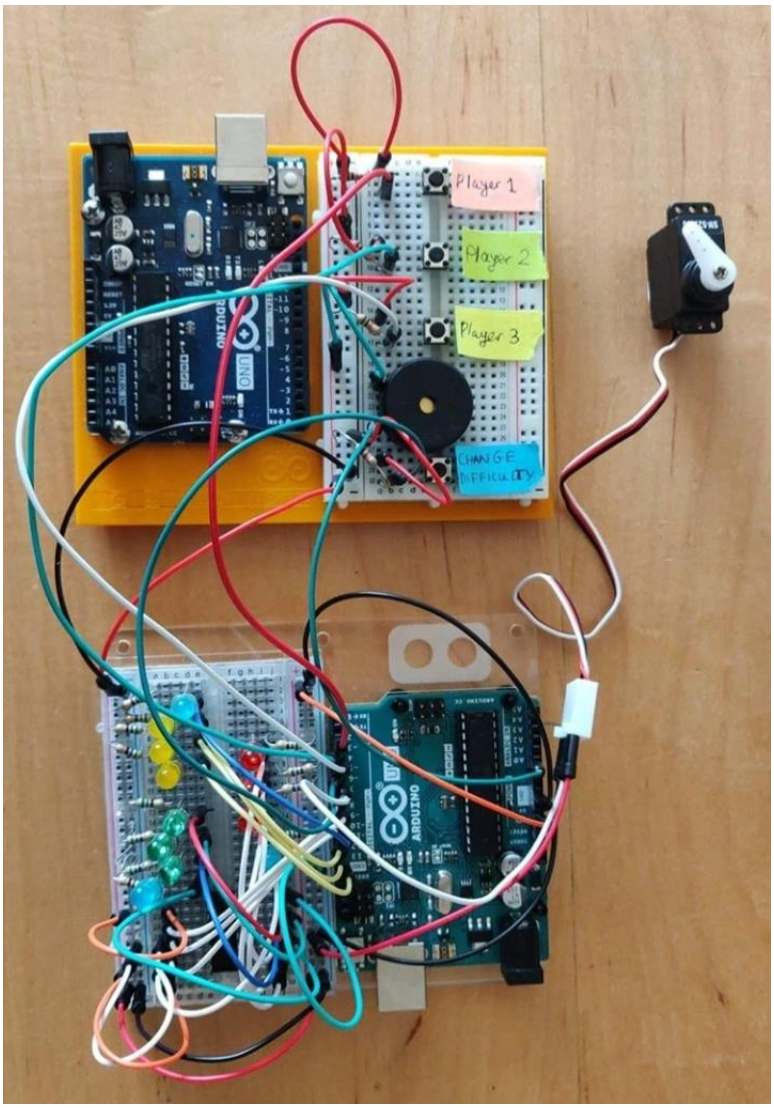


Arduino Whack-a-Mole

A REPORT

Alex Zamurca and Vedant Nemane| Group 71| CM10194 Coursework 1 | 04/11/2019

Introduction

The whack-o-mole program is an adequate introduction to the Arduino board and a great way to familiarise ourselves with the way the components work as well as show the extent of the capabilities of the Arduino. Programming the Arduino board is more interesting than that of writing programs on a computer for a console since the physical components and circuitry are a major factor too.

Because of this, the code and circuit must be changed every time the specification changes or the requirements are changed- for example changing from one player to two players is relatively easier, compared to upgrading from two players to three because of the numbers of interrupts and the limited space for LEDs. This meant that we had to plan how we were going to put in place the features, and consider the time constraints that we were going to set for completing each of the goals.

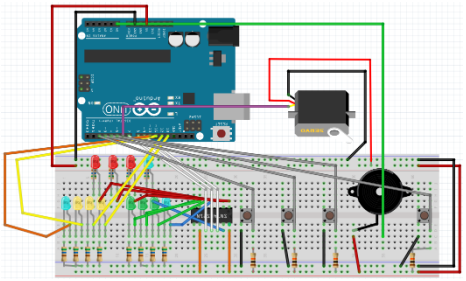
How the project was implemented

We have decided to include a downloaded library that allows the Arduino to use any pin as an 'attachInterrupt' pin. This comes with the benefit that the shift register does not have to be used for attachInterrupts and any number of pins can be used as inputs from here on out. The winning melody is also something we thought would be an interesting feature to add to make the game more interesting and feel more rewarding but again, this takes up more lines in the code and increases the size of the file. But because the libraries have to be imported, the files will take up more storage space. And since we do not actually own the library, we do not have rights over it and have only used it for educational purposes.

We decided to use interrupts to implement the buttons which overall turned out to be a good decision since the alternative would be difficult to implement and sort out timing for. By using interrupts, we made sure that no matter what, the code in the interrupt ran all the way through and then resumed with the main loop. However, the disadvantage with using interrupts is that you are often limited to 2 interrupts due to the type of Arduino that we are using and that delays cannot be used inside interrupts as they are meant to be run instantaneously.

Something that in hindsight which was a bad decision was using a shift register to get extra pins for the LEDs. Using a shift register meant that we had to use more complex code and essentially rewrite all of the old code to accommodate for this component as well as change the wiring. Nearer the end, we understood that we could have also used the analogue pins as digital output pins, which would have completely removed our need to use the shift register or the function called 'updateshiftregister'. However, because we did, the wires are more spread out and in general is much neater than if we had implemented this, which also made it easier to fix.

How the project works



The code begins by declaring the inclusion of certain libraries and definitions of the musical notes for the piezo. The arrays that relate to the melodies which need to be played according to the notes that they contain are also initialised. The win melody- taken from Instructables is also credited within the code itself. Durations for the notes being played also need to be declared initially too. An analogue pin was used an output pin for the piezo. All the LEDs are declared which include 3 LEDs for each of the 3 colours as well as the score LED for each of them. One interesting coding format we found for using the shift register was that it gave an extra 8 outputs, coincidentally the same as that of the number of bits per byte. Furthermore, storing the status of LEDs within an integer data type would be inconvenient since it is particularly difficult to manipulate individual digits contained within the number, and only two data values need to be help per digit (whether the LED is switched on or off). Hence the introduction of the data type called byte was instrumental in controlling the shift register. Boolean variables were also declared to store whether the player(s) had hit the button on time which decided which sounds were played on the piezo.

The function called "setup" configures specific ports to be ready for digital output and configures the interrupts for button input. The main loop can be split into three major parts: the LEDs; the servo; the checking.

Every time the loop is run, the current scores are displayed on the serial monitor, the LEDs are all switched off for a split second, only to be updated with a random LED light up. Initially, there is also a pre-condition that all scores must be below 10- if they are not the program skips to flashing buttons when a player has won. This also helps with keeping the program flashing lights rather than continue with the game or flash lights incorrectly. The delayTime variables work off the data provided by the difficulty button- which means that although the code uses the same variable for selecting when to switch on an LED, the time is always based on the current difficulty set.

The servo is controlled by doing 'if' statements to check who is leading and getting the servo to point towards the button (by using 45-degree increments) of the player who is winning.

The checking block checks if any of the players have reached a score of 10, in which case they have won, and all LEDs will start flashing after which the winning tune is played. It was found that you could not play the winning tune AND do the flashing of the lights at the same time since the playWon() function must be resolved and removed from the instruction stack before the lights can play. With more time we could have resolved this issue with the implementation of threading. At the end of the code, all Boolean variables that check for repeated input from the button or whether the player hit the button correctly are reset to false. Functions for playing the melodies are also declared after the main loop function. The button interrupts are then declared for each player, running similar code for each player. All a player's LEDs are checked with a for loop and if one of them was on, then the score must be incremented, output sent to serial monitor, update the status of the Boolean for repeated input and make hit to be true. The difficulty function checks what the current difficulty is and provides the appropriate time delay for that. This works because of the use of global variables that retain their value after being changed throughout multiple functions. The function called update shift register refreshes the LEDs.