WordPal – Standalone Handheld Daily Word Game Implementation

Nala Rivera, Hilda Deveaux, Daniel Ortiz-Gratacos

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — As we progress more into the modern era, technology has become increasingly distracting. Our device, WordPal, aims to combat that by allowing the user to play a popular daily 5-letter word guessing/deduction game without distraction. Built for those who want to unwind from social media but still wish to partake in their daily ritual, WordPal streamlines gameplay by providing the user with a dedicated tactile keyboard, as well as an innovative feedback system that lights up the keys using LEDs to inform the users of what letters they've exhausted or are correct about. WordPal also includes Wi-Fi capability, so users may play the same game as millions of others around the globe; for additional fun, users can also generate as many words as they want entirely on device.**

***Index Terms* — LED lamps, wireless fidelity, liquid crystal displays, microcontrollers, regulators.**

I. Introduction

Millions of people around the world partake in the same daily ritual: playing the game Wordle. The rules are simple to understand: you have 6 chances to guess a 5-letter word by inputting other 5-letter words. Players receive feedback in the form of the game coloring the letters of their input word; letters turn yellow when they are present in the correct word but in the incorrect position, and green when they are in the correct position in the word.

Wordle has turned into a social phenomenon due to its simplicity, low time commitment, and its shared daily words, and it leverages this by allowing users to post their scores online in a 5x6 grid that displays the feedback the user received during their game. This was the key to Wordle's initial popularity; the ease of posting scores led to users trying to "outscore" their friends, cascading into millions adopting Wordle as the aforementioned daily ritual it has now become.

However, in the modern era, social media has become increasingly distracting and toxic. Even since the relatively recent rise of Wordle, we have seen a drastic shift in this regard. Therefore, our group deigned to create a device that would allow users to participate in the peaceful pastime of Wordle without hinging their enjoyment of the game on their participation within these increasingly toxic mediums. We desired to maintain the core elements of Wordle, being the game itself, feedback system, and score sharing.

Our resultant device, WordPal, brings the Wordle game format a new life in its easy-to-use form factor, with physical keyboard buttons like those of the previous generation of mobile phones, and an easy-on-the-eyes LCD screen to display game information. The major innovation of WordPal is the presence of 26 LEDs, one behind each key, that light up and display the current best feedback for each individual letter.

II. Hardware Components

As previously mentioned, WordPal implements an LCD screen, a keyboard, and 26 LEDs. In order to drive these elements, we also need a microcontroller unit, as well as a battery-driven power system in order for the device to be portable.

A. Microcontroller Choice

When choosing a microcontroller, we determined that the most important feature to hinge our choice upon was the capability of Wi-Fi connectivity. If we desire to use the real daily word to maintain participation in the social aspect of the game, we need to obtain the word itself over the Internet. With this caveat in mind, our choice had essentially been narrowed down to a microcontroller in the Espressif ESP32 family. ESP32 microcontroller units are well-represented in the hobbyist space, in addition to being well-documented and having many supporting libraries that make developing around it significantly easier. We went with a modern chip in the family, the ESP32-S3-WROOM-1-N8R8, as it has a built-in Wi-Fi antenna as well as ample flash memory for us to store data such as the list of valid guessable words, graphics memory, and previous user scores. The ESP32-S3-WROOM-1-N8R8 is also available on a development board package from Espressif, which significantly eases prototyping before the assembly of the final PCB.

B. Display Choice

As previously discussed, we chose to use an LCD (liquid crystal) display for WordPal. We considered using an OLED (organic LED) or even an E-ink panel for the device, but at the display size of 4 inches diagonal that we were targeting, the price of these displays was out of the question; this was especially true for E-ink, as any E-ink display with a reasonable refresh time or the appropriate

color variety would run us to almost double our final cost of materials. A TFT (thin-film transistor) LCD display was therefore our best bet, which was not necessarily a bad thing, as these displays are quite easy to interface and work with, especially partnered with the ESP-32.

Our specific display module of choice was the MSP4021, a 4-inch TFT LCD with a resolution of 480*320. The display is driven by the ST7796S display controller, which is interfaced with through SPI (Serial Peripheral Interface) from our ESP-32. We opted to mount this screen in a "portrait" orientation, as this would allow the space to be used as effectively as possible for the game.

C. Lighting

The innovative feedback lighting system uses Inolux IN-PI15TAT5R5G5B addressable RGB LEDs. This model was chosen because of its conveniently small size, allowing us to fit LEDs quite close to our key switches and lessen the gap between. Additionally, this model is addressable, and can be "daisy-chained" with many more LEDs of the same model on the same data line; the LEDs are functionally shift registers. This meant we would only need to dedicate 2 IO pins (an output, and an input to complete the loop) from the already busy ESP32 towards our LED system, allowing us more lines to design other functionality.

Due to the innate functionality of standalone LEDs, we decided to denote unused letters as red. Traditionally, the unused letters are simply denoted as a dark gray color, compared to the light gray of untested letters. Initially, we considered having one of these two conditions being represented as white, but this would end up using extra power, as it would require us to use the blue channel of our LED. Therefore, we went with red, as we already needed to dedicate power to the red channel due to the usage of yellow (displayed using combined green and red channels) for another type of hint.

The LEDs are also the only component of WordPal which is powered using 5 volts.

D. Serial Communication

WordPal implements serial communication over USB for both internal and user-facing purposes; for the latter, there is functionality allowing for users to share their daily scores over a serial format to facilitate participation in the social aspect of the game in a simpler manner. The USB port itself is a female USB-A 2.0 port, which only contains 4 pins: 5V power, ground, and a differential data pair.

Communication between a USB host and WordPal's ESP32 microcontroller is facilitated by the Silicon Labs CP2102N chip, which is the accepted standard for interacting with ESP32 devices over a serial connection, being a drop-in successor to the classic CP2102.

III. Power System

A. Battery

The batteries for WordPal are two rechargeable and replaceable NiMH (nickel-metal hydride) cells, in a standard AA form factor. These cells have a voltage of 1.2 V, which they maintain with far more consistency than traditional alkaline AA batteries maintain their rated 1.5V. The two cells are wired in series to give us a relatively stable 2.4 V, which is converted into voltages usable by our subsystems.

The initial choice for this device was a lithium-ion battery system, as well as an on-device battery charging system. However, due to additional safety constraints and other related inconveniences, this was unable to be brought to fruition; general restrictions on battery charging systems, as well as the usage of a USB-A 2.0 port, also hindered and eventually halted the development of an internal NiMH battery charging system.

B. 5-Volt System

As previously mentioned, the only subsystem of WordPal powered using 5 volts is the LED feedback system. This required the design and implementation of a separate regulator system for the LEDs alone, which was designed to facilitate a maximum output of 400 milli-Amps (2 watts of total power) to support all channels of all LEDs being turned on at the same time (in an absolute worst-case scenario). The design was built around the TPS61023 boost regulator IC from Texas Instruments.

C. 3.3-Volt System

The 3.3-volt rail on WordPal is built to provide up to 650 milli-Amps (roughly 2.15 watts of total power) to the remainder of systems on WordPal. Anything close to the full capacity is only necessary in occasional bursts, however, as the ESP32-S3's power consumption significantly increases during Wi-Fi connectivity, using up to 300 milli-Amps during this period. When Wi-Fi is inactive, the device passively consumes roughly 200 milli-Amps from the 3-volt line to power the screen, its backlight, and the ESP-32's other functionalities.

D. Safety Features

WordPal comes equipped with two polyfuses, one for each voltage regulator circuit, that prevent overvoltage or overcurrent drawn from the batteries. These both reside past the system power switch.

IV. Software

A. Use Case Diagram

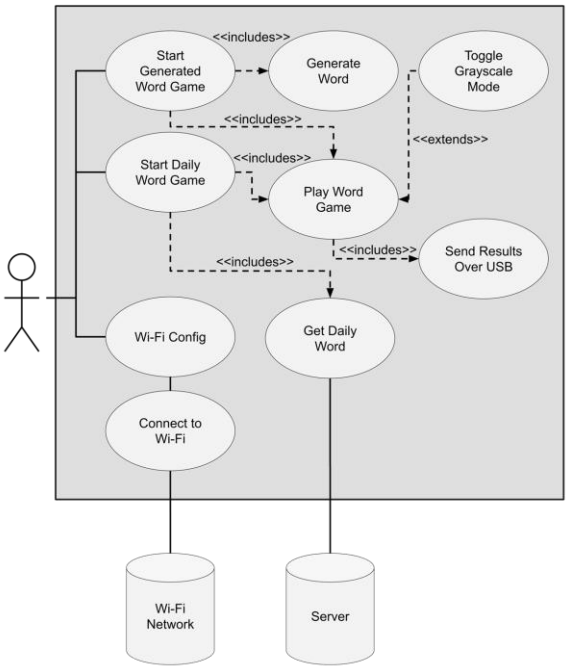

Fig. 1. WordPal use case diagram.

The Use Case Diagram has three relevant actors: the User, Wi-Fi network, and Server. The user is able to start a generated word game, start a daily word game, do Wi-Fi configuration, play word game, and toggle grayscale mode. The system connects to a Wi-Fi Network, gets a daily word from the server, generates a word, and send results over USB although the user does have some control over when it sends it since it waits until the user presses a key. Starting the respective games each involve getting the relevant word relative to their type and then playing the word game. The grayscale mode isn't a strictly necessary feature to implement, but the user is able to toggle between grayscale mode being on or off on the first round which affects how the feedback works on the screen and the LEDs. There are other automatic processes that take place that aren't mentioned here due to the nature of a use case diagram.

B. State Diagram

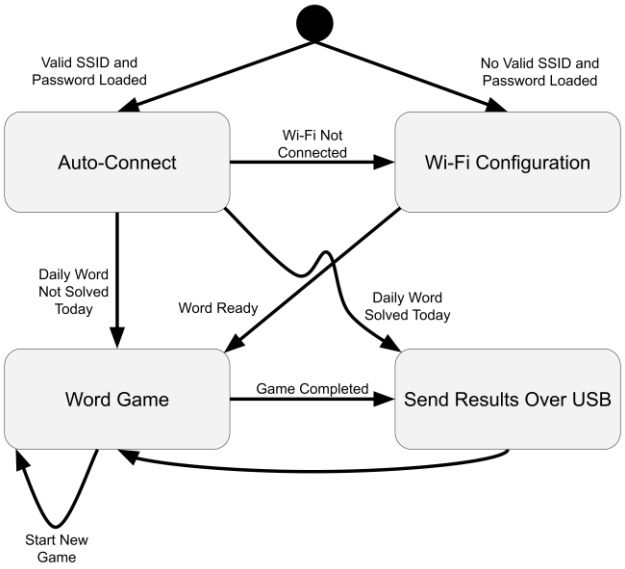

Fig. 2. WordPal state diagram.

The state diagram has 5 states: the initial state, Auto-Connect, Wi-Fi Configuration, Word Game, and Send Results Over USB. The initial state goes to either the Auto-Connect state if valid SSID and Password are loaded (this is implemented via loading saved Wi-Fi config information from flash as well as checking to see if the lengths of SSID and Password are valid in some sense) or to the Wi-Fi Configuration state if not. From the Auto-Connect State, it can go to Wi-Fi Configuration if the Wi-Fi is not connected, or in the case where the Wi-Fi is connected, it could go to Word Game if the daily word is not solved today or Send Results Over USB if the daily word is solved today. In relation to checking the solved status of daily word, there are three checks. The first check would be if the daily word isn't solved. The second would be if the daily word solved date matches the current date, and the third one would be if it doesn't and if the last solved date isn't equal to an empty string (although the extra latter condition of the third check may not be necessary). WordPal only time syncs when Wi-Fi is connected which usually would be to the current time.

From Wi-Fi Configuration, it goes to Word Game if the word is ready. This can happen in different ways. The user could get to a generated word game via holding the enter key. It could also go to a daily word game if the Wi-Fi was connected and the daily word was obtained. However, if something happened where the daily word wasn't obtained, the code would continue to a generated word game. From, the Word Game state, a new game could be started by holding the enter key or it goes to the Send Results Over USB state after game completion. Though not strictly necessary, it would wait until the user presses an input, and

then it would send the serial results before going to a generated word game which it does automatically.

C. Software Flowchart

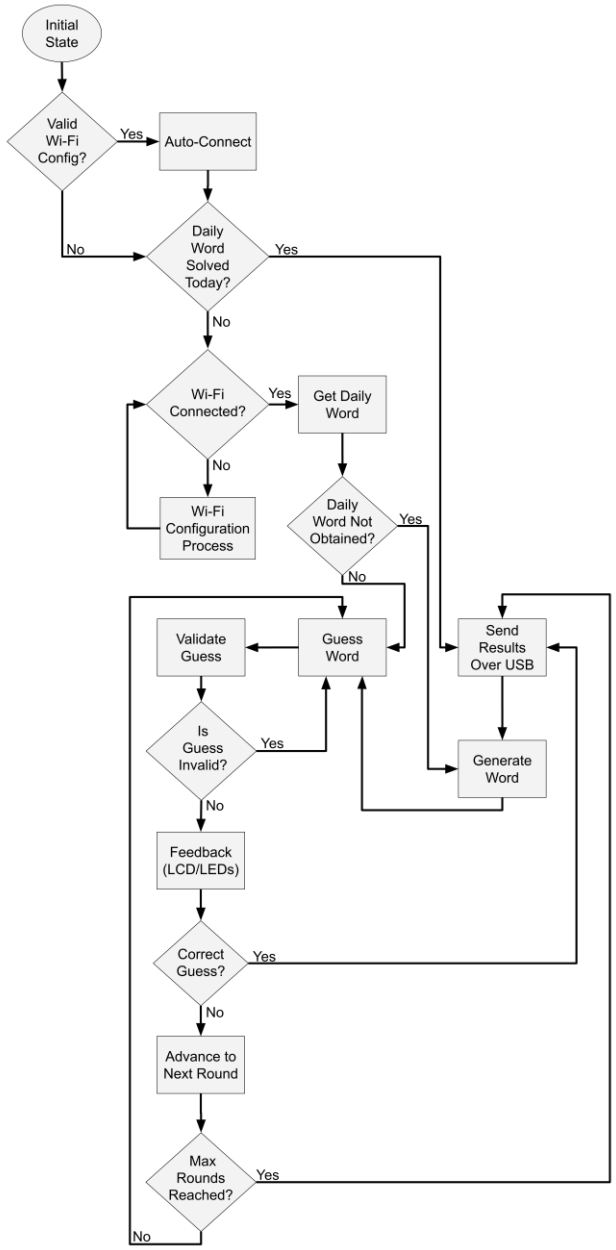

Fig. 3. Software Flowchart.

The software flowchart shows a rather high-level overview of some of the basic flow of WordPal's software. There is some overlap with the state diagram. For the loop relating to the Wi-Fi Configuration Process and Wi-Fi Connected, essentially the user would continue being taken through the Wi-Fi Configuration process until the Wi-Fi is connected (or if the user holds the enter key to go to a generated word game). When it comes to a game, the user would guess a word and then the guess would be validated. If the guess is invalid, the user would guess a word again otherwise it would continue to feedback in the LEDs and LCD. Then it would check for the correct guess which then it would go to send results over USB if the guess was correct. Otherwise, it would advance to the next round, and then another check would be done to see if the max amount of rounds were reached. If yes, then it would send results over USB otherwise the user would guess a word again.

D. Class Diagram

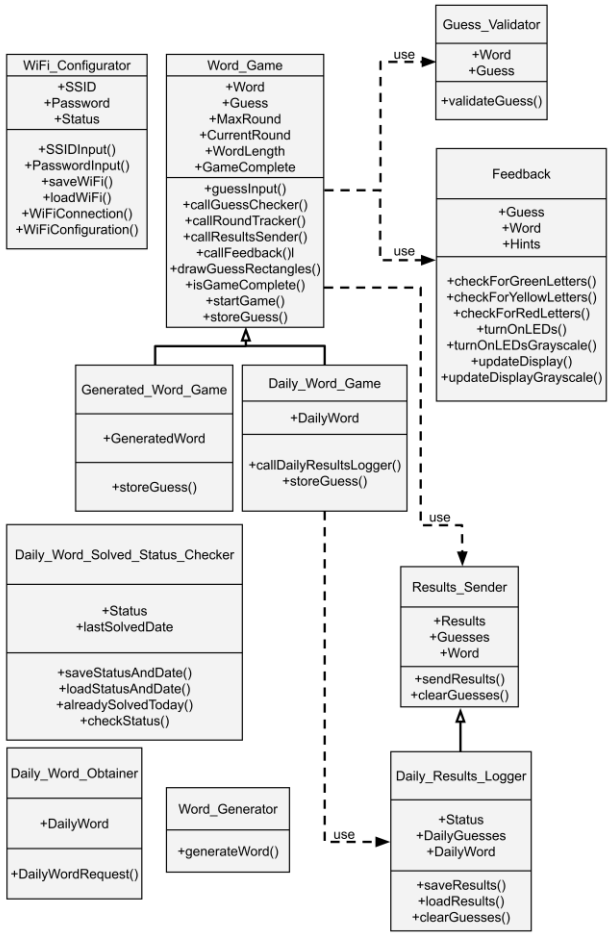

Fig. 4. Class Diagram.

The class diagram is simplified since there are more classes, attributes, and methods than what is listed here. Wi-Fi_Configurator handles SSID and Password Input, the saving and loading of Wi-Fi config info, and Wi-Fi Connection. Word_Game handles guess input and uses Guess_Validator, Feedback, and Results_Sender. It keeps track of rounds, checks if the game is complete, and does the initial conditions for a new game. It also draws the rectangles that the letters of the guesses go in.

For Generated_Word_Game, its main difference is that it adds GeneratedWord as an additional attribute. For Daily_Word_Game, its main difference is it adds DailyWord as an additional attribute, adds the method callDailyResultsLogger(), and unlike Generated_Word_Game, it changes storeGuess() to work a bit differently. Guess_Validator checks the validity of a guess by checking against word arrays (the word arrays are AnswerWords and ValidWords). Feedback allows for the relative correctness of letters to be checked and to affect the UI in terms of LEDs and display to give the relevant information, either in color or grayscale.

Results_Sender allows for results to sent and for their attributes to be cleared. Daily_Results_Logger adds the attributes Status, DailyGuesses, and DailyWord. It also adds in its methods a way to save and load results (more than the attribute results) as well as its own version of clearing its attributes. Word_Generator generates words which is done by returning a pseudorandom selection from AnswerWords, and Daily_Word_Obtainer obtains the daily word from a server. The Daily_Word_Solved_Status_Checker can save the status and date as well as load it, check if the current date is equal to the last solved date as well as check status.

When it comes the class diagram, the relationship types used were use dependencies and inheritance. The dependencies could be rather indirect at times, but it perhaps made sense especially if one considered it from the perspective of if a class was changed, would the dependent class change or break. There were potential dependencies and interactions relating to control flow, but the class diagram maybe isn't intended to deal with such things. Also, all the attributes and methods of the classes are public.

E. Programming Environment Choice

Programming environment is defined in this context as the set of software things which allow for software development (it could also involve hardware). There were three options: Arduino Core for the ESP32, ESP-IDF, and MicroPython. The relevant criteria included ease of use, programming language support, library support, performance, and suitability for prototype development. Arduino Core for the ESP32 was chosen out of the three. It maybe struck a good balance in allowing for beginner friendliness and reasonable performance. The downside of more mixed quality libraries on average didn't factor much, and Arduino in general being suitable for rapid prototyping maybe applied well for WordPal.

F. Display Library Choice

For the display library, there was TFT_eSPI, Adafruit GFX, and LovyanGFX with the relevant criteria of ease of use, feature set/capability, and performance. Feature set/capability was something that they all could be considered satisfactory for WordPal even if there was variance in what they had access to. Adafruit GFX in terms of performance would have maybe been too much of a hinderance for WordPal. TFT_eSPI maybe was overall easier compared to LovyanGFX, and though it has lower performance than LovyanGFX, for WordPal it's enough. TFT_eSPI was chosen, but it could have been switched if it didn't work out, but that ended up not being necessary.

G. LED Control Choice

For LED control, there was FastLED, Adafruit NeoPixel, and pololu-led-strip-arduino with the relevant criteria being LED compatibility, ease of use, and performance. FastLED did have the edge over the others in terms of LED compatibility. Using FastLED for WordPal is doable. The performance part was rather speculative due to perhaps a lack of proper information on the subject, but it's maybe fine for WordPal. FastLED was selected, and it could have also been switched out as well, but there may be no need to.

H. Wi-Fi Software Stack and Optional Layers Choice

This one was a bit different from the others. Due to the use of the Arduino Core for the ESP32 from earlier, Arduino-esp32 WiFi.h, a full native Wi-Fi stack, has to be used since the usage of another would be wasteful and redundant. So, the selection here was to use it by itself or with another optional layer. The other two options are WiFiManager and AsyncTCP, and the relevant criteria are ease of use, feature set, and dependencies. When it comes to ease of use, the optional layers would maybe add extra difficulty. In AsyncTCP's case, it would have added unnecessary complexity for no practical gain. In WiFiManager's case, there is potential for WiFiManager integration to be an additional feature. One thing it could allow for is to configure WordPal's Wi-Fi with another device which could have some utility. There could have also been some potential utility in relation to testing though that ended up not being necessary. In the matter of dependencies, it would go against WiFiManager since it requires other libraries. WiFi.h was used without optional layers though some supporting libraries were used like HTTPClient.h [1]. While WiFiManager could be integrated, it's not necessary, and it likely won't be at this stage of the project.

I. Graphical User Interface

Fig. 5. Graphical User Interface Screens.

The image above shows similar screens to how WordPal's screens will be which are the Auto-Connect Screen, Wi-Fi Configuration screen, and Game Screen when grayscale mode is off and when it's on. There are minor differences between the screens here and the UI in WordPal. For instance, the colors in WordPal are brighter and more vibrant due to the display and backlight. Also, the tft.color565() function changes 24 bit color values to 16 bit color values which means the colors could be somewhat different that way (although the effect it would have would be making it darker if not exactly representable in RGB565, so the display and backlight have a bigger effect) [1] [2] [3] [4]. There is an overlay, but there is still a gap between how it looks here and how it looks on WordPal [1]. WordPal's GUI is in portrait mode, and the style is of a minimalist variety and a dark mode leaning. The font in the image is Arial which is similar to how the font (particularly size 4) is on WordPal.

There would also appear black text with a white background on the lower portion of the screen sometimes relating to different things like communicating that valid Wi-Fi config has loaded. The font size of this text is 2, and the font type looks different from how text looks in font size 4.

On the Wi-Fi Configuration screen, there is a blinking cursor that appears after text relating to the field the user is on (SSID or Password). The text should also not go out of the bounds of the field. The latter characters of the string would be the ones that appear in the field when the string is longer than what can be shown.

On the game screen as mentioned before, when grayscale mode is off, the feedback on display would be similar to New York Time's Wordle with the exception of red being used for incorrect letters instead of a shade of gray (and also in regards to the letters being lowercase). When it comes to grayscale mode, the lighter the shade of gray, the more correct it is. When it is the lightest shade of gray, the color of the text is black which aids in readability and also makes it be more distinct. The color of the more middle gray is the same color as the border which could be another way to distinguish the "yellow" letter since the border is no longer distinct.

V. Hardware Implementation

A. Keyboard and Lighting

The keyboard component of WordPal is implemented in the traditional QWERTY keyboard layout, with three rows of keys: the standard letters comprising the top two, and the bottom row containing two extra keys, Enter and Backspace. These are Omron B3U series switches, chosen for their small size and profile.

The keys being arranged in 3 rows is of significance not only to the physical board design, but also in the wiring implementation. As can be observed in Fig. 4 below, the keys are wired in a matrix with 3 rows and 10 columns, with the last column reserved for the extra 10th key (the letter P) in the first row.

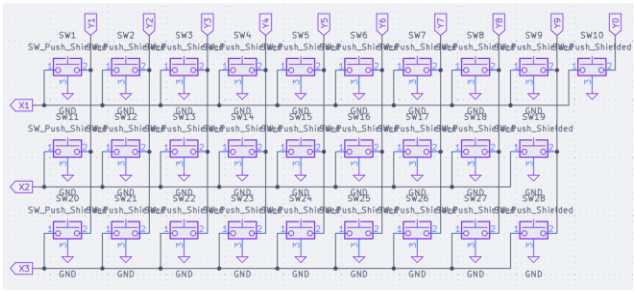
Fig. 6. WordPal keyboard schematic, arranged in the 3x10 matrix layout.

The matrix is the standard for keyboard implementation, as it allows us to reduce the 28 pins that would be needed to poll each key switch individually to as few as 11 (7 rows and 4 columns). Instead of taking this theoretically optimal 7-by-4 layout, it was instead chosen to use a 3-by-10 matrix in order to ease PCB layout design.

The matrix layout functions by using the ESP32's internal resistors to pull each column (labeled with Y#) to a high logic level, and pull one row (labeled with X#) at a time to ground level. The process of pulling each row to ground is known as "scanning" the matrix. When a key on that row is then pressed, the corresponding column will be connected to that row and therefore shorted to ground. The column pins, which are marked as inputs on the ESP32, will recognize the short and use the corresponding row and column to determine which key has been pressed.

In between rows of keys lie the RGB LEDs. As previously discussed, these LEDs are addressable and therefore only occupy two pins on the ESP32. Each LED has a DIN line attached to either the microcontroller (for the very first LED in sequence) or the previous LED's DOUT; each DOUT line is either connected to the next DIN or back to the ESP32 (for the very last LED). There are 500 Ohm resistors at both pins of the MCU to control current flow, as well as one 0.1 micro-Farad capacitor on each row's power line to mitigate frequency interference.

This configuration is demonstrated in an identical implementation in Fig. 5 below, albeit with only 3 LEDs in the chain to ease readability; the full design has all 26 LEDs in one chain.

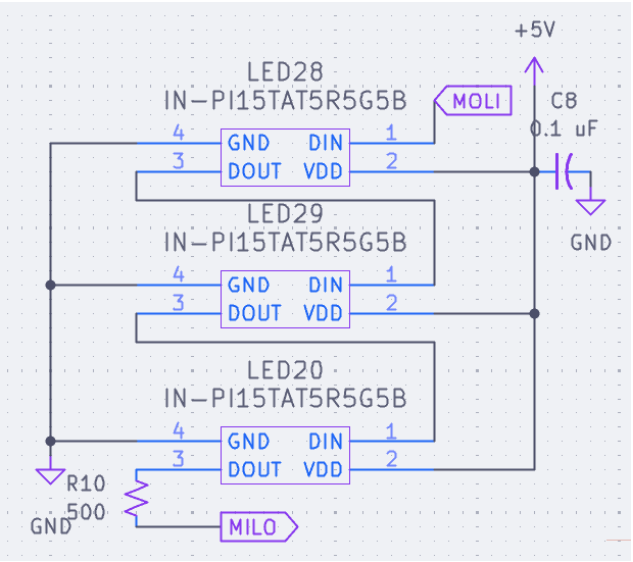

Fig. 7. Example daisy chain consisting of only 3 LEDs.

The below figure displays the final PCB implementation of the keyboard and LED systems. The three blue traces on the left are the key row lines, the first 10 red traces leading upward are the key column lines, the last two red traces are the LED loop, and the fourth blue trace is 5-volt power for the LEDs. The switches need no power of their own.

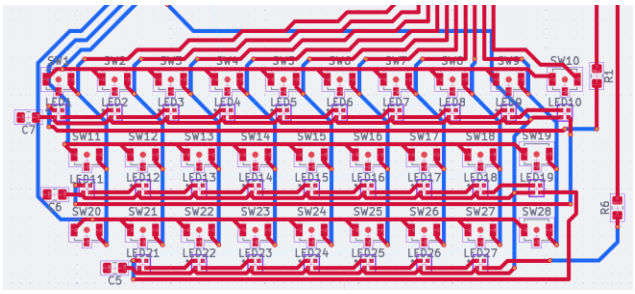

Fig. 8. WordPal keyboard and LED PCB layout.

B. Display

The display is interfaced with through SPI (Serial Peripheral Interface), as mentioned before. This project relies on the TFT_eSPI graphics library, which allows for easy drawing of shapes, fonts, and images on our LCD screen. A 9-pin header connects the display to the main board, as well as providing the display controller and backlight with 3.3-volt power.

VI. Enclosure

A. Overall Enclosure Structure

The enclosure of WordPal is designed as a two-part system consisting of a top case and a bottom case. This design allows the device to be assembled securely while keeping the overall form compact and easy to use as a handheld device. The PCB and all internal components are placed between these two parts and are held in place once the enclosure is screwed together. This layout helps keep everything aligned while also protecting the electronics during use. It also makes the device feel more solid overall when it is fully assembled.

The top case acts as the main user facing side of the device and is responsible for how the user interacts with WordPal. It includes the display opening as well as cutouts for components that need to be accessible from the outside. The bottom case serves as the base of the enclosure and provides the main structural support for the system. It also contains holes needed for fastening, which allows the two halves to be secured together with screws. Together, these two parts create a closed enclosure that keeps the internal components protected while still allowing for easy assembly.

This two-part design also makes the enclosure easier to work with during development and testing. If adjustments are needed, each half can be modified or reprinted without affecting the entire enclosure. This is especially useful

when refining component fit or making small changes to the layout. It also simplifies assembly, since the internal components can be placed into the bottom case and then secured by attaching the top case. Overall, the enclosure was designed to keep the device stable, protect the internal components, and allow for straightforward assembly while maintaining a clean and functional layout. This also helps improve durability during regular use, since the enclosure acts as a protective barrier for the electronics while the device is being handheld.

Fig. 9. Enclosure Design

B. Display Integration

For this design, one focus was designing the enclosure to accommodate the LCD module. The display sits above the PCB due to its existing standoff configuration, which introduces a height that the enclosure must account for. This required ensuring that the top case had enough internal space to avoid interference with the display when the enclosure is assembled. If this spacing is not considered, the enclosure may not close properly or could make contact with the display.

The top case includes a display opening that allows the LCD to be visible while keeping the rest of the PCB inside the enclosure. By designing around the existing display configuration, the enclosure supports proper fit and positioning without requiring changes to the PCB or display module. Additional clearance was included in the opening to account for small variations in 3D printing, which helps ensure that the display can be placed into the enclosure without forcing it into position. This makes assembly more consistent and reduces the risk of damaging components.

C. Key Integration and User Input

Another important consideration in the enclosure design was integrating the key switches with the top case. The switches are arranged in a fixed layout on the PCB, so the top case was designed to match their exact positions. The top case includes individual cutouts for each key, allowing keycaps to extend through the enclosure and be pressed by the user.

Rather than being part of the enclosure itself, the keycaps pass through the top case and are accessed directly by the user. The letters and special characters corresponding to each key will be integrated into the keycaps, which keeps the labeling clear and consistent. This allows the enclosure to focus on providing structure while the keys handle the user input.

The cutouts must be aligned carefully with the switch layout to ensure that each key press is properly registered. If the openings are not positioned correctly, it can affect how the keys operate or make them harder to press. Each opening also includes enough clearance to allow the keycaps to move freely. Overall, the enclosure is designed to support the key layout by providing proper spacing and accurate user input. This helps ensure that the keys function as intended while maintaining a consistent and organized interface.

References

[1] ChatGPT. OpenAI. Accessed: Mar. 25, 2026 [Online]. Available: https://chatgpt.com/

[2] "RGB565 Colour Reference" Chips'nCode. Accessed: Mar. 26, 2026 [Online]. Available: https://chipsncode.com/tools/rgb565-colour-reference/?utm_source=chatgpt.com

[3] "Convert and Display Color Images on an Arduino TFT Screen" Instructables. Accessed: Mar. 26, 2026 [Online]. Available: https://www.instructables.com/Convert-and-Display-Color-Images-on-an-Arduino-TFT/

[4] "RGB bit mask help" Github. Accessed: Mar. 26, 2026 [Online]. Available: RGB bit mask help · micropython · Discussion #11518

Declaration of AI Use

ChatGPT contributed to the development of the software section of this paper.