that will turn the LEDs on and off as to flash them every 500ms. This is built into a procedure called **checkEnd()**. As seen from the **loop()**, this is executed after running the random light selector and the check who is ahead procedure. Pros, Clear winner from the clean flashing lights that is checked every few second after the execution of the other code. Quite reliable in determination of the winner but not entirely optimal. Cons,This is unlikely due to the time frame that the arduino runs on but there is a potential for it to determine the wrong winner if for say red has won, in sense that red got points faster but due to the sequence of the code, blue may be determined the winner if their is a fault in the timings.

Step 4 - Extend to 2 player mode, add three more LEDs and a second button , adapt the game as so button1 links to red and button2 links to blue.

- This was essentially just doing everything i was when i had a player one but implementing it to accommodate a second player . had a playerTwoInput() procedure now which is the same as playerOneInput shown in **playerOne/TwoInput()**, but i introduced a few new variables. I also needed to attach another interrupt to the button as to detect player two inputs and link to the playerTwoInput Procedure. In terms of the random generation , as seen in **randomLight()** the lights generated is actually unique to each one as so they dont get the exact same light to give the game a sort of variation too it. Pros, Simple and clear implementation that enables a player 2 to play alongside player 1 with ease. Cons, Duplicated code. My interrupts both basically do the same thing but for alternative colours, however i cannot pass parameters into interrupts to prevent this. Also a duplicated code in the checkEnd() procedure which should not be too much of an issue but can certainly be made more efficient

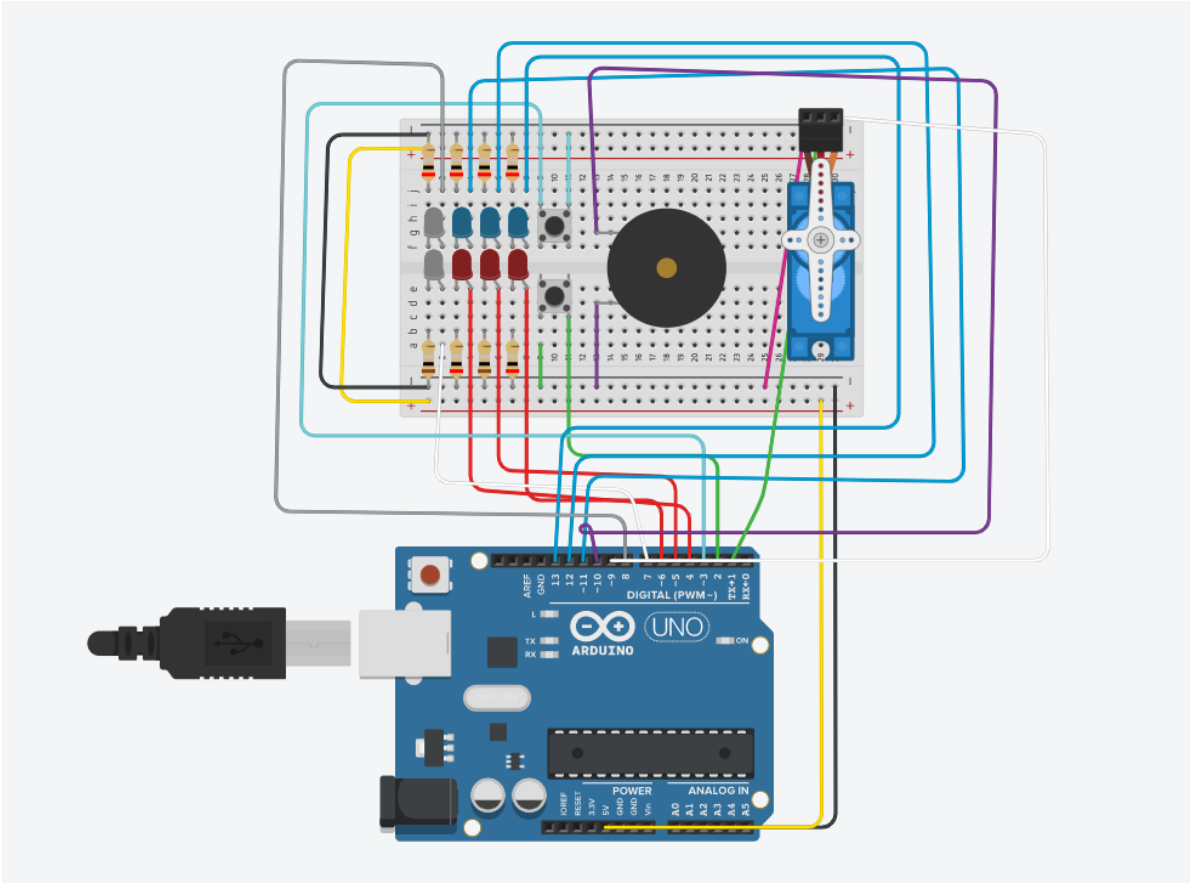
Extension addition 1 - Include a sound output (for correct and incorrect button presses)

- For this section i simply connected a *piezo* device to my circuit and configure it to make a unique low and high tone for each players corresponding hit and miss on the arduino lights. I decided to seperate the sounds out slightly as to just give a bit more insight into the perform on the players. The code i used to implement this is shown in **playerOne/TwoInput()** where the high pitch noise is played. If the player has pressed a button and no light has appeared , then the for loop will not set foundRed or foundBlue to true and therefore after the for loops execution a low tone on piezo plays. It would be the same code that runs for foundRed but a lower tone to make it sound a little different. Pros,Clear indication of if a correct button has been pressed or incorrect. Cons, Gives the game a more interactive feel as their is louder more clear sensory output to make it feel more like an actual arcade game.

Extension addition 2 - Using a servo to determine who is in the lead of the game

- I liked the idea of this feature as it actually gave the users insight into leader whereas before they were blind guessing or just counting the beeps / light flashes.
- This was implemented simply using a servo with varying degrees depending on the score the users were at
- If the scores were equal then i would write 90 degrees to it as to put it in between the two players, if the red was higher then blue then i would write 150 degrees to it to point to red side and finally if blue score was greater than red i would write 30 degrees as to point towards blue.
- This function was checked in the loop after each random light execution.
- I did have it at 0 for blue , 180 for red and 90 for equal before however 0 and 180 by literally looked like pointing towards nothing therefore i tweaked it a bit to be more clearer.
- This is coded in a procedure called **checkAhead()** in my code.
- Pros,Clear indication of the lead in the game to give it further interactivity
- Cons, as it runs after the delays of random , which are short, it does take a slight delay before registering the lead, however it does not affect the game really.

Arduino board



Schematic view