Combining the LCD and keypad codes.

The code that moved the cars steering when # or \* is pressed was taken and a message displaying what option has been chosen has been added to each ‘else if’ statement to combine both components. Not sure if this will work as the same ESP32 pins are used for both so will set up on a breadboard first to test it.

When it’s run the code doesn’t work, the numbers are not displaying the number that was pressed and the LCD display isn’t showing the correct messages.

I tried to moved the keypad to the other side of the ESP32, I changed the pin numbers in the code to reflect this.



I used GPI0 pins on the other side of the pi to prevent any overlap, when this was uploaded the serial monitor showed endless 1s being pressed even though this was not the case in reality. This meant that the LCD was displaying continuously the message set for a number which meant that aspect of the code was working, there was an error in selecting the suitable pins. After a bit of trial and error I still couldn’t get it to work.

I used the link <https://arduinogetstarted.com/tutorials/arduino-keypad#content_about_keypad?utm_content=cmp-true> to do some further research into how the keypad works to pick the correct pins. The row pins should be connected to output pins and the column pins are connected to input pins in a pull up state so that if the button is not pressed it is in a high state. I used <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gpio.html?highlight=gpio%20pull%20up#:~:text=ESP32%3A%20Only%20pins%20that%20s,s%2034%2D39%20do%20not>. To do some research into which pins can be set as high states for the column pins and to find the code to enable this. The table on <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/> was used to find out which gpi0 pins could be set as pull up as only pins that support input and outputs can be set, this was too complicated so the snippet of code below was gotten from <https://www.upesy.com/blogs/tutorials/how-to-use-gpio-pins-of-esp32-with-arduino> .

Graphical user interface

Description automatically generated with medium confidence

Configured column pins as inputs and pulled them up.

Text

Description automatically generated

And added a command to clear the LCD display when a new key is pressed so that the message will remain on the screen until a new command is given. This was put inside the if key loop so that it only clears upon pressing something. I swapped 21 to 13 and 22 to be 4, this now worked so the serial monitor and LCVD screen both displayed the correct values for all rows and columns.

Text

Description automatically generated

This worked well for the 3rd column, it could read the buttons pressed and display the correct number on the serial monitor as well as the correct message on the LCD display. The 1st and 2nd column did not work at all.

All the functions/commands for the LCD library can be found <https://www.make-it.ca/i2c-lcd-display-on-arduino/#:~:text=clear(),when%20you%20start%20your%20program> .

At this point the rows seem to work but to confirm I set the display to show which number has been pressed by adding this bit of code. This displays the value chosen by the key press on a new row, this means that when 3 is pressed it will display 3 etc for each number. This confirms that the row pins are functioning as the number is correctly identified, this means that it is just row 1 and 2 that are causing an issue.Graphical user interface, text

Description automatically generated

I realised that the reason the first 2 columns were giving an issue is that they are 21 and 22 which are the sda and scl pins so they had to be moved. I swapped 21 to be and to be. Now when tested this works, the correct number and message is now displayed for all of the buttons pressed.

Next the breadboard circuit had to be transferred to the EEEBot. I used headers to attached the LCD screen and to attach the wires from the ESP32 up to the 3rd layer. Although this means that the wires will always have to be entered into the headers in the same order, it means that the board can still be removed if needed. I took lots of pictures of the breadboard circuit and notes to confirm which pins of the ESP32 were connected to which colour wore and which pins of the LCD and keypad. As there are limited colours of wire that I could chose from some colours are duplicated between each side of the board. I did my best to keep each duplication separate, for example, green was used for both components but only once for each. I decided to have the headers opposite each other. This means that the track had to be cut down the middle but I preferred to do this instead of moving the header further down the board. This would have meant longer wires connecting the 2 board layers, longer connection running along the board and would have left less space to stick the keypad later on. I made an error when designing the common ground, my plan initially was to have the ground strip run along most of the width of the board to allow easier connections to the LCD as there are 4 pins connected to this. I ended up connecting the orange in between these ground connections meaning that I had to break the track and reconnect the ground on another strip. This still worked and connected everything that was needed it just meant that I had to solder a few extra wires onto A picture containing text, electronics

Description automatically generatedthe board.

When this was finished and the code from the breadboard circuit was uploaded it didn’t work. The LCD display was displaying an incorrect message of ‘number chosen 7’ even though different numbers were being pressed and the serial monitor was not displaying any numbers at all. The first step was to check the circuit design, we cross-referenced it with the notes I had made and the pictures of the original breadboard circuit that worked. As all of these matched so we performed continuity tests across the cut track to ensure that they were all don correctly. One of the cuts separating the yellow and orange wires came back as still connected. To fix this the cut was made a bit larger until the continuity test stopped beeping meaning the track were no longer connected. Next I uploaded the code again and tested it by pressing different buttons. The correct numbers were now being displayed on both the LCD and the serial monitor as well as making the motor speed and servo angle change as expected. This means that everything is connected correctly and communicating properly. This means that the code can now be developed to enable the EEEBot to perform maze navigation as required. The code used to test is called ‘display serial and LCD numbers’ even though it also moves the motors.