The GameFrame: Portable Gaming Solution

Allen Chion, Frank Weeks, Israel Soria, Levi Masters

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

***Abstract* — The GameFrame is a portable gaming device designed for those who want something more than tapping a screen or keyboard in their gaming experience. It provides a more physical way of interacting with a game, which is something usually restricted to arcades. This device is a stand alone gaming board capable of, at minimum, chess. It can be played with two players against each other or one player against an AI. The GameFrame is intended to be an enjoyable on the go experience. This machine utilizes an 8x8 grid of buttons that displays the current status of the game and also allows the user to make moves by pressing them. The GameFrame has the versatility for additional games to enhance the user experience. The additional games only need to be implemented via software configuration.**

***Index Terms* — Artificial intelligence, Embedded Software, Lithium Batteries, Neural networks, Regulators.**

I. Introduction

Passing time in the age of the smartphone has become somewhat simplistic and monotonous; open an app, tap the screen in a few places to cycle through content, receive exclusively audio and visual feedback, rinse and repeat. We seem to find ourselves in these cycles of content absorption without any real stimulation beyond what is experienced when awarding a "like" of some kind to a digital post. Gaming on these flat screened devices can also lose a lot of its charm, as all one really experiences is the display of a make-believe game layout and whatever sound the speakers play when a piece is moved. In the modern world, not enough attention is given to the smaller details in gaming and handheld entertainment. Game hardware neglects details such as haptic feedback from button pushing and discrete modules for each individual piece in a game rather than everything blending into one big screen.

The GameFrame aims to break some of the monotony of mobile entertainment. With physical buttons that actually move and respond as they are pressed and real physical modules corresponding to the movable areas of a game, the GameFrame has a much more interactive feel as a device compared to a flat screen. The GameFrame is a device that can host a game such as chess for multiple players or even for player versus computer. The device being portable, lightweight, and with sufficient battery life can be used as the primary source of entertainment during travel, or other periods of downtime. The GameFrame also does not have loose pieces to be lost in transit, and cleanup is as simple as putting away a book! Being so simple and robust, the GameFrame hopes to establish itself as an essential item for those who want a different gaming experience.

Fig. 1. Jubeat arcade machine.

A game that inspired some of the design of the GameFrame is a rhythm game called "jubeat." This game plays music and displays patterns corresponding to the music that the player must tap in rhythm. This creates an experience with immersive and sensational appeal. When the game was released into a mobile platform, it seemed to lose most of its charm. The game became dull and the major factors of physical touch response disappeared. This is exactly what the GameFrame hopes to add back into the gaming experience - the enjoyment and stimulation of a real physical game.

As a portable gaming device, the GameFrame is lightweight, affordable, and has a long battery life. Using the Jetson Nano to run the game, GameFrame has enough computational power to run different games and AI. The main challenge we faced when designing GameFrame's ability was cost. Durable housings, high density batteries, and lightweight components all tend to cost more as they move up in performance. Ultimately, the development model and prototyping means we spent more overall, as components were bought individually and manufacturing cannot take place at large scale. That is, we were unable to

acquire many parts at wholesale prices. The end product could have costs cut substantially as the manufacturing process is improved and parts are bought in bulk.

As far as environmental impacts, the Game Frame contains a battery, as well as computer chips utilizing silicone and electricity. One of the more popular batteries on the market that are rechargeable are made of lithium-ion, which can cause fires under stressful heat conditions. If disposed of properly or if we utilize one with gel electrolytes instead of liquid, we are able to alleviate some of the environmental impact. The GameFrame aims to both eliminate the need for a larger battery as well as reduce its overall power consumption by running highly efficient hardware components and utilizing efficient programming practices. With good disposal and recycling practices, our portable device's environmental footprint can be accounted for. Environmentally conscious alternatives will always be a consideration as they present themselves.

II. Development Environment

The software for this project runs on the NVIDIA Jetson Nano developer kit (series 945-13541-0000-000) - an especially powerful kit specializing in small AI and machine learning for embedded development environments. This kit is specialized for running serial AI type programs which we utilized in some of our player versus computer modes for the games run on the GameFrame. Though it is probably possible to run the GameFrame on a lower performing chip, we, the developers, wanted to get some experience with this particular variety of chip.

The Jetson Nano uses the "Compute Unified Device Architecture" that we want to get experience with. The CUDA is NVIDIA's own proprietary platform and API model that is used in their devices. So, experience in CUDA allows for future programming on other NVIDIA devices and future projects that may make more full use of the CUDA. CUDA also compiles programs from C, C++, Fortran, Python, and Matlab with some added commands. Picking up the language is more about the method of coding as opposed to learning a whole new language from scratch. The Jetson Nano comes preloaded with Ubuntu linux on it, facilitating an accessible programming environment.

CUDA allows for high optimization of parallel computation in graphical based processors. This is especially useful in this particular project, as we programmed in games that have up to 64 differentiated sections during play. Lots of parallel computation is useful for either backtracking or training an AI to play the games, for example.

III. Game Engine

The game engine generates a new instance of the game on call of the create game function. This function takes in what kind of game it will create as a parameter, then it initializes a blank state of the game that has chosen to be played. This function is called regardless of if the game is played in single player against the computer or user versus user multiplayer mode. From here, the status of each piece has a value assigned to it and either the graphical interface or the physical board (depending on the stage in development) should be able to display the game. Game squares have attributes such as "is_occupied", "piece_type = knight," "color = black/white," etc to indicate the current status and availability of each square. Each square is also assigned a value according to a matrix, so moves can be calculated mathematically. For example, (0,0) is for the top left square with the format of (row,col). The pieces have an attribute as to what square they are on. Both the piece and square are updated upon each move.

The game checks for a win status immediately after every move. Since only one piece is moved at a time for most games, the checking function only checks pieces affected by the most recent move. The checker checks the status of any affected pieces where the piece moved to, as well as the place where the piece moved from. For player versus player games, the game becomes usable again for the next player to make their move after the check is complete and give the user an indication that a move may be made. After the next move from the player, the cycle progresses in the same way until a win, stalemate, or end game condition is found. For player versus machine games, the algorithm to determine the next move is run and executed upon completion of the game win check function. The check win function is run after player input, before the AI picks its move. Likewise, it is run after the AI makes its move, before the player is told to make their move. The game win function should also check to make sure the move is legal and does not result in a gridlock event. In some games, gridlock events are legal to make, but result in a stalemate. The game win function then ends the game and sends a signal to send a "stalemate" call to the interface or board.

The storage of squares is in a two-dimensional array. This is consistent with the conventions of calling squares by their column and row. Squares in the array are objects, and declared to have all their individual components assigned initially to zero (or None), but are assigned their true initial value depending on the create game function.

Game squares have attributes related to all games declared, but only those in relation to the game currently being played modifies on call of the create game function. Storage of pieces is within one of multiple different one-dimensional lists or within individually declared variables, depending on what game is being played. Like the squares, pieces are automatically declared zero/None or default values that are modified according to the create game function call. Game pieces are treated as objects with attributes and modifiers.

IV. Artificial Intelligence

One key feature of the GameFrame is that one user can play a complex game against the computer. We utilize an algorithm based on backtracking in order to have the computer solve for its optimal move. Once the machine is set to one-player mode, this causes the mode of operation to automatically calculate and execute the next move of a computer player before allowing the user to place their next move. Different difficulties are needed for the computer's move-making algorithm, as not everyone has fun playing at the same difficulty levels. Different games may also benefit more from different methods of solving algorithms, so multiple styles may be used to solve games. Artificial intelligence and backtracking are the two main types of game solving algorithms that are used, but others are available as well.

In an artificial intelligence design, we have the machine run many different instances of the game against itself and reward it based on how well it does. To create different levels of difficulty, we add randomness into the algorithm itself, or even reward it differently depending on the different difficulty levels we hope to achieve. Three discrete difficulty levels are created for the user to play against and each of them correspond to the different difficulty levels of easy, medium, and hard. Though our hardware supports the python coding language as well as many different libraries and frameworks, we kept most of it proprietary both to have control over it and learn more about AI implementation.

In our backtracking algorithm, numerous different varieties of solving are implemented in order to determine the difficulty level. In the higher difficulty levels of the backtracking algorithm, almost every different path that can be taken is explored, then the best one is taken. In order to limit the difficulty of the computer, less paths might be explored, giving the human player more opportunity to “trick” the machine into making a bad move. This also comes with the advantage of resembling more closely how a human player plays chess - analyzing numerous different courses a game may follow and picking the one with the most opportunities of success. By utilizing backtracking, more possible difficulty levels can be chosen from than if using an AI implementing Deep-Q learning, or convolutional neural networks, as a variable can be declared for how much backtracking the algorithm does in the decision making process. Thus, backtracking allows for a slider type of scale instead of a few discrete levels.

V. Front End

The important objective that was carried through the process of developing the front-end GUI was functionality. The front-end takes in the button inputs pressed from the player and correctly directs it to the game engine. This then results in the game engine to correctly interpret this message and respond with the appropriate response to be displayed to the user. The front-end menus we created provide a simplistic design for the user to be able to safely navigate through to arrive at their desired destination. :The front-end GUI was the bridge between the user and the game engine to ensure proper moves are carried out throughout the game. The majority of the front-end aspects are created thanks to one of python’s libraries, PyGame.

This library is the backbone of much of the functionality carried out with each command. PyGame helps in creating many of the shapes and buttons displayed to the user while adding plenty of customization. We implemented functions and classes in order to be able to create reproducibility in our code. For example, each game we made has a game menu attached to it. Initially we made game menus for each individual one with their own functions. However with careful examination, we became aware that much of the elements are shared between each game menu. We wrote these functions again to create a cleaner coding environment and ended up with an easier environment where buttons or text that need adjusting are simpler to do so.

Ensuring that the front-end is properly communicating with the back-end is something we heavily tested. We did a lot of human testing on the front-end to make sure that every intended use is correctly shown on the screen. Buttons that the user needs to see are shown when they need to be. For example, when a player selects play they must be presented with the option to play against another person or the ai. This was done with Flags being thrown in the code for certain situations.But the most important element is being able to correctly read the user’s input commands and carry them to the game engine for process.

An example of the in game GUI that is shown to the players is shown below.

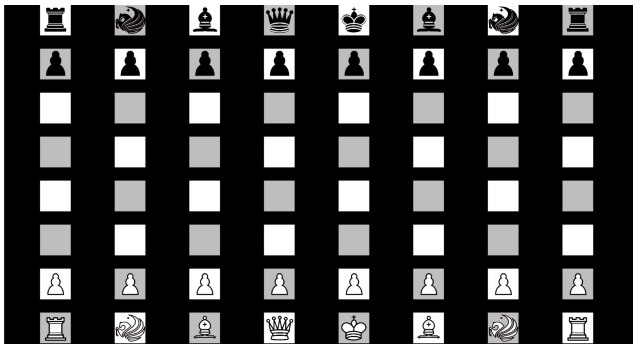


Fig. 2. GUI interface under the buttons.

VI. Overview of ICs

The ICs used include three voltage regulators, a shift register, and an MSP430 on the main PCB. Additionally, there are two breakout boards with RS232 connectors.

For the regulators, all three regulators are the LM2576 since we do not need any boost converters as originally anticipated. For two of the regulators they supply 5V, but we separated them to isolate noise and to ensure each sufficiently supplies non-competing current to the proper device. These 5V regulators power our LCD screen and our Jetson Nano. The 3.3V regulator powers the MSP430.

The shift register is an SN74HC595 8-bit shift register that we use to strobe the columns one at a time in our button decoding process. Doing this allows us to decipher the contact presses while minimizing the amount of wires needed to the embedded processor.

The RS232 breakout boards use MAX3232 ICs and a standard RS232 port. One connects to the MSP430, while the other connects to the Jetson Nano.

VII. Voltage Regulation/Battery

The current sinking devices needing power are an LCD panel that needs 5V, a Jetson Nano needing 5V and the MSP430 which needs 3.3V. The approximate power draw is as follows:

TABLE I. Power Draw of each component

Device:	Voltage	Current	Power
MSP430	3.3 V	.350 mA	1.155 mW
Jetson Nano	5 V	1-2 A	10 W
LCD	5 V	1-3 A	5 W
Battery	12 V	3 A	36 W

More amperage can be afforded after step-down conversion and the actual power that our devices draw is less than the expected. The battery has a listed Wh life of 66.6. Assuming maximum current draw from our devices this gives us roughly 2.5 hours of use. In actual use while developing on the Nano or running our game engine, we saw a typical lifespan of 3-4.5 hours of life before battery depletion. At least, to a threshold where the voltage was too low. This is thought to be caused by the devices not always running at max load. The Jetson Nano and display showed only 1 A drawn via a current clamp at multiple different times.

The amount of voltage output from the LM2576 is decided by tying the output voltage to the feedback with positive feedback oriented resistors. There is an additional factor of multiplying by 1.23 V. That is to say the voltage was determined by 1.23 * 1+(R2/R1). With this in mind, the regulators were set to the needed voltages. The Nano was highly sensitive to the input voltage it was receiving, so the resistors were changed to be even more precise than the others, while we also supplied a tad higher voltage to account for the voltage drop on the way to the device.

VIII. Heatsinks

The power dissipation across the 3.3V regulator is extremely low and there are no thermal issues with operating as is. Both of the 5V regulators have a little lower voltage drop across the regulator, but a much higher current draw. Since the LM2576 has a thermal-ambient resistance of about 41℃/W, a power efficiency of about 75%, and a maximum operating temperature of 125 ℃, we needed a heat sink to help with heat dissipation.

Voltage drop across regulator:

$$(1 - 0.75) * 12 = 3\,V \quad (1)$$

Power dissipation across regulator (LCD):

$$3\,V * 3\,A = 9\,W \quad (2)$$

Power dissipation across regulator (Nano):

$$3\,V\ *\ 2\,A\ =\ 6\,W \tag{3}$$

Temperature increase (LCD):

$$9\,W\ *\ 41\,℃/W\ =\ 369\,℃ \tag{4}$$

Temperature increase (Nano):

$$(Nano):\ 6\,W\ *\ 41℃/W\ =\ 246\ ℃ \tag{5}$$

The heatsinks used to counteract these large temperature increases dissipate around 16.7 ℃/W. Additionally, in order to ensure a more preferred airflow for the heatsinks a small fan was mounted to pull ambient air away.

IX. Embedded Code

With the MSP430 having a main function of decoding our buttons, most of our embedded code deals with this task. The controller outputs a serial and clk to the shift register to strobe the columns. It then has pin setup as inputs to see if the rows have been pressed.

The other task dealt with by the embedded code is to relay information between the controllers. The detail between transmission can be found under the following *Controller Integration* section.

X. Controller Integration

In order to connect the MSP430 to the Jetson Nano serial communication was used. Both devices had UART ports that could be setup for serial communication. The Nano operates on 3.3 V UART communication, while the MSP430 operates with 3.3 V or 5 V. Though both controllers can communicate directly on 3.3 V, we decided to continue using the breakout boards with MAX3232 on them. These ICs and ports added extra error resilience and component protection through voltage regulation and the use of the RS232 standard.

Communication errors still exist every so often on the serial lines, but they are few and far between enough that they could be handled through software. To do this the software only registers a button press once a certain threshold of press transmissions are received. Any transmission that is outside of our range of 0 to 63 is entirely thrown out by the Jetson Nano.

The range decided for transmission was the aforementioned 0 to 63 which fits into 7 out of the 8 bits allotted for a single transmission across UART. Other considerations for reducing error were to use the leading bit as a transmit flag bit or to send multiple transmissions for every one registered press. We decided against these because the occasional communication errors have no consequences on our current implementation with the current error handling.

XI. PCB

The goal of our main PCB is to deal with decoding the physical buttons and relaying that information offboard to the Jetson Nano. It also is what manages and distributes the power from our battery. Most of the details of the ICs and functionality are covered in other sections in this document. The PCB was designed in Eagle.

Of note, our PCB was chemically etched at home (with proper waste management afterwards of course). The process let us turn out a PCB prototype in the course of an afternoon after the design was completed. As a problem during the process we did not realize that the image had to be mirrored both vertically *and* horizontally. Luckily we were able to salvage the prototype by soldering the ICs to the bottom. After this, everything worked. Even though our intention was just to prototype, we decided to leave it that way because it worked. Improvements to the vertical profile of the device is able to be made if the PCB were populated from a house with surface mount type components.

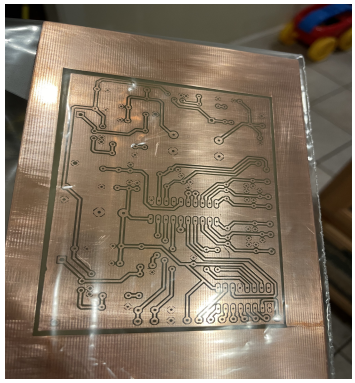


Fig. 3. Fresh PCB after it finished etching

XII. Screen

One of the most influential and challenging components we had to work with was the screen. Originally, we wanted a perfectly square screen since most, if not all, of the games we wanted to implement were square in design. However, as we browsed for our screen, it became clear that finding a screen that fit our budget and the size that we wanted was impossible. We decided to go with a screen that was 14in. by 8.5in.

XIII. Buttons

At first, we still wanted to do a square design by only displaying an 8.5in by 8.5in board on the screen and not use the rest of the screen. But when we were designing the buttons, we realized that the buttons would be too small to see the pieces displayed once we added the conductive actuators. So we decided to expand the left and right sides of each button to add space for the actuators since we had plenty of horizontal space but lacked vertical space.

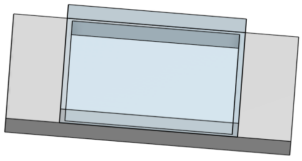


Fig. 4. Shape of new button design to accommodate the actuators without reducing visibility.

When creating the buttons, we used a CNC router to cut out two custom pieces for each button from PETG: the 1.6 in. by 0.94 in. base of the button and the 1 in. by 0.94 in. tile that goes on top of the rectangular base. The vertical height needed to be reduced a tiny bit more from 1 in. by 1 in. due to the vertical screen limitations. When assembling, epoxy glue was used to adhere the square tile to the rectangular tile. A clamp locate system was used to ensure proper alignment as the epoxy was applied and during its initial curing phase (about 15 min). Finally, the four conductive actuators were inserted into the four corners of the rectangular base which were custom cut for fitting them as they were milled.

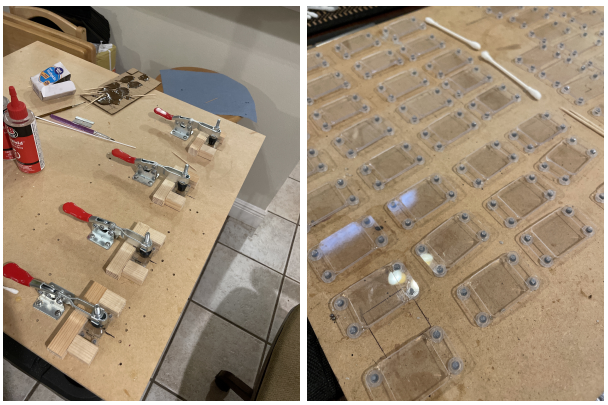


Fig. 5. Depiction of the alignment system gluing four at a time (left) and each of the finished buttons drying (right)

The main benefits to this new shape and design are that we still have a square viewing area for the pieces of our games, and we directly utilize the horizontal screen space. Some other benefits to this new design was that the distribution of force is better, and the buttons were easier to press. Of course the drawbacks are now that all our games will need to be resized and expanded horizontally. This makes all of the games look unconventional in shape.

XIV. Housing

There are two main parts of the housing: the base box and the lid. It was designed to hold the PCB, the Jetson Nano, power supply, the screen, and the buttons. Inside the box, there are shelves that were designed to have the screen sit flush to the top of the box with all the other components underneath the screen. There is also a hole cut out for access to the power switch and charging port.

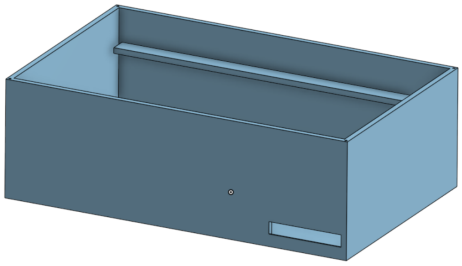


Fig. 6. Model of the box

The lid was designed to enclose the whole housing and also to make sure the buttons stay in place on top of the panel. Due to the fact that the buttons have almost no vertical gaps and essentially touch each other, there were no horizontal divides for the buttons. The vertical gaps were molded to fit within the lower parts of the button so only the square top of the button is shown and interacted with.

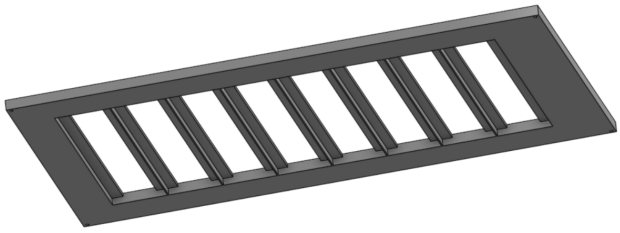


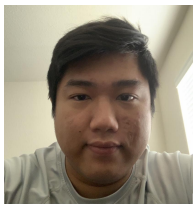
Fig. 7. Model and view of under the lid

XV. Conclusion

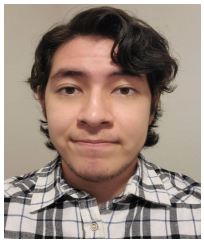
GameFrame was designed to provide a unique experience while also being simple to understand. While not the most conventional way to play board games, it is definitely something that will catch some attention.

Throughout the two semesters working on this project, we ran into a bunch of challenges, but we overcame them through communication and collaboration. This project has provided us with invaluable experience which will help us when we have to work with new and different projects within our field. There is clearly a lot more room to learn and improve.

Acknowledgement



Allen Chion is a computer engineering student at the University of Central Florida. He specializes in embedded systems hardware and programming.
Allen is a member of the development team who is the primary embedded programmer. He is responsible for the main bridge between the hardware and the software, as well as integration testing between the two main components. He is also responsible for designing the housing for the project.



Israel Soria is a computer engineering student at the University of Central Florida. He specializes in programming frontend applications, machine learning, and web development.
Israel is a member of the development team who is primarily responsible for the frontend development and user interface and user experience of the device. Additionally, he helps with the backend development and testing, as well as prototyping and creating automated tests.

Frank Weeks is an electrical engineering and computer science student at the University of Central Florida. He specializes in the hardware for this project, but also has extensive experience with high level software development due to his computer science background.
Frank is a member of the development team who is primarily responsible for the hardware and electrical engineering aspects of the projects. Much of the choice in batteries, voltage regulators, displays, switches, and input/output devices are chosen or designed by Frank. He is largely the designer for the logistics of the hardware and physical challenges of an 8x8 grid with both input buttons and output displays. Additionally, he is largely responsible for the hardware system testing and the development of the PCB and board layout. He had a large part in the manufacturing of the PCB and Switches and added to the embedded software.

Levi Masters is a computer engineering student at the University of Central Florida. He specializes in logic systems design, artificial intelligence, programming backend applications, and software architecture. He is a member of the development team who is primarily responsible for the backend application that the GameFrame runs games on. The logic system of the game as well as the AI that plays against the user in single player mode are also delegated to him.

References

[1] Team Member. (2020, October 28). How to Castle in Chess? Chess.com. https://www.chess.com/article/view/how-to-castle-in-chess.

[2] How the Chess Pieces Move: The Definitive Guide to Learning Chess Fast. iChess.net. (2021, July 14). https://www.ichess.net/blog/chess-pieces-moves/.

[3] Hasbro. (2003). Checkers Instructions. f.g.bradley's. https://www.fgbradleys.com/rules/Checkers.pdf.

[3] Mott, V. (n.d.). Introduction to Chemistry. Lumen. https://courses.lumenlearning.com/introchem/chapter/other-rechargeable-batteries/.

[4] 3D Printer Filament – The Ultimate Guide. All3DP. (2021, July 19). https://all3dp.com/1/3d-printer-filament-types-3d-printing-3d-filament/.

[5] Language Solutions. NVIDIA Developer. (2021, April 21). https://developer.nvidia.com/language-solutions.

[6] What is Switch Bouncing and How to prevent it using Debounce Circuit. Circuit Digest. (2019, June 7). https://circuitdigest.com/electronic-circuits/what-is-switch-bouncing-and-how-to-prevent-it-using-debounce-circuit.

[7] Arduino Playground - SoftwareDebounce. (n.d.). https://playground.arduino.cc/Learning/SoftwareDebounce/.

[8] Knight, D. (2021, May 19). Introduction to Linear Voltage Regulators. DigiKey Electronics - Electronic Components Distributor. https://www.digikey.com/en/maker/blogs/introduction-to-linear-voltage-regulators.

[9] Person. (n.d.). Which ADC Architecture Is Right for Your Application? Which ADC Architecture Is Right for Your Application? | Analog Devices. https://www.analog.com/en/analog-dialogue/articles/the-right-adc-architecture.html.

[10] Air vs Liquid: Advancements in Thermal Management. Boyd Corporation. (n.d.). https://www.boydcorp.com/resources/resource-center/technical-papers/air-vs-liquid-cooling-advancements-in-thermal-management.html.

[11] Everything You Need to Know About Heat Pipes. Advance Cooling Technologies. (n.d.). https://www.1-act.com/innovations/heat-pipes/#benefits.

[12] Encyclopædia Britannica, inc. (n.d.). Checkers. Encyclopædia Britannica. https://www.britannica.com/topic/checkers.

[13] Encyclopædia Britannica, inc. (n.d.). Pong. Encyclopædia Britannica. https://www.britannica.com/topic/Pong.

[14] LCD-Keys. www.lcd-keys.com. (n.d.). https://www.lcd-keys.com/.

[15] Buttons – lcd pushbuttons. SUZOHAPP OEM. (n.d.). https://oem.suzohapp.com/products/buttons-decks-and-dps/buttons/lcd-pushbuttons/.

[16] Dc-96f. DC-96F Heavy Duty Plastic Electronics Enclosures | DC Series. (n.d.). https://www.polycase.com/dc-96f.