Disability Friendly Pictionary

CSE 321 Project 3

Andrew Schick

Fall 2021

Table of Contents

Introduction

This document is the culmination of a four-week design process. This document outlines system specifications, internal software features, the design process, user setup, and a testing plan for a 2 to 4-player disability friendly Pictionary game. The project idea came from my love of painting and drawing which individuals with very shaky hands or missing fingers can struggle greatly at. The idea of the project was to map hand motion to display on a screen allowing users to draw without having to pick up a pen or pencil. While the motion mapping is the main feature of the system, it also comes with a display to keep score and time rounds as well as a keypad to make guesses and change the drawing functionality.

Specifications

- Keypad inputs are software debounced
- Ports in use:
    - E – keypad outputs
    - F – keypad inputs
    - C – IR sensor outputs
    - A – Dot Matrix inputs
    - D – Dot Matrix input
- All components can hand 3.3V to 5V power supplies
- LCD uses SCL and SDA pins
- Dot Matrix uses miso, mosi, and cs pins
- Dot Matrix is controlled via whole LED rows
- Only the first 2 rows of the keypad are in use

Features

- A single game has 3 rounds
- The prep a drawing stage is 5 seconds
- The drawing/guessing stage is 25 seconds
- Continuously updating LED Dot Matrix to display the drawers’ picture
- Blinking cursor
- Score, current round, and time left display
- Runs indefinitely
- Thread safe
- Synchronized

- Keypad constraints:
    - ‘A’ – invert pen
    - ‘B’ – invert eraser
    - ‘1’ – player 1 is making a guess
    - ‘2’ – player 2 is making a guess
    - ‘3’ – player 3 is making a guess
    - ‘4’ – game start
    - ‘5’ – incorrect guess
    - ‘6’ – correct guess

Required Internal Feature Integration

Watchdog – Used to restart a program in which a thread is hanging. For this program all of the threads are polling except for one, game_t, which runs through multiple while loops and also interreacts with the global mutex. For this reason, if this thread hangs, the game state can lag within a stage in which case the watchdog will restart the game. The dog will be kicked after each stage is completed.

Synchronization – Method used to coordinate a programs event. The only synchronization used is a single mutex that guarantees safe state for the Dot Matrix. This is very important because there can be 3 threads at anytime trying to access and update this data. Wherever this data is altered or read from in the code is encapsulated by this single mutex.

Bitwise driver control – Manually bit masking specific pin or port functionality/state. The only bitwise control done in the code was for the keypad polling. It was used to easily high and low state for the rows on the keypad.

Critical section protection – Using semaphores or mutexes to synchronize thread reading or writing of shared memory. Only one mutex is used because the only shared state is four variables pertaining to the Dot Matrix. Writing to these variables can not take place at the same time due to the blinking LED functionality.

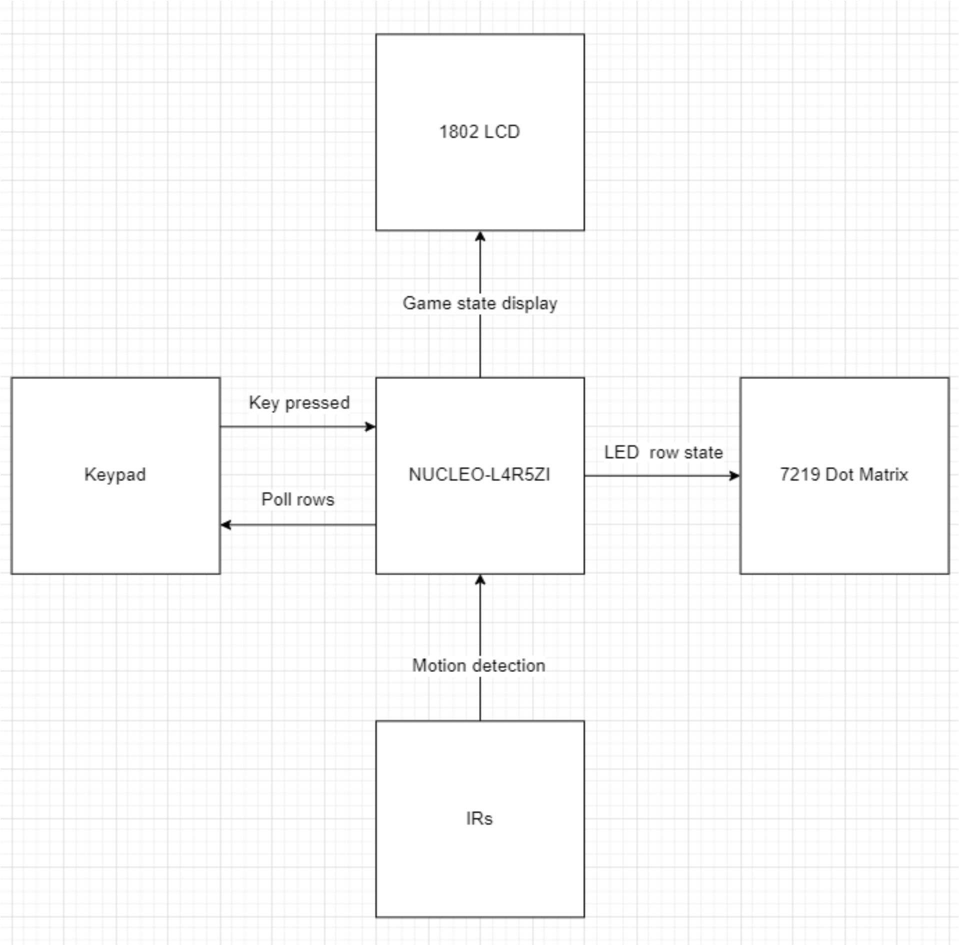
Threading – Splitting up a program into multiple concurrent processes. There are four total threads running at any given time including the main thread. The idea was, for each object/state that needed polling, it would be done by an individual thread. Hence the threads are used to poll; keypad inputs, IR outputs, inputs to the Dot Matrix, and inputs to the LCD.

Interrupts – Cause a process to halt until it is exited, in which case it will return to where it was before it was interrupted. The only interrupt pins used were to deduce which keypad button was clicked. It was done using Interrupts on the specific column pins and corresponding ISRs to determine which row was pressed.

Design Process Recap

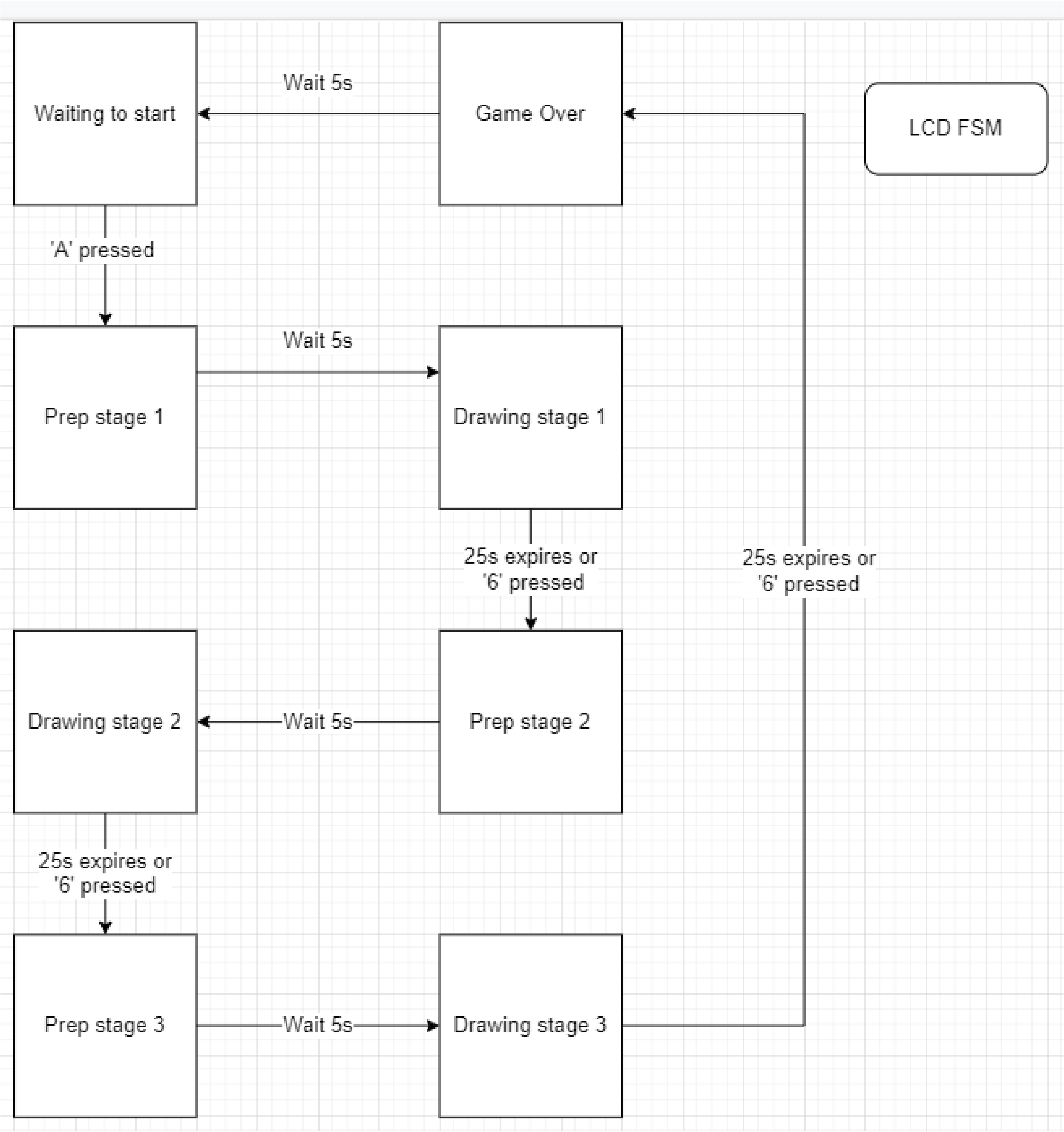
This project was initially planned to be a humidity mapping and recommendation system however due to the NUCLEO-L4R5ZI not having Wi-Fi capabilities along with testing and implementation concerns, it was switched to the current project. The first step of understanding the new system was to realize what the main functionality and purpose of that system would be. Next, physical components for the system were picked out to correspond to the inputs, outputs, and constraints defined for the system. Finally, the software and hardware implementations/integrations were done through unit testing spit between each stage of the program. These individually unit tested pieces were then integrated together.

Block Diagram

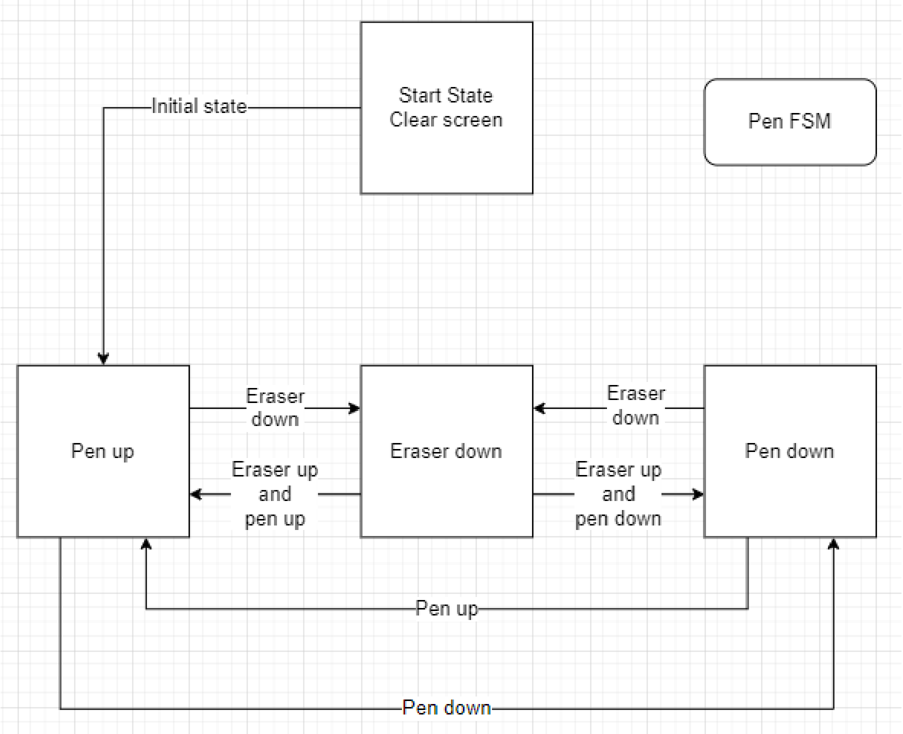


FSM Diagrams

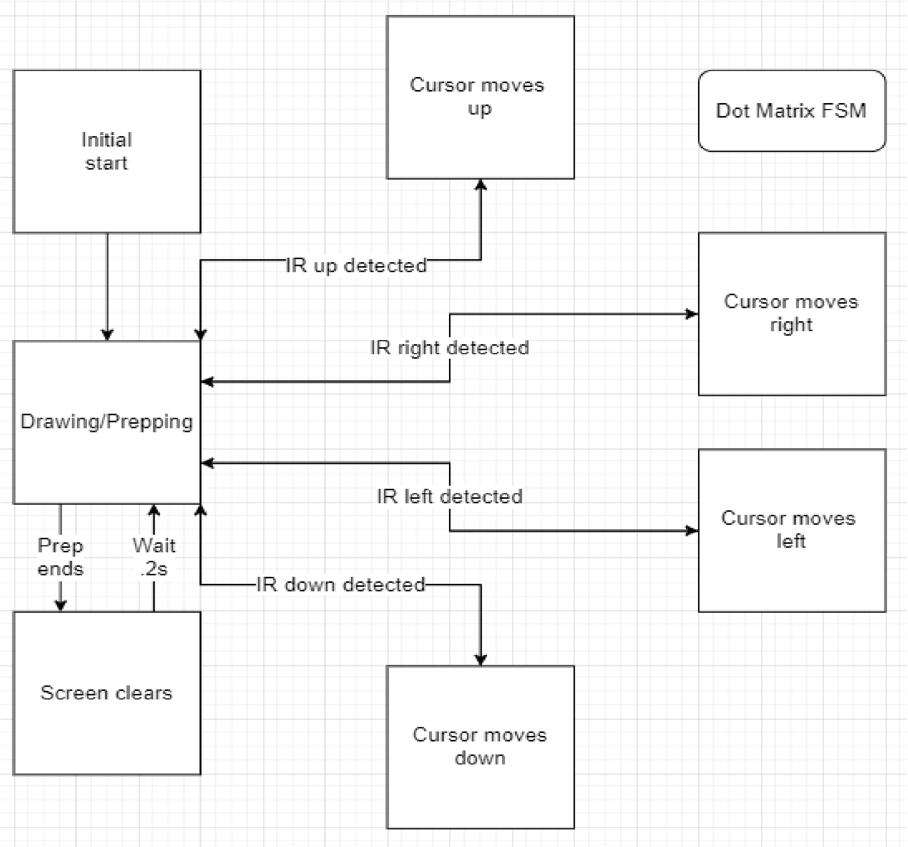
LCD



Cursor



Dot Matrix



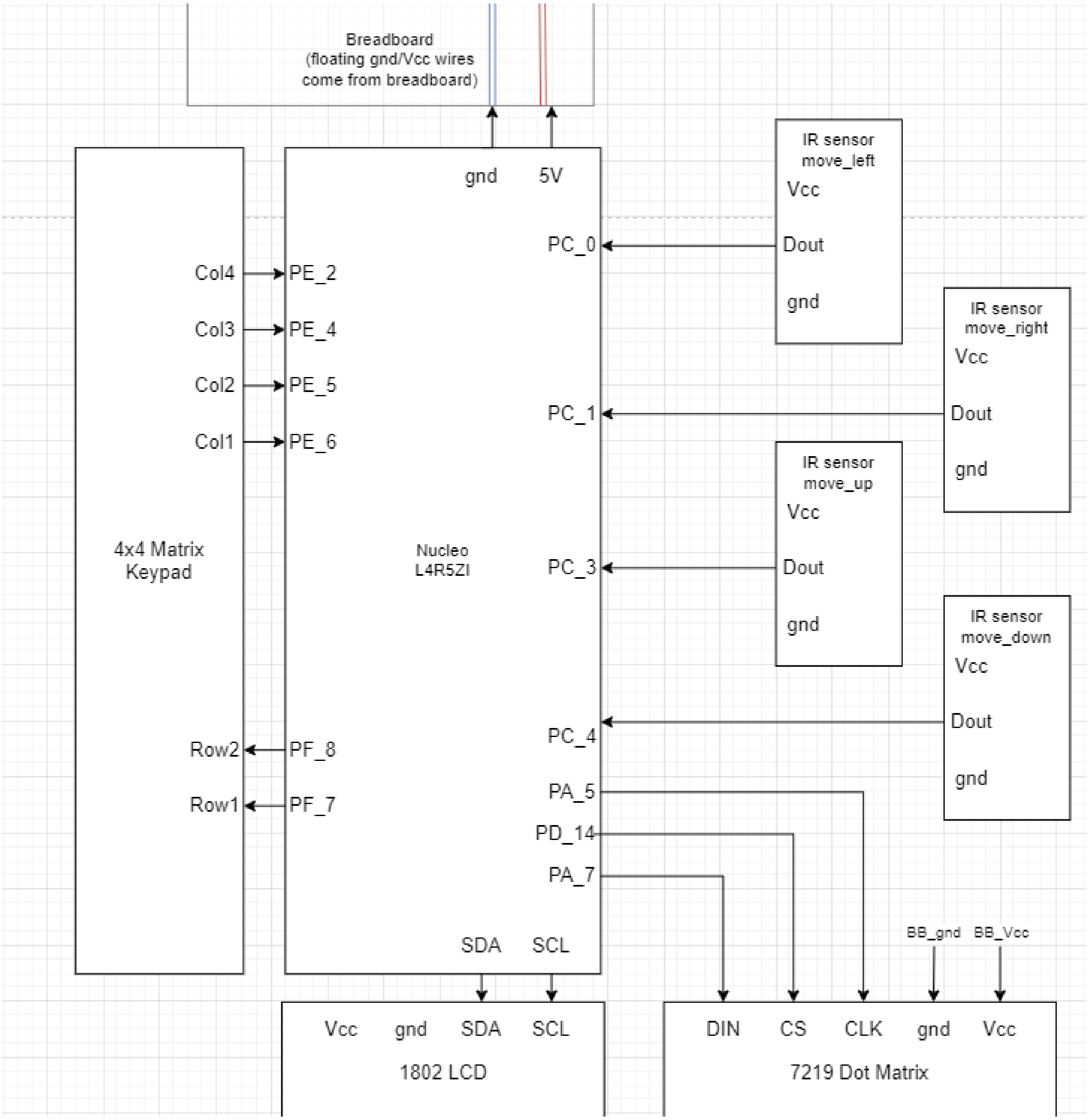
BOM

See BOM Index for digital links.

- Female to Female/Male jumper cables
- Male to Male jumper cables
- Breadboard
- 1802 LCD
- NUCLEO-L4R5ZI
- 4x4 Matrix style Keypad
- Micro USB cable
- 7219 LED Dot Matrix
- 4 IR sensors

User Instructions

Schematic



Building the system

When setting up, it is recommended to refer to the system schematic above for a visual representation.

- Connect the NUCLEO to your computer using the USB.
- Connect 3.3V or 5V to the positive rail on the breadboard
- Connect gnd to the negative rail on the breadboard
- Feel free to connect all component Vcc pins to the positive rail and gnd pins to the negative rail
- Keypad: col4 -> PE_2, col3 -> PE_4, col2 -> PE_5, col1 -> PE_6, row2 -> PF_8, row1 -> PF_7
- LCD: SDA -> SDA, SCL -> SCL
- IR sensors: PC_0, PC1, PC_3, PC_4
- Dot Matrix: DIN -> PA_7, CS -> PD_14, CLK -> PA_5

Using the system

- Depending on which direction the dot matrix is facing, the IRs will move the cursor in different directions, so it doesn't matter which IR Douts connect to which pins.
- If all the lights on the dot matrix light up when the program starts running, unplug the USB and plug it back in.
- Clean build and run.
- Keypad inputs
    - Player X wants to guess: '1', '2', '3'
    - Start game: '4'
    - Incorrect guess: '5'
    - Correct guess: '6'
    - Invert pen: 'A'
    - Invert eraser: 'B'
- Follow directions on the LCD to play a game.
- When the game starts the drawer will have 5 seconds to think of something to draw, this is the prep stage.
- At the end of the prep stage the screen will be cleared and drawing can commence.
- The drawer will have 25 seconds to draw a picture, during this time players also have the ability to guess.
- The cursor being used to draw will initially have pen up and eraser up.
- If the drawer wishes to draw, they need to put the pen down.
- If the drawer wishes to eraser, they need to put the eraser down.

- If the drawer wishes to move the cursor without altering the picture, they need to pull the eraser and pen up.
- A player will guess by pressing their corresponding player number on the keypad and making a verbal guess.
- The drawer will then confirm if their guess is correct or not.

Test Plan

System was split up into many unit testing stages. However, these were the two general testing scenarios.

Dot Matrix cursor

Before integrating the Dot Matrix with IR sensor inputs, it was tested using keystrokes. The pen and eraser were initialized as being up.

- ‘w’ – up
- ‘a’ – left
- ‘s’ – down
- ‘d’ – right
- ‘e’ – invert eraser
- ‘p’ – invert pen

Keystroke	Expected behavior/Test case
None	Cursor blinks in place
A	Move left without drawing
D	Move right without drawing
W	Move up without drawing
S	Move down without drawing
P	Pen down cursor blinks in place
A	Move left while drawing
D	Move right while drawing
W	Move up while drawing
S	Move down while drawing
E	Eraser overwrites pen
A	Move left while erasing
D	Move right while erasing
W	Move up while erasing
S	Move down while erasing
E	Pen remains down

A	Move left while drawing
D	Move right while drawing
W	Move up while drawing
S	Move down while drawing
P	Cursor moves without drawing
D	Move right without drawing
W	Move up without drawing
D	Move right without drawing
W	Move up without drawing

Gameplay

Inputs:

- ‘X’ – don’t care
- ‘L’ – left
- ‘R’ – right
- ‘U’ – up
- ‘D’ – down

Present State	Key	IR	Expected Behavior	Test case
Press 4 to start	4	X	Prep stage, R1, counts 5s	Game starts
R1 Prepping	X	X	Drawing starts	Wait 5s for the drawing to start
Drawing	X	LL	Blinking LED moves 2 L	Cursor doesn’t draw
Blinking LED	A	RR	2 solid LEDs are R	Cursor draws
2 solid LED	B	L	1 solid LED is R	Cursor Erases
1 solid LED	B	R	2 solid LEDs are R	Cursor draws
2 solid LED	1	X	Nothing changes	Player 1 guesses
1 solid LED	6	X	R1, P1 increments	Correct guess
R1, P1 increment	X	X	Prep stage starts	Round ends
Blinking LED	X	X	Matrix clears after prep	Matrix clears
Blinking LED	X	DD	2 solid LEDs are D	Cursor draws
1 solid LED	B	U	1 solid LED is D	Cursor Erases
Blinking LED	1	X	Nothing changes	Player 1 guesses
Blinking LED	5	X	P1 is decremented	Incorrect guess
P1 decremented	2	X	Nothing changes	Player 2 guesses
Blinking LED	6	X	P2 is incremented	Correct guess
R3 drawing	X	X	Round times out	Drawing timeout
1s left to draw	X	X	Lasts for 5 seconds	Game over
Press 4 to start	4	X	Test cases run again	Game restarts

Results

The system works seamlessly between the hardware and software integrated parts. All the test cases passed for both the shown unit tested portions as well as smaller units' tests. These smaller tests checked IR, keypad, and LCD behavior. The only flaw found in the system came in the form of the blinking LED moving. Because the cursor needs to blink and this blinking state contains critical sections data the thread causing the blinking LED needs to sleep for .2s while holding the mutex. This can cause some lag if the drawer tries to move across the screen very quickly. This lag can similarly be caused by the thread that manages cursor movement based on IR motion detection. The lag is a small price to pay to ensure the accuracy of the Dot Matrix state.

Recommendations for Improvement

Use of Watchdog

While the watchdog is essential for ensuring the program runs indefinitely and was a requirement for this project, it ruins the feel of the game. The MBED watchdog has a max timeout of ~32s, however a player really needs more than 32s to draw a picture, preferably greater than 60s. This max timeout cannot be changed; hence the drawer is only given 25s to draw. This program would function better on a platform with a higher watchdog.

Guessing pauses drawing stage

The way the guess is implemented currently the drawer has to keep drawing while simultaneously verifying all the incoming guesses. This is very difficult and with the already short drawing window it leaves the drawer with less time to think and draw accurately. Having the timer and cursor pause would put less strain on the guessing part of the game.

Vibration signaled guesses

In the code there is bit masking used to enable pins for the implementation of a vibrator. This functionality however was never fully implemented. Had it been, a specific vibrator would have vibrated corresponding to the player who had guessed and would continue until the guess had been verified.

Smooth drawing controls

While the game works, the drawing feels less like drawing and more like how it was tested, 'wasd'. The initial plan was to use 8 IR sensors placed in the formation of a 4x4 matrix. The IR sensors would track the drawers' hand and map the movement on to the LEDs fluidly, to mimic actual drawing. It became apparent right after testing the IRs that this would not be possible with their weak ~2cm range of detection. To make this possible different IRs would've been needed.

BOM Index

Female to Female/Male jumper cables

https://smile.amazon.com/EDGELEC-Breadboard-1pin-1pin-Connector-Multicolored/dp/B07GD312VG/ref=sr_1_1_sspa?crid=2XJQXGCJBKIVY&keywords=female+to+female+jumper+wires&qid=1639137049&s=electronics&sprefix=female+to+female+jumper%2Celectronics%2C172&sr=1-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUFWU1pVSTVMRUc1UDkmZW5jcnlwdGVkSWQ9QTA5NTAzNjkyQVQ0S0xZMFdUUEYxJmVuY3J5cHRlZEFkSWQ9QTAyMTc1NDZRQVJERDNEQzgxM1Mmd2lkZ2V0TmFtZT1zcF9hdGYmYWN0aW9uPWNsaWNrUmVkaXJlY3QmZG9Ob3RMb2dDbGljaz10cnVl

Male to Male jumper cables

https://smile.amazon.com/Premium-Breadboard-Jumper-100-Pack-Hellotronics/dp/B07GJLH7V1/ref=sr_1_1_sspa?keywords=male%2Bto%2Bmale%2Bjumpers&qid=1639137182&s=electronics&sr=1-1-spons&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExWjFVMDhTV05OMTE2JmVuY3J5cHRlZElkPUEwMTY4NDA5MTZLRThDTkhNVjczQiZlbmNyeXB0ZWRBZElkPUEwMjU4NDg4UkwzWUoxRjM1Mjg4JndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ&th=1

Breadboard

https://smile.amazon.com/DEYUE-Solderless-Prototype-Breadboard-Points/dp/B07NVWR495/ref=sr_1_1_sspa?keywords=breadboard&qid=1639137081&s=electronics&sr=1-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEzMUxaMVBSM0daQTdLJmVuY3J5cHRlZElkPUEwNjI0MzQyMzNMNjVDTTBIQzQ2RyZlbmNyeXB0ZWRBZElkPUEwODA0MjYyUldVVVowSDhIV1VFJndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ==

1802 LCD

https://www.mouser.com/ProductDetail/713-104020111

NUCLEO-L4R5ZI

https://www.mouser.com/ProductDetail/511-NUCLEO-L4R5ZI

4x4 Matrix style Keypad

https://www.amazon.com/Matrix-Membrane-Switch-Keyboard-Arduino/dp/B07THCLGCZ/ref=sr_1_5?keywords=4x4+matrix+keypad&qid=1639311353&sr=8-5

Micro USB cable

https://smile.amazon.com/AmazonBasics-Male-Micro-Cable-Black/dp/B0711PVX6Z/ref=sr_1_3?keywords=micro+usb+cable&qid=1639137130&s=electronics&sr=1-3

7219 LED Dot Matrix

https://www.amazon.com/ACEIRMC-MAX7219-Display-Single-Chip-Control/dp/B08VHX2KC4/ref=sr_1_8?crid=276F517QCP371&keywords=arduino+dot+matrix&qid=1636404440&qsid=137-0148648-7603540&sprefix=arduino+dot+matrix%2Caps%2C59&sr=8-8&sres=B07FFV537V%2CB08KS68GYZ%2CB07KQ31FX6%2CB00LSG54O2%2CB08VHX2KC4%2CB07T28MSP5%2CB07X4W3MVD%2CB08CSJNGJL%2CB088LWLHP5%2CB08SQT9GGB%2CB089752GYX%2CB01567H3PG%2CB079NJLGMG%2CB01D8KOZF4%2CB08745JBRP%2CB074WMHLQ4

4 IR sensors

https://smile.amazon.com/HiLetgo-Infrared-Avoidance-Reflective-Photoelectric/dp/B07W97H2WS/ref=sr_1_37?crid=2ZECVUH3BF3V9&dchild=1&keywords=arduino+sensor&qid=1598036691&s=electronics&sprefix=arduino+sen%2Celectronics%2C165&sr=1-37