GameFrame

Group 16:
Allen Chion - Computer Engineering
Levi Masters - Computer Engineering
Israel Soria - Computer Engineering
Frank Weeks - Electrical Engineering and CS

Contents

1. Executive Summary

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 plain 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 more from their mobile gaming experience.

A game that inspired some of the design of the GameFrame is a rhythm game called “jubeat.” This game plays music to the gamer, then the gamer taps tiles in time with the music. This creates an experience with immersive and sensational appeal. When the game was transformed into a mobile version however, 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 mobile oriented gaming device, the GameFrame needs to be lightweight, affordable, and have adequate battery life. The GameFrame should also have sufficient computational power to run and play games using its gaming engines and game solvers, of which we might refer to as the AI. As well, it should be robust enough that it can be dropped, thrown into a backpack, and withstand how small children tend to handle things. The primary factor that will likely impact the GameFrame’s ability to accomplish these things will be 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 will spend more overall, as components will be bought individually and manufacturing cannot take place at large scale. That is, we will be 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 will be 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.

2. Customers, Sponsors and Contributors

2.1 Potential Customers

2.1.1 UCF Engineering Department

The department of electrical and computer engineering at the University of Central Florida, as directed by the organization ABET are seen as customers that we must satisfy for our project.

The department of engineering requires students to partake in a design challenge. This challenge should incorporate all the aspects of the engineering degree of the students involved. For our particular group, this includes aspects of both computer and electrical engineering. This means circuit design, logic and application programming, and embedded programming should all be incorporated into the project.

This project satisfies the department's demands. It is a design project complete with a physical board layout on a PCB, a microcontroller with embedded programming, as well as software applications. Each member of the group is involved in numerous aspects of the project, but also primarily works within the specialized area of expertise they hope to gain skills for.

2.1.2 UCF STEM Day

STEM day is a free event put on by the University of Central Florida in order to showcase science, technology, engineering, and mathematics programs and activities at the university to Kindergarten through twelfth grade students.

UCF annually hosts the event with different projects and activities the university has in relation to the STEM fields. This event generally needs lots of *interactive* STEM related design projects and presentations. The university prefers having those projects and presentations created by UCF students. In the past, robotic games have been used to display class projects focused on design. They show the desire for interactive game-like projects continues to be high.

The GameFrame is designed to be an interactive gaming board that can accommodate one or two people. This means that students participating in the STEM day can play against other students, or against the computer. They are able to learn about the kind of things possible to achieve at the University of Central Florida in these programs. The GameFrame also serves as a physical representation of programming and software, which is often difficult to showcase in person to younger audiences.

2.1.3 Parents

Another audience seen as potential customers are parents who have children in the age range that requires constant supervision. Additionally, these children are often developing their critical thinking and logic skills.

The need of these parents often involves their children sitting still and quiet for a period of time while they cook dinner, do laundry, or simply want to recover with some peace and quiet. The easy solution can often be to hand them an internet enabled smart device. However, this presents the problem of moderating what the child is able to access. This winds up being just as much supervisory work. On top of this, lots of seemingly good content on the internet can lack interactivity and lead to children losing out on valuable problem solving and critical thinking development. A device that can be stand-alone and interactive would be a much better solution.

The GameFrame happens to be a stand-alone and very interactive gaming device. No worry needs to be given as to what users or children may be exposed to, as the content is complete and understandable at the time of use. No unexpected content is streamed to the device. With haptic feedback and multiple thought-provoking games to choose from, young minds can be engaged, and problem solving skills can be exercised. The GameFrame provides hours of entertainment for parents to provide to their children while they catch up on the day.

2.1.4 Mobile Gamers

People on the go may want something to do to keep themselves busy or have some entertainment. These people, when traveling, often find themselves with lots of downtime, but little to do during it. Many forms of entertainment either require a stationary area of operation, cannot be bumped or moved, or require a stable internet connection which is not always available to travelers.

The GameFrame provides an excellent solution for the people on the go. The gaming platform does not require an internet connection, has no moving pieces, and is completely self powered and freestanding. Additionally, this device is much more engaging and interactive than smart-phone gaming, and will not eat into the battery life of your cell phone over time.

2.1.5 Waiting Lounges

Offices where people would generally be provided with magazines, toy chests, or televisions while people wait for their appointment could benefit from the GameFrame.

Waiting for an appointment can be one of the most mundane and time-wasting experiences a person can go through. Little work can be done, no real insight as to the timeframe of the wait is known, and entertainment is generally limited to outdated magazines or house renovation TV shows. A solution that both engages and entertains someone would be a perfect addition to any waiting room.

The GameFrame provides an all-inclusive ticket to entertainment for bored customers. With games most people know how to play in a novel package most users have not grown tired of, patients in waiting lounges can have something to do other than staring at their phones or reading old magazines. The GameFrame has little barrier to entry and can accommodate users of most ages. It aims to provide a pleasant and interesting wait period for one's next appointment.

2.2 Contributors

2.2.1 Members of Group 16

Allen Chion
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.

Levi Masters
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.

Levi 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 will play against the user in single player mode are also delegated to him. Levi will largely be responsible for testing and partially integrating the engine for usage on the hardware itself.

Israel Soria
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
Frank Weeks is an electrical engineering 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.

2.2.2 Consultants and Advisors

Jon “Box” Klages
Jon Klages is a contact with knowledge and experience of arcade machines, including jubeat - an arcade-style game that inspired some of the GameFrame’s design. He has extensive knowledge of the inner workings of arcade-cabinet-style games. Jon serves as a consultant for the development team for the groundwork design ideas and understanding of what components in general will help get the job done.

Arthur R. Weeks Jr.
Professor Arthur R. Weeks Jr. teaches at the University of Central Florida in the department of Electrical Engineering. Dr. Weeks provided consulting information for us on how to implement various aspects of embedded and hardware design.

3. Project Narrative

3.1 Project Motivation

Our group consists of three computer engineers, and one electrical engineer and computer science dual major. So, we were looking for a software intensive project which still involved electronics. While engineering is often used to solve practical problems, we wanted to create something less practical and instead something focused on fun. After brainstorming and working through a few ideas, we arrived at a gaming robot that plays games like chess. This gaming robot would allow for enough design in the electrical engineering space that it makes sense for a senior design project rather than for a software development class. As well, there are plenty of opportunities for software development and there is plenty of potential AI to give our three computer engineers the space to code. Being essentially an 8x8 grid, we also have options and ideas to code in a few different games aside from chess. Such games could include checkers, connect four, tic-tac-toe, or even a rhythm game like jubeat. Being this open ended allows us flexibility in the complexity of the project. We may run ahead or behind schedule and we can simply choose to add more or less features if we are near the end and should we want to implement more games. This project could also be something fun to show off at events like STEM day, which UCF often hosts for K-12 students.

Our very first ideas included a trading card sorter organizer, a poker playing robot, a smart blind system, and an eco-friendly liquid product dispenser (like soap, detergent, cleaning chemicals, etc.) for in-store distribution. After discussing our goals for this project, we leaned into the poker playing robot as it had the most amount of software complexity. However, various issues came to light with this idea. The multitude of mechanical components involved in handling cards and poker chips would be beyond the scope of our fields. One of the biggest issues was the dexterity of the mechanical arm. After some more consideration, we figured it would be better to change the game being played entirely, rather than trying to solve the problem of playing a card game through different means. Eventually, the gambling robot evolved into a gaming robot.

Some of the motivation for our extra considerations, while still balancing key things like budget, is to build a project and skills worthy of our resumes. Experience with certain devices and languages, interfacing hardware components while following standards, and integrating AI into our game engine are all achievements and skills we can display at the end of a project like this.We all look forward to working on this project, as the end product will be something fun to use, and each of us have something to work on in an area we like.

3.2 Project Description

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 will provide a more physical way of interacting with a game, which is something usually restricted to arcades. This device will be 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 will have the versatility for additional games to enhance the user experience. The additional games will only need to be implemented via software configuration.

3.3 Project Challenges

For this project, each of the group members will take on some new roles we have never performed before. Each time we hope to implement a new feature we will have to educate ourselves and do research first before we can start making or tweaking it. This will make development much slower than a typical project where we would be more likely working with prior knowledge and experience that just compounds with some learning and research. We all have personal lives, school, and jobs to be involved with as well, and do not have an uninterrupted eight hours a day to devote solely to this project as we would with one in a career. Unique schedules that do not always line up also creates a need for some kind of organizational scheme. The scheme has to allow us to keep working on a schedule that works for each of us with the short time that aligns being allotted for important meetings and briefings.

This project began towards the end, or arguably the middle, of a pandemic. Working in such a way creates a list of additional challenges to overcome. Scarcity of resources has hit most sectors of global trade, and this will likely last the duration of the development of this project. Although distribution of all sorts is of concern, a very relevant specificity to mention would be the known semiconductor shortage. Higher prices are something we will have to cope with, as global infrastructure has taken a massive hit in the past year and a half. Also on top of this, demand for consumer electronics has skyrocketed, which worsens the strain on having electronic parts available. In the beginning phases of this project, we are confined to remote contact only, however this is the challenge that is most likely to be resolved as the world slowly reopens and more people get the vaccine. Most of the discussion of the future and implementation of this project have had to take place over VoIP, leading to a lack of interpersonal communication.

3.4 Market Analysis of Competitive Products

Competitive products to ours include a palm-sized digital chess device which uses an unsophisticated LCD and typically requires using a stylus. Alternatively, there are portable chess sets which use a variety of physical chess pieces. A consideration for comparison is also a device like an iPad or tablet PC as these can allow versatility to add and play more games with the same device.

The proposed solution alleviates some of the downsides of these competitive products. Using a rhythm game cabinet known as jubeat for inspiration we wish to design a portable GameFrame with 8 by 8 (or 64) tangible buttons. The LCD will be of a better quality and not require the hassle of a stylus. Additionally, with portability in mind, supplementary pieces are undesirable and not required by our device. Even with chess as our focus, the device will have software modularity that will allow other games to be designed for it. Though it will not be able to have the full game library capabilities of a tablet, the tactile nature of the device should prove to be a more enjoyable experience. A sample of what jubeat looks like can be seen in the appendices under "Other Links."

With the aforementioned comparisons in mind there are several goals that exist. The device should be:

- Lightweight - No heavier than the approximate weight of a laptop
- Portable - Easy to use with no external pieces, and sustainable on a battery for period of time
- Reasonably priced - No more than the price of a tablet all together

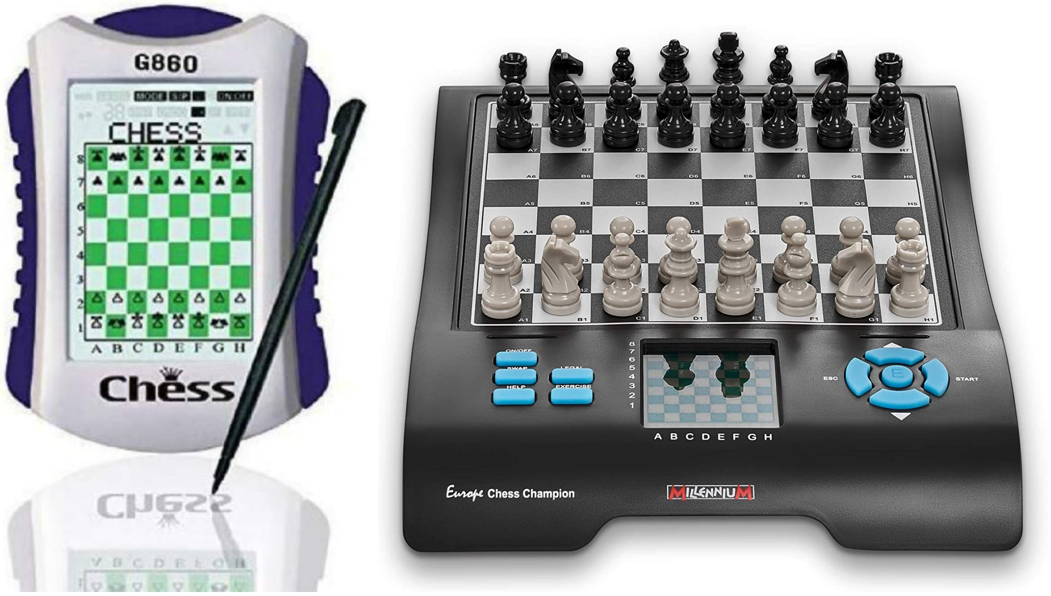


Figure 1: Example pictures of competitive products. On the left is the aforementioned stylus comparison. On the right is the referenced kind with portable chess pieces.

3.5 Alternate Considerations

We were considering three other projects at first. We thought of a smart blind that would integrate a bunch of features. Some of which included features from a previous group. We thought the design was simple and reliable, but we didn't know if this project would be something that we would be proud to create and tell future employers about. So, we decided to keep it as a fail-safe in case we could not think of a better design. Also, one of our team members wanted to gain more experience with artificial intelligence, so we decided this project would not suffice. A different project we considered was a refillable station that would be used by grocery stores for refilling plastic containers to lower the waste of plastic. This project lacked something that really held our interests similar to the smart blind, so we decided to not choose this project as well. It seemed like it would be something that really could not be built upon with additional features.

We then considered a kind of robotic arm that plays poker with another person. We realized this project could be a bit closer to the field of artificial intelligence since we would make the robot learn when it is wise to bet and when to fold. This project also grasped our attention more in comparison to the other two projects, because it seemed more entertaining than the others. However, we realized that it would be very difficult to pull this off since the arm would need to be very dexterous since it would be very demanding for the arm to be able to pick up playing cards. So we tried to stray away from card games to avoid the challenges that arose from them. We then decided on chess using the grid-based buttons featured in jubeat. With this system, we can add more features, such as playing jubeat or other possible games in addition to chess, depending on how successfully we proceed with the project.

4. Requirement Specifications

4.1 Hard requirements

Requirement Type	*Name*	*Description*	*Value*
Power	Battery Life	As it is portable, the design will require a sufficient power supply unit. The device should last a minimum of 2 hours so as to be sufficient for portable play. Hopefully, we will be able to reach closer to 4 hours.	2-4 hours minimum
Physical	Dimensions	The dimensions of the device should not exceed 12-inches in either length or width. The depth is not mandatory to be below a specific required length.	1-foot width or length maximum
	Weight	The weight should not exceed 4 lbs so as to not weigh more than a laptop with similar dimensions	4-lbs maximum

Requirement Type	*Name*	*Description*	*Value*
Monetary	Unit Cost	The device should not exceed that of a $400 tablet. Being priced above the $40-60 of comparable cheap portable board games is justifiable, however, with our intent of a higher quality experience.	$400 per unit maximum
Software	Chess	The device should display the ability to play through a game of standard chess with 1 or 2 players.	Playable chess Both 1 and 2-player functionality

Table 1 (Hard Requirements): This lists our mandatory requirements influenced by customer and group member desires.

4.2 Soft requirements

Requirement Type	*Name*	*Description*	*Value*
Physical	Dimension	A goal of keeping the device within 2-inches deep should help for our goal weight.	2-inches
	Drop Resistance	The machine should be capable of surviving a drop from up to 5 feet, the standard for smart-phones.	5-feet
	Temperature resistance	The device should be able to withstand conditions from 0-100 Fahrenheit - typical weather fluctuation.	0° - 100° Fahrenheit -18° - 38° Celsius
Cost	Unit Cost	A soft requirement is to make it within the cost of more expensive portable chess games which can go upwards of $300.	$300 per unit maximum
Software	Computer Move Time	The player should not have to sit and wait very long for the computer to make a move; 1 second is the maximum time.	1 Second
	Difficulty Levels	AI should have 3 (or more) difficulty levels so players can have more personalized experience.	3 Levels

Table 2 (Soft Requirements): A table of our desired requirements representing stretch goals.

4.3 Constraints

Constraint Type	*Name*	*Description*	*Cause*
Power	Battery	The device must be able to operate off of sustained battery life.	The device is portable
	Power Port	There needs to be a port that allows charging via a USB power source.	The device needs to be rechargeable
Physical	Buttons	There should be 64 physical buttons or partitions.	The device must play chess
Software	Game Library	Games solutions that require a GPU cannot be implemented.	Computational power of hardware
Cost	Budget	The project has a budget limited by the expenditure capacity of our group members.	We are not sponsored
Time	Due date	The time we have to work on our project is limited	Senior design has two semesters

Table 3 (Constraints): A collection of constraints imposed upon us by outside factors or design decisions.

4.4 Realistic Design Constraints

4.4.1 Economic

An economical constraint is that our budget is $400. Aside from that we personally do not wish to spend an exceptional amount for this device. This leaves constraints on our product because if one area of the device costs too much it may affect other parts of the device; it could cause us to either go over our intended budget or to cut corners in other areas of our device. As previously mentioned in our project challenges, the availability of certain parts may also cause us to change something out if it is not available.

4.4.2 Environmental

The impact our device could have is that it could replace the need for people to purchase multiple different kinds of board games and instead have them all in one device. This could reduce the need to create board games and save those resources, since we produce the same game but digitally. An effect the environment has on our device is also our power usage. We can aim to reduce our environmental footprint by aiming to use recyclable materials. As stated in the "Related Standards" section, there is a required criteria and satisfying optional ones that can lead to us having bronze, silver, or gold level conformance.

4.4.3 Social

Our product's primary social constraint to consider is the fact that it will only support at most two players. There are not many gridded board games in existence that can support more than two players. However, given the nature of the device, it is still possible for more than two people to use the product through taking turns playing or having non players watch the two players instead.

Another social constraint to consider is how the bystanders are affected by the device. For example, the game should not be very loud to potentially disturb people in the surrounding area. Proper volume control should be implemented so the user can adjust according to their environment.

4.4.4 Political

For our political constraints, we dived into the various laws and regulations that dictate consumer electronics. There are many government agencies that oversee these laws such as the Federal Trade Commission and the Department of Energy.

To ensure that a product is safe for children, there are a few laws in place that require strict compliance and testing to allow a product to be marked as safe for children. Otherwise, the product must contain a label that states "keep out of the reach of children". While a lot of these laws, such as the Consumer Product Safety Improvement Act that states electronics intended for children must contain less than 100 parts per million of lead content, are important restrictions, many of them will not have a direct constraint on our ability to build this project. Most of this is due to the fact that we are not manufacturing these parts. We should be able to safely assume that any component of the build we intend to use should already comply with these safety laws since they are available to purchase from a reputable source. However, we should still keep in mind to thoroughly research a material or part we intend to use to make sure they are already in compliance. If a part seems like it is not safe or sketchy, we should not consider it even if it means we have to spend more money on a more reputable product.

A law that will most definitely cause a constraint in our project if we want it to be safe for children is the Federal Hazardous Substances Act. In the law, there is a section that states the product must not contain any sharp point or edges. This must also hold true where normal use and potential damage, or abuse, must also not expose any sharp point or edges. While the law states that this must be evaluated by a commission, it is not likely we will be able to actually have it evaluated by a government agency. However, this law is still an extremely important consideration as we design the enclosure of our project. This constraint will cause some important design touches where the corners and edges are rounded off and we have to make sure whatever material we use is sturdy enough to not break easily through normal use.

Another important constraint, while not directly a law, is the Transportation Security Administration (TSA) policy that prevents a lithium ion battery of 100 watt hours or more into a plane. Obviously we would like for our product to hold a really big charge and have extended use capabilities, but we also want the product to be easily portable. So battery capacity should be limited to allow our potential users to easily bring this into an airplane or wherever they go.

While this constraint will not affect the physical design of the product, copyright laws will determine what games we will consider to install on it. To avoid any potential copyright infringement, the main games we will consider will be part of the public domain such as chess and checkers. There are obviously many more choices than those, but to make things easier, we should check if the game we want to implement is in public domain first.

4.4.5 Ethical

From an original idea perspective, we want to make sure that our project is our own unique design and not something copied from someone else. While we do not have to worry about creating our own games since we plan on implementing games in the public domain, it is important to do our own implementation using our controls.

While mentioned in laws that are aimed towards the safety of children, we should seek to ensure the product is safe to use for everyone. Ethically speaking, we should not cut corners in a way that may harm the user. This may ultimately cause our cost to go up to avoid compromises.

4.4.6 Cultural

Currently, the only demographic that this product is targeted towards are English speakers. Due to the cultural limitations of our project members, there are not a lot of languages we can support in Game Frame. Ideally, we would like to support many languages so that our product has the potential to reach a more international audience, but given our own cultural constraints, it is likely that English will be its only supported language as it is the language all of us share in common.

Another limitation we have is that it is likely not worth implementing games our main demographic do not understand. Games from other cultures like Shogi and Mahjong are most likely not popular enough for people to immediately understand. Of course like previously mentioned, it is not out of the picture to consider if we were seeking a more international audience. However, for us to specifically target a different demographic, a completely separate version of the Game Frame might have to be made that includes games from that region.

4.4.7 Health and Safety

A health restriction that can be considered as a constraint for our device includes ensuring the device does not heat up too much in case someone decides to play the Game Frame on their lap.

Making sure that the device is not too heavy is also good to avoid carrying and moving excess weight. This means that we must make sure the Game Frame is not too heavy, so we must be mindful of the components we use. Cables can not be exposed for the safety of the user and must all be contained within the product.

4.4.8 Manufacturability

Manufacturability of the product restricts the design that we can implement. We are only able to manufacture at the level of college students since we do not have access to typical manufacturing assembly. We are somewhat limited by the services and tools that are available to us. All of the manufacturability of hobbyists, as well as some of the perks of being students, such as educational licenses and lab access, are the main things we get to work with.

4.4.9 Sustainability

Issues will arise from sustainability. Being aware of the components that will be used the most - the panels - will help us in identifying sustainability concerns. Parts of the device that will be used the most must be resilient in order to ensure continuous usage. We need to make sure that the constant application of force on the panels, for example, will not damage them or any other part they interact with. Since the issue is not completely avoidable we can at least make it so that the panels have good durability and feasible repairability. These are both constraints sustainability causes.

Another design factor that puts a strain on sustainability is the portable nature of the device. Since the GameFrame is portable it must have the ability to endure certain drops and being carried around. The constraint here dictates the viable materials we can use.

4.5 Standards

Standards are created to provide a cohesive way of designing, testing, and evaluating engineering products. There are many organizations that create standards for a variety of fields for different engineers to follow. Some of these include the American National Standards Institute (ANSI), International Organization for Standardization, and the Institute of Electrical and Electronics Engineers (IEEE). For our project, we will acknowledge certain standards created by IEEE that will affect the overall design of the product. Following these standards will ensure that the product can be easily evaluated and be safe for consumers. While these standards will be acknowledged, most, if not all, of these standards will be used as more of a general guideline during our design process for our project rather than a strict compliance, as there are many limiting factors to our actual ability to comply with them. Some of these requirements include certain manufacturing and corporate practices that we cannot control or which cannot be applied at all to the scope of this project. However, if applicable in the future, these requirements could easily be transitioned to actual compliance. On the software end, we have created our own personal standards for how we intend to write our code. This will help streamline our development process and make working together a lot easier.

4.5.1 Software

Standard Type	*Standard Choice*	*Explanation*
Square Assignment	(row,col)	Call any particular square by its row and column, starting at the top left with (0,0).
Function Casing	CamelCase	Camel casing is used when calling or creating a function.
Object Casing	snake_case	Snake case is used when calling or creating a function.
Variable Casing	snake_case	Snake case is used when calling or creating a variable.
Global Variable Casing	SNAKE_CASE	Capital snake casing is used when calling or creating global variables.

Table 4 (Software Standards): Personal standards used for software, some of which are based on common practices for the language we plan to use.

4.5.2 Environmental and Social Standards

IEEE Std 1680.1™-2018 Standard for Environmental and Social Responsibility Assessment of Computers and Displays is aimed to reduce the environmental impact and improve social responsibility when it comes to electronic product usage. This standard has two types of performance criteria: required and optional. Required criteria must all be met for the product to follow this standard. Meeting optional criteria can earn a product points to achieve different levels of conformance to this standard. There are three tiers of conformance: Bronze, Silver, and Gold. Meeting all of the required criteria without any of the optional criteria will instantly grant the product Bronze level conformance. To meet Silver level conformance, the product must meet at least 50% of the optional criteria and all the required criteria. To meet Gold level conformance, the product must meet at least 75% of the optional criteria and all the required criteria.

This standard's requirements can be broken down into 10 parts: Substance management, Materials Selection, Design for end-of-life, Product Longevity, Energy Conservation, End-of-life Management, Packaging, Life Cycle Assessment and Product Carbon Footprint, Corporate Environmental Performance, and Corporate Social Responsibility. Each category has their own detailed required criteria and optional criteria to satisfy compliance of that category. Many of the optional criteria are just expansions of the required criteria. For example, a required criteria to satisfy the standard is to use at least a certain amount of recycled plastic, and you can gain extra points towards a higher tier of compliance by satisfying the optional criteria of using even more recycled plastic.

In the substance management and materials selection, some of the required criteria include no mercury in light sources, less than 25g of bromine and chlorine in plastic, at least 25g of plastics are recyclable, and plastic must be separable. Optional criteria that give more product points include even more restricted cadmium, beryllium, and using more recycled material. Some of these required criteria are geared more towards manufacturing, which is something we will not likely be able to comply with since we are buying whatever is available. We can potentially choose parts already made that comply with these standards.

In the design sections, some of the requirements state that certain components such as lithium-ion batteries and whatever needs to be treated must be visibly identified. The plastic parts must also be separable. To ensure product longevity: service support, spare parts, and battery replacement must be services provided by the manufacturer. Some optional criteria include longer rechargeable batteries, ease of repairability, and the ability to remove batteries.

While this is unlikely to apply to this project specifically, there are many optional criteria that involve the corporate level of the product. It is highly encouraged for the product's greenhouse gas emissions, greenhouse gas emissions from transportation, corporate carbon footprint and social responsibility be taken into account. Most of these requirements are optional.

Design Impact of the Environmental and Social Standards

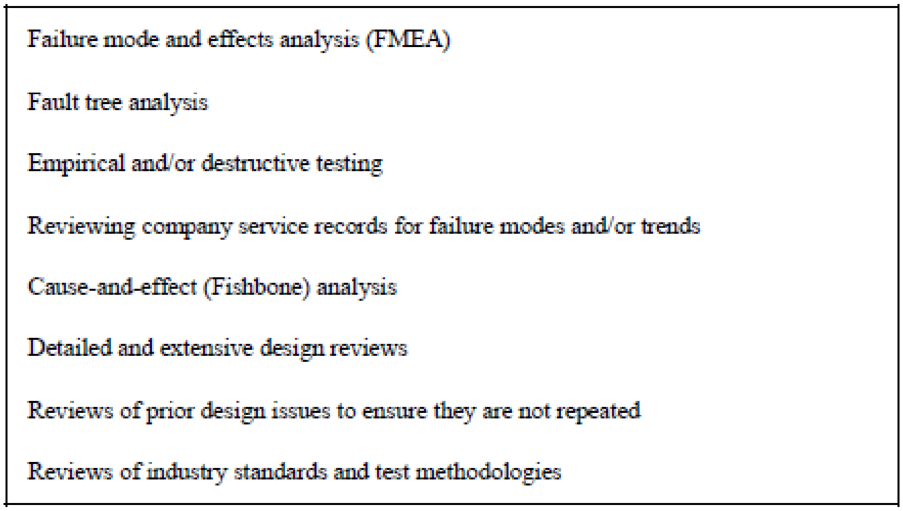
While this standard provides a good guideline to follow, there is a lot in the standard that does not apply. The main reason is that many of these standards apply to the manufacturing process of the entire product. For this project, we will most likely be sourcing all our parts from already manufactured components, and we will only be creating a single completed product for our Senior Design demonstration. Required criteria such as having spare parts available, having service support, and ease of recycling will most likely not be met for the project scope. We also do not have the proper tools to determine the carbon footprint created from our time creating the product. It would definitely be something to consider in an actual, real world product development cycle. As a result of that, complying with this standard already does not work. These things would most definitely be considered if we made plans to sell the product or set up our own manufacturing process.

However, there are some criteria in the standard that would provide some good guidelines with how we would like to approach this project. For example, high energy efficiency would probably be something we strive to achieve. While we will probably not seek out to meet the ENERGY STAR program of efficiency, it is within our best interest to try and make it as energy efficient as we can.

4.5.3 Battery Standards

IEEE Std 1725™-2011 Standard for Rechargeable Batteries for Cellular Telephones is aimed to ensure a reliable user experience and operation of cell phone batteries. While not applicable to this project, the standard also outlines recommendations for the actual design, manufacturing process, and testing procedures of the cells in the lithium-ion battery itself. Even though this project is not about cellphones in particular, our project may consider using a rechargeable lithium-ion battery to power a mobile platform. This standardizes the criteria for verification of quality and reliability of those batteries.

While the standard aims to minimize possibilities of hazardous outcomes, it is not possible to analyze for every potential issue and scenario that may appear in batteries. The design analysis aims to reduce hazards from one or two independent faults during battery's use. To do this, various design analysis tools include, but are not limited to:



Failure mode and effects analysis (FMEA)

Fault tree analysis

Empirical and/or destructive testing

Reviewing company service records for failure modes and/or trends

Cause-and-effect (Fishbone) analysis

Detailed and extensive design reviews

Reviews of prior design issues to ensure they are not repeated

Reviews of industry standards and test methodologies

Figure 2 (Battery Design Guidelines): Design analysis tools aimed at preventing faults in batteries.

To ensure the safety of the user, the standard outlines user interactions and responsibilities that the user should know regarding the use of the battery. This information should be printed on something for the user to see, such as on the device itself or on a user manual:

- Do not disassemble or open, crush, bend or deform, puncture, or shred.
- Do not modify or remanufacture, attempt to insert foreign objects into the battery, immerse or expose to water or other liquids, or expose to fire, explosion, or other hazard.
- Only use the battery for the system for which it was specified.
- Only use the battery with a charging system that has been qualified with the system per this standard. Use of an unqualified battery or charger may present a risk of fire, explosion, leakage, or other hazard.
- Do not short circuit a battery or allow metallic or conductive objects to contact the battery terminals.
- Replace the battery only with another battery that has been qualified with the system per this standard. Use of an unqualified battery may present a risk of fire, explosion, leakage, or other hazard.
- Promptly dispose of used batteries in accordance with local regulations.
- Battery usage by children should be supervised.
- Provide an explanation of security implementation and battery authentication.
- Avoid dropping the device or battery. If the device or battery is dropped, especially on a hard surface, and the user suspects damage, take it to a service center for inspection.
- Improper battery use may result in a fire, explosion, or other hazard.

Design Impact of the Battery Standard

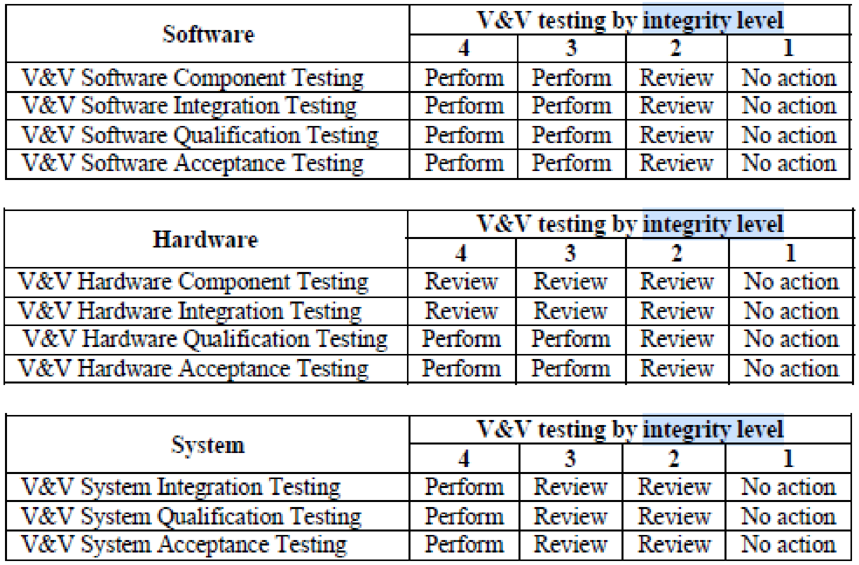
The battery is one of the more important parts of the project, but it will also be something we do not have the most control over. However, what this standard helps us do is know what to look out for when deciding what battery to get for this project and its installation. Preventing physical damage is part of the standard and that is something extremely important to consider when designing our project. We will likely give more consideration to protecting the battery inside the final build to protect it in the event the device is dropped. One important part of the standard we may adhere to is correctly displaying to the user the device's battery life. Another part of the standard that may be of use is making sure that the battery can be easily replaced. With this in mind, we should not integrate the battery into the build in such a way that makes the battery hard to remove in case there is an issue with it. However, things such as detecting faults in the battery or temperature will not be likely due to our own limitations. Just like our political constraints mentioned, it is highly unlikely that a battery being sold on the market is not already extensively tested, but we may still acknowledge the standard in testing and evaluating if the battery is good and safe to use.

4.5.4 System, Software, and Hardware Verification and Validation

IEEE Std 1012™-2012 Standard for System, Software, and Hardware Verification and Validation is a process standard that defines the verification and validation process that is applied to a product's system, software, and hardware development throughout its design. The process will determine if the product meets all the requirements and satisfies its intended use. The purpose of this standard is to establish a common framework for analyzing a product's system, software, and hardware life cycle processes. It defines the tasks for a proper verification and validation process.

Verification and validation are their own separate processes. The verification process aims to check whether the product conforms to requirements, satisfy standards, and satisfy criteria for a complete product life cycle. The validation process aims to check whether the product satisfies system requirements and satisfies the product's original intended use - does it do the thing it was meant to do? The results of this process can facilitate early detection and correction of any problems that may occur and enhance the management of the development process. It assures that the product will meet the performance metric, schedule, and budget that the project initially set out to meet. It also can provide an early assessment of how the product will perform.

The standard is organized into four parts: Common, System, Software, and Hardware. Common tasks are directly related to planning, support, and management of the verification and validation process. These tasks include generating the verification and validation process, system requirements review, and final report generation. Systems, Software, and Hardware tasks all include design evaluation, interface analysis, traceability analysis, component test plans, integration test plans, qualification and acceptance tests, hazard potential, security analysis, and risk analysis of the product. There are also four different levels of integrity for verification and validation testing:



Software	V&V testing by integrity level			
	4	3	2	1
V&V Software Component Testing	Perform	Perform	Review	No action
V&V Software Integration Testing	Perform	Perform	Review	No action
V&V Software Qualification Testing	Perform	Perform	Review	No action
V&V Software Acceptance Testing	Perform	Perform	Review	No action

Hardware	V&V testing by integrity level			
	4	3	2	1
V&V Hardware Component Testing	Review	Review	Review	No action
V&V Hardware Integration Testing	Review	Review	Review	No action
V&V Hardware Qualification Testing	Perform	Perform	Review	No action
V&V Hardware Acceptance Testing	Perform	Perform	Review	No action

System	V&V testing by integrity level			
	4	3	2	1
V&V System Integration Testing	Perform	Review	Review	No action
V&V System Qualification Testing	Perform	Review	Review	No action
V&V System Acceptance Testing	Perform	Review	Review	No action

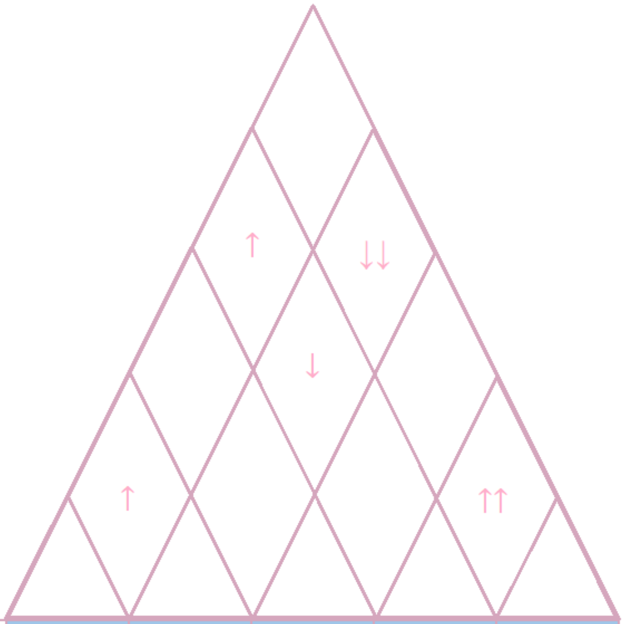
<u>Figure 3 (V&V Testing Tables)</u>: Verification and validation testing with different integrity levels.

The requirement "perform" means that a verification and validation organization specifies and creates the test, confirms that it has been done correctly, and reviews the results. However, it does not require that the organization does the test itself. As long as the verification and validation organization observes the test, it can be done by the project creators. To obtain the highest integrity level, verification and validation testing is mandatory.

Design Impact of the Verification and Validation Standard
Considerations for this standard will shape the way we test every aspect of our project throughout development. Instead of strict compliance, this standard will definitely be used as a guideline for how we verify and validate the goals we set out.

4.6 House of Quality

The best way to analyze the house of quality would be to explain the relationships on it. Furthermore, it would be a good idea to focus on key positive relationships in our design to help meet goals and requirements. Likewise, avoiding negative relationships or at least balancing them as much as we can could be beneficial. Whatever the case for a negative relationship, we need to lean towards the design decision that will reach our specifications. In other negative relationships they have no bearing on a requirement, so we are able to lean towards what we see as most useful or beneficial.



		Unit Cost	Battery Life	Development Time	Dimensions	Weight
		-	+	-	-	-
Low Power*	-		↑↑		↑↑	↑↑
Repair Cost	-	↓				
Development Cost	-	↑↑	↓	↓	↓	
Display Quality	+	↓↓	↓↓	↑		
Lightweight	+		↓		↑	↑↑
Résumé Benefit	+		↑	↓↓		
Target Specs.		<$400	>2 hours	Not specified	<12”x12”x2”	<4-lbs

It should be noted that Low Power is using “-” to denote that less *power consumption is desirable. That is to say, the device would be* more *low power.*

Figure 4 (House of Quality): A QFD of our initial design.

4.6.1 Development Cost and Time

If something takes little time to develop, it likely was something we already knew or was a skill that was not hard. This causes a low development time to have little benefit on our resumes; resume development is a personal goal of ours from the project. Thus, with development time having no impact on requirements and adding nothing towards goals, resume benefit is probably what we’ll lean towards.

The development cost should be noted as being different from cost per unit production. The more we spend on parts for a unit, the more development will cost, but the opposite is not necessarily true. However, on the topic of development time, in a number of cases, spending more development time can save us on cost to develop. Building, crafting, or designing components ourselves can be cheaper. Thus our college student wallets plead we lean towards devoting more time over the alternative of spending more money.

4.6.2 Display

As the quality of the display increases, the complexity of its architecture and of the design to facilitate it does as well. The quality of the display also costs more and is incredibly detrimental to the battery life of the device. Too low on visual fidelity would create a low quality product that does not feel good to use. The display quality should be sufficient enough to not create a negative customer experience while also being as minimal as possible in order to keep the cost low and promote the longevity of the battery.

4.6.3 Size and Weight

Although this is not entirely true, smaller parts tend to cost less. Simply put, this is because smaller parts use less materials. For electronic components, this trend is less applicable, but can sometimes be true depending on the type of component in question. The good thing about this is that we want small parts and we want less cost. Getting components that have small dimensions and less price is a win-win. It should be noted that the density of varying parts causes cost and weight to have less, if any, correlation.

Density is a varying physical property that makes it hard to reliably say that weight and dimensions have a correlation. The density is dependent on what the material for a part is. Even if not reliably true, there is an obvious correlation between size and weight. This correlation is especially true when comparing parts of the same material. It is easier to say that lower dimensions have less weight than to say that something that weighs less has lower dimensions. Thus, the conclusion is that using smaller parts will result in an overall less weight for the most part.

4.6.4 Power

Smaller parts also commonly have lower power requirements. Low power parts also weigh less typically (eg: low power batteries). For obvious reasons as well, using a bunch of low power parts will make the battery life goal much more attainable. There are many benefits for low power parts with no huge drawbacks. Any drawbacks that exist are not those that affect our goal requirements or specifications. That is not to say there cannot be an unforeseen detriment, but for the time being it seems very wise to gravitate towards low power components.

A quick note can also be made about large batteries. The larger a battery given a certain energy density, the more power it can supply. This negatively impacts the weight and the dimensions in favor of power. A balance must be found to meet both of the specifications, since they are deemed as our most important design requirements for the purpose of being considered portable. To briefly mention, lithium-ion batteries can alleviate the troublesome correlation some and provide good energy per weight. This is covered and expanded upon elsewhere in this paper (specifically, under “Research”).

4.6.5 Repair

Repair cost is not an important factor for us, especially considering we would prefer to keep unit cost low since. The unit cost has an actual goal we aim for. It probably is selfish on the ends of developers but repair cost is not on the forefront of our mind for designing this project. On the plus side, repair cost is somewhat alleviated from the fact that design and parts are inherently modular. Replacing malfunctioning parts is cheaper than buying a new product. Unfortunately, any customers who are not tech savvy would find this to be little consolation.

4.6.6 Resume

Resume benefit is a very influential goal towards some of our decision making. Being able to reach our soft requirement goal of a four hour battery life could potentially prove to be a challenge. Deciding on the right battery and developing the right power managing circuitry might develop problem solving or experience that would be good for a resume.

5. Design

5.1 Overview

The “GameFrame” is an all-encased board consisting of an 8x8 grid of buttons made of plexiglass with a digital display under them to show the status of each tile. It will also consist of an NVIDIA Jetson Nano development board as the brains of the software with an MSP430 to handle the hardware.

The current design plans to use no more than four switches for each button, but this might lessen should power or cost requirements necesitate it. This comes to a total of 256 switches planned. The power regulation of a portable device has to be managed well in order to maintain a proper battery life. The hardware will keep the power regulation in the forefront of the design. The design will also implement a separate cheap controller in order to handle the hardware and to help allow concurrent work on the project. The Jetson Nano, the intended software controller, could perhaps suffice alone, especially to handle the display hardware. Auxiliary buttons for power or other features are considered for the side of the machine, but aside from the power switch they are not fully rigid design choices yet.

As for software, it will be modular in nature to support easier development of other games for the device. An overview of the subsystems to be implemented in software can be found under the “Software” part of this “Design” section.

5.2 Technology

The software for this project will run 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 will utilize 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.

5.2.1 CUDA for game solutions

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

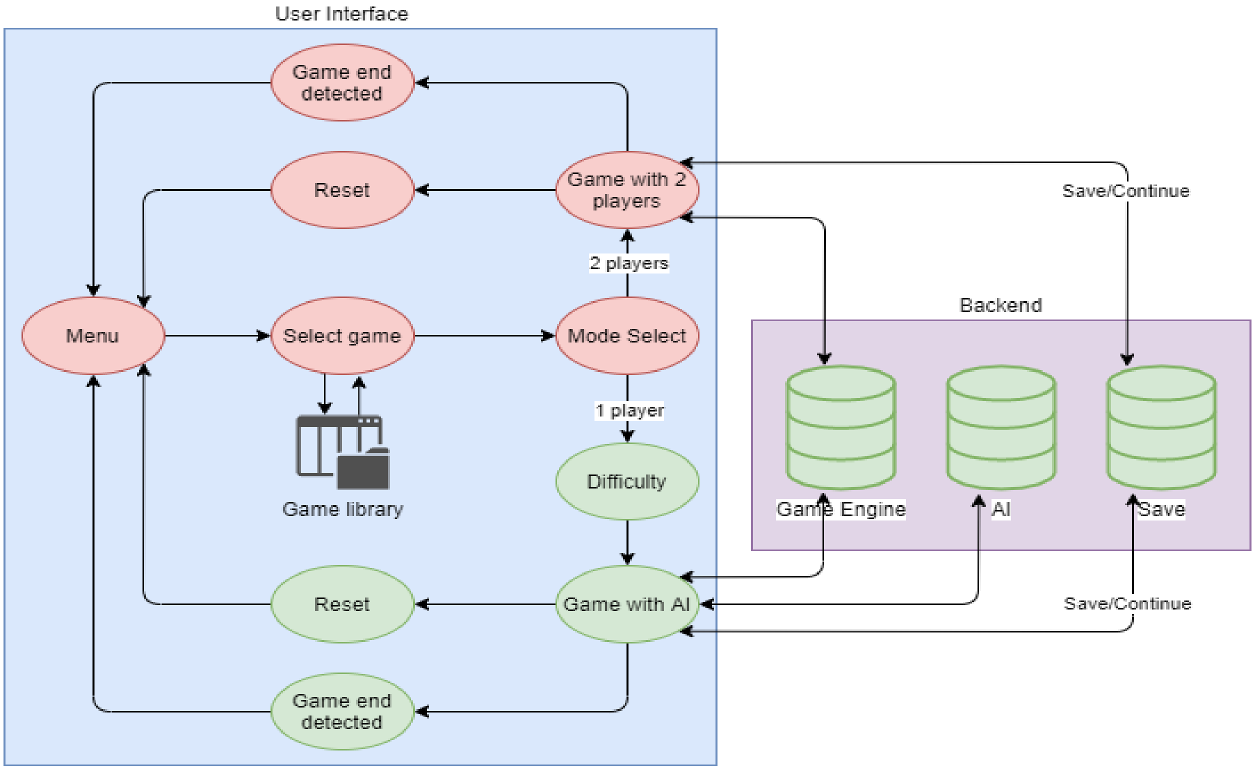
5.2.2 CUDA/Jetson Nano for Embedded programming

CUDA is not especially created to run generic embedded type software/hardware applications like the RaspberryPI or an Arduino board is, but it will still be more than plenty powerful enough to run these applications. This development kit is absolutely capable of what is needed for the software and hardware interfacing of this project. It may just be somewhat too powerful compared to what is really needed to run the GameFrame hardware components like the buttons, displays and speakers. This device has the necessary 40-pin header with I2C, I2S, SPI, and UART connection types.

5.2.3 Alternatives to Jetson Nano

The Jetson Nano is our choice of development platform primarily for the purpose of solving games when playing against the computer. Other boards would be able to run the games themselves, but may need toned down game-solving algorithms compared to what the Nano will be capable of running. RaspberryPI and Arduino boards are two that would be very suitable for running a game and even solving them, though runtimes may be annoying to the player. These boards feature all the necessary components, from embedded programming design to an application capability of running the games we are looking at designing.

5.3 Software



Red	*Israel*
Green	*Levi*

Legend - It is likely that everyone will have some hand in each part of the project, but this denotes who will be primarily focusing on each task.

Figure 5 (Software Block Diagram): The organizational structure of our software design at the highest level of abstraction.

5.3.1 Game Engine Design

The game engine will generate a new instance of the game on call of the create game function. This function will take in what kind of game it will create as a parameter, then it will initialize a blank state of the game that has chosen to be played. This function will be 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 will have 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 will have attributes to them such as “is_occupied”, “piece_type = knight,” “color = black/white,” etc to indicate what the current status and availability of each square will be. Each square will also be 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 will have an attribute as to what square they are on. Both the piece and square will need to be updated upon each move.

The game will check for a win status immediately after every move. Since only one piece will be moved at a time for most games, the checking function will only check pieces affected by the most recent move. The checker will check 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 will become 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 will progress 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 will be 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 will then end the game and send a signal to send a "stalemate" call to the interface or board.

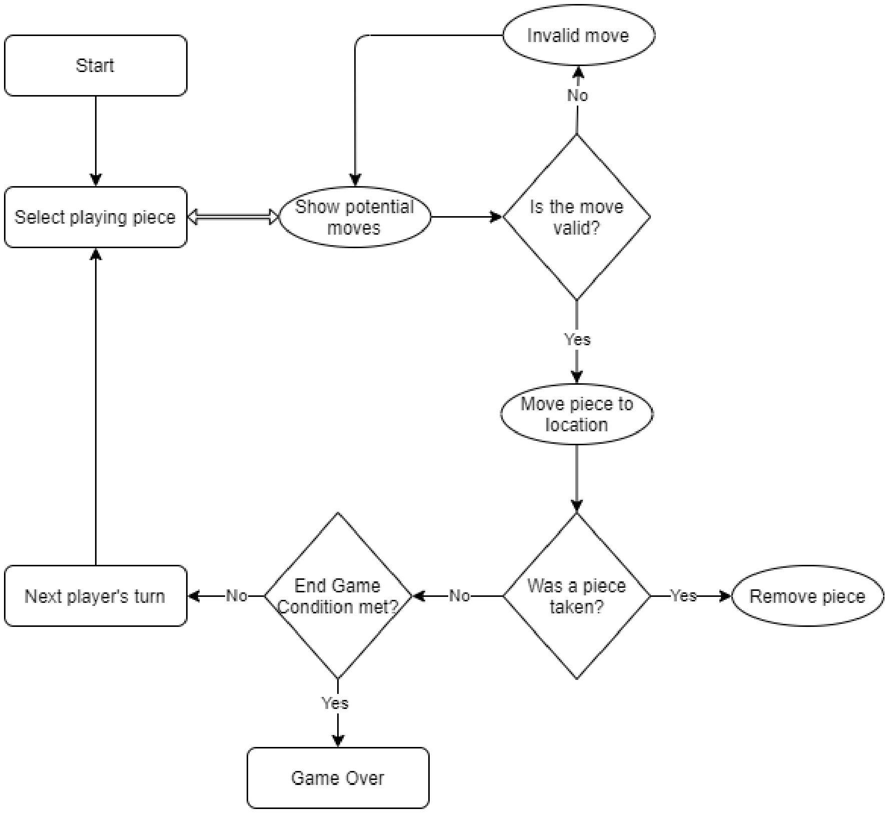


Figure 6 (Activity flow diagram): A diagram depicting a typical gameplay loop.

The storage of squares will be in a two-dimensional array. This will be consistent with the conventions of calling squares by their column and row. Squares in the array will be objects, and declared to have all their individual components assigned initially to zero (or None), but will be assigned their true initial value depending on the create game function. Game squares will have attributes related to all games declared, but only those in relation to the game currently being played will be modified on call of the create game function. Storage of pieces will be within one of multiple different one-dimensional lists or within individually declared variables, depending on what game is being played. Like the squares, pieces will be automatically declared zero/None or default values that will be modified according to the create game function call. Game pieces will be treated as objects with attributes and modifiers.

5.3.2 Player vs. Computer

One key feature of the GameFrame is that one user can play a complex game against the computer. We will utilize an algorithm based either on artificial intelligence or on backtracking in order to have the computer solve for its optimal move. Once the machine is set to one-player mode, this will cause 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 will have 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 will be used, but others are available as well.

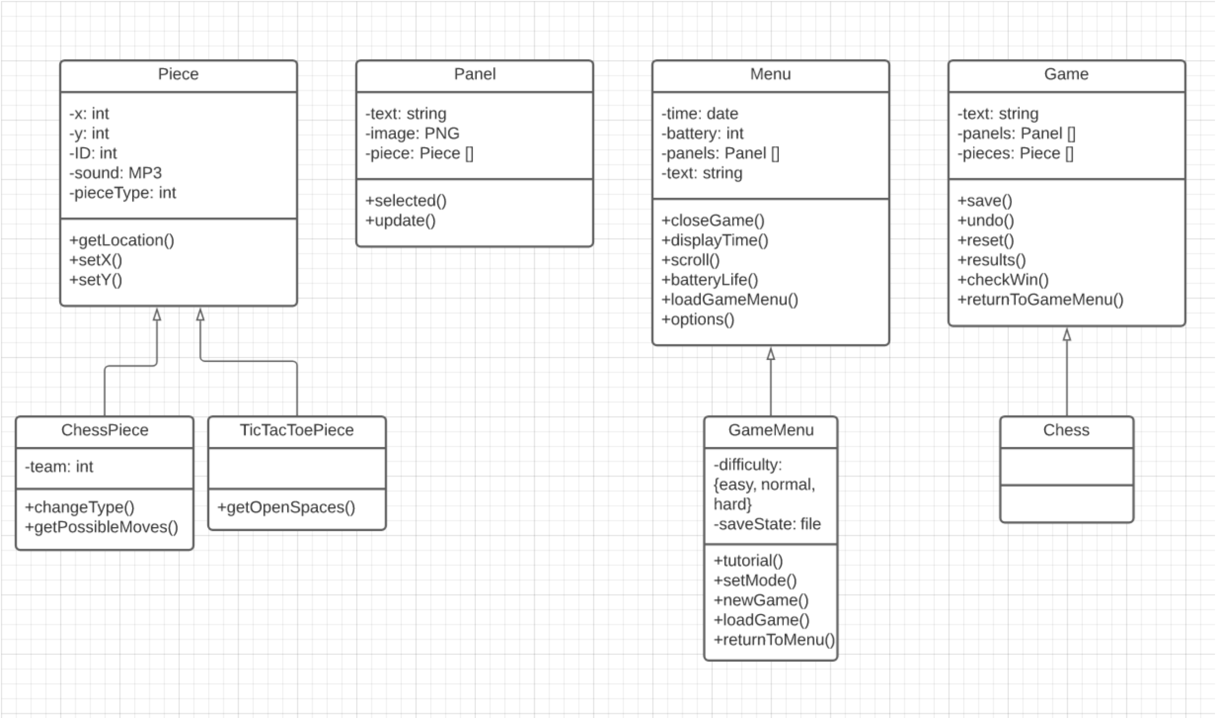
In an artificial intelligence design, we will have the machine run many different instances of the game against itself and reward it based on how well it does. To create different levels of difficulty, we will add randomness into the algorithm itself, or even reward it differently depending on the different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

If backtracking is determined to be the optimal way to solve a game, numerous different varieties of solving will be implemented in order to determine the difficulty level. In a particularly complex and effective backtracking algorithm, almost every different path that can be taken will be 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, 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.

5.3.3 User Interface

The part of the software that will be interacting with the user will be the main menu that we are constructing and the game software themselves. Once the main menu is displayed, the user will interact with the menu through the use of the buttons on the device. The user will select which games they wish to play and the GameFrame will direct the user to that game's menu screen. Once the user is directed to the game's menu screen, the user will set the options of how they wish to play that certain game, such as computer difficulty or if they will be playing with another human. The software will take the information given by the user and create the game that they want to play. Once the game is set up, the user will be communicating with the game trying to follow the game's rules. The software must be able to make sure that the user is following the rules and the software will be showing the user how the game is progressing. Once the game is finished, the software will inform the user if the game was a victory, loss, or a draw. The software will then ask the user for it's next decision, whether the user wishes to replay with the same settings or if the user wishes to head back to the game's menu.

5.3.4 Class Diagram



<u>Figure 7 (Class Diagram)</u>: The class diagram for our software.

For our class diagram we separated our code into the four separate classes shown above. We will be elaborating on how each class operates individually and how it coincides with the software as a whole.

Menu Class

The Menu Class is essentially the beginning of where the user will be taken once they begin to use the Game Frame. Here the user will see a screen similar to the one in Figure 5 in the software prototype. Here the menu must be able to display: the time accurately based on the user's setting, the list of games that are available, the battery percentage remaining in integer form, and text for the game's name and for displaying the product name. The menu will also consist of panels which are a derivation of the class Panel. Using these instances of Panel we will be able to create certain panels to display text and decide whether they will have functionality or not.

The menu must also be able to perform certain functions such as starting up a game, displaying the time, being able to show the correct battery percent, and load the options for user customization. There is also a scroll() function that we have that will allow the user to scroll between the game selection panel. This function will be implemented only if we have a game library that exceeds the normal menu page and needs to have more space for other games. Once a game is selected from the menu to play the menu will access that game with loadGameMenu() and this will lead into the game's own menu. This is shown in Figure 6, Game Menu, in the software prototype section.

This menu is known as GameMenu in the class diagram above and is an inheritance of the Menu class. GameMenu will perform the same as the main menu but it will be tailored to that specific game. It will look similar to the main menu but it will also have a tutorial for the specified game and it will also keep the user's last game if they saved it. GameMenu is also where the player specifies whether they are playing with a computer or another person with setMode(). If they play with a computer they will then pick a level of difficulty for the computer, either easy, normal, or hard.

Panel Class

In the Panel class, this is where we will focus on the individual panels that will be displayed on the screen. Each panel will be serving some sort of purpose, they will either display something, have a function or they will just showcase the background. For example, we already discussed the main menu briefly in the previous class section. The way the panels will be used in that scenario is that the panels that lead to important features such as the settings or to games will have the function selected() and will follow the function of being selected. Other panels on the screen however will have no functionality to them. Meaning that once they are pressed, nothing will happen. This will happen to any unused panels or panels that are simply used to just display text. Panels that are completely unused will simply be used to display the background that we decide to place.

Panels that are used in game will also need to be aware as to which game piece should be displayed, for example like displaying a pawn or a knight in chess. This is why Panel has reliance on the Piece class to appropriately display the correct piece. The panels will also have to update themselves whenever the player makes a move that is allowed. The panel selected will update and then update another panel that the user has chosen to move. Also we will also be adding in features such as pressing certain panels for a certain amount of time to perform some feature like an undo turn. This will be further investigated if time allows us and depending on the game itself.

Piece Class

The piece class will be responsible for storing the different pieces from different games and their movement. The x and y variable will be responsible for the location of the pieces on the board. Since our design is a grid the x variable will be the horizontal location and the y variable will be the vertical location. Both of the variables will range from zero to seven since there are eight panels available for use. Each piece will also have an associated sound attached to them to add a nice sense of realism when moving the pieces. We will be saving the sound files as MP3 files and the images we need as PNG.

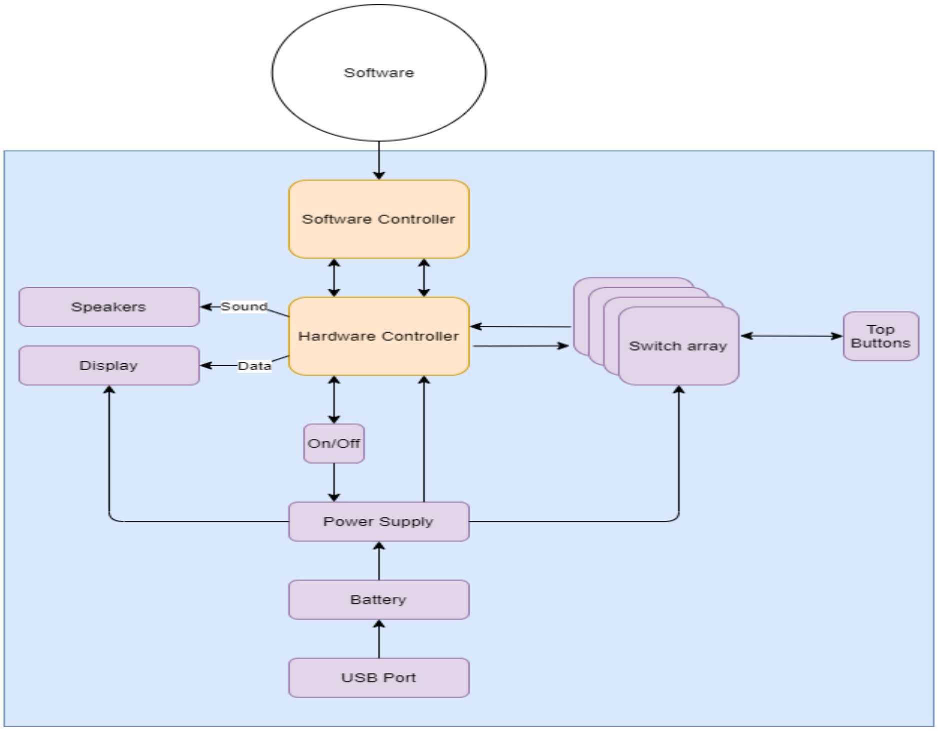
Each piece will have a corresponding ID that will make it unique between itself and the other pieces. Every piece will also have a type. For example, if the game was chess the piece itself would have it's own ID but the pieceType could be like a pawn or a knight. Based on this typing, the piece will be allowed certain movements. We placed example subclasses that would inherit the Piece class, more will be added based on the amount of games we implement into the system. The subclass for ChessPiece will differ by having a function that allows the piece to change types. This function will be only for the pawn types as they can switch to any piece once they reach the opponent's side of the board. Something similar will be implemented for checkers as once the piece reaches the opposite side it can become a king.

Game Class

The Game class will be the actual game that the user will be experiencing. Figure 7 in the software prototype section shows an example of the game being played out. Here the game will either begin a new instance of the game or from the game menu the user will load an old save file. The Game class will need to utilize the panel class and piece class to appropriately set up the game. Once the game is set up the game will be making sure if there is a victory or a draw with the checkWin function. While the game is being carried out, the game will also have the ability to save the state the game is currently in so that the player may resume at a later time. During the game, the user will be able to perform an undo feature in case they committed a move by mistake. This will be performed by issuing a command that we will agree upon at a later time such as pressing two specific panels on the device for a certain amount of time.

The game will also have a feature to return to the game's menu and if they did not save it will ask them if they wish to save before leaving or else they will lose any unsaved progress. If checkWin sees there is a winner or a draw then the results function will lead to the result screen where it will ask if they wish to reset the game and retry or if they wish to go to the game's menu. The chess subclass is just an example of the games that will be inheriting from the Game class. The majority of the functions and attributes are already in the parent class however, if needed additional functions will be added to specific games.

5.4 Hardware



Purple	*Frank*
Orange	*Allen*

Legend - It is likely that everyone will have some hand in each part of the project, but this denotes who will be primarily focusing on each task.

Figure 8 (Hardware Block Diagram): The organizational structure of our hardware design at the highest level of abstraction.

5.4.1 Hardware-Software Interfacing

The main hardware components that will be interacting with the software will most likely be the 64 buttons on top of the screen. However, most microcontrollers do not have enough pins to connect one button to each pin. For example, while the Jetson Nano has 40 pins, only 28 of them are GPIO and even less of those are set to GPIO functionality by default. Some of these pins are used for I2C and UART protocols by default and have GPIO as secondary use if needed. To be able to utilize the limited number of pins to connect all 64 buttons, it will be necessary to create a button matrix to connect multiple buttons to a single GPIO pin for input detection. While it is possible to wire all 64 buttons into one pin, it can get immensely complicated trying to uniquely identify each button on a single pin and cause a lot of unnecessary work. It will be a lot easier to divide the buttons up and assign them to multiple pins. To strike a balance between pin efficiency and simplicity, we will probably interface four buttons to a single GPIO pin. As a result, only 16 pins will be needed for input detection. This will leave plenty of other pins for communication such as I2C and SPI.

For the GPIO pins to detect input from the buttons, it is likely we will be using I2C as the protocol for communication between the buttons and the software. I2C has plenty of advantages for our use such as support for multiple devices and only using two bidirectional wires to communicate between multiple devices. One of its main drawbacks is the relatively slow speeds since I2C requires a pull-up resistor. However, this drawback should only be negligible, if not completely irrelevant. The only information that will be transmitted is either a 1 or a 0, which correspond to when a button is pressed.

While some pin headers use 5V signals, most of them, such as the Jetson Nano, use a 3.3V signal. As mentioned previously, the pin will either read a 1 or a 0. It reads a 1 when a 3.3V signal is detected, and it reads a 0 when the pin detects ground. For the buttons to utilize this functionality, The button will open or close the circuit that sends the signal to the pin. While the button is idle and not pressed, the circuit is open, and the pin will read the 3.3V which results in 1 as the input. When the button is pressed, the circuit is connected to ground and the pin will read a 0.

To interface these signals in a way that software can use it, it is likely that both the embedded code and software code will be processed by the microcontroller's processor. The embedded hardware will translate the signals created by the buttons and communicate directly with the software. An example for how this interaction would work is the controller detects that a pin went from 1 to 0, processes which button was pressed, and sends that signal to software to process the selection. The software will then send the data back to the controller, and the controller will determine where to send that information based on where the pin detects a button press to mark where a player would like to place a piece.

Another way to interface this is to have separate controllers for the embedded hardware code and the software code. Going this route, the two controllers will communicate with each other through the SPI protocol since it is the fastest of the three. Which way we decide to interface the software and hardware will heavily depend on the features of the microcontroller(s).

5.4.2 Hardware Controller

We will use a separate microcontroller in order to handle the hardware. Most likely, this is to be an MSP430. The MSP430 has a convenient development kit which lets us load the program onto the MCU chip before socketing it onto our PCB. It is easy to use, especially since it is an embedded processor we are familiar with from our course work. More than ease, the MSP430 is on the cheaper end of microcontrollers while also being power efficient. Overall, it is considered low-power.

The main function of the hardware controller will be to manage and relay the array of switches to the Jetson Nano. Depending on how we implement sound, the hardware controller might also relay information from the Jetson Nano to a sound chip. On the topic of sound, if we decide to include audio input, the hardware MCU might be the avenue in which to convey the information to the Software Controller.

The initial model of choice is the MSP430G2553. This is exclusively because of our decision on a G2ET LaunchPad. The LaunchPad allows us the flexibility of using any MSP430 with 14/20 Pin DIP capabilities and the ease to move it to and from our PCB. It just so happens to come with an MSP430G2553. If the features need to be refined, we can look into other models that fit the DIP. Such models are detailed in the table included in the research section discussing the MSP430. That section also discusses the features in question.

5.4.3 PCB

It is reasonable to organize the hardware into up to four separate PCBs. Likely, we will use less than this. For example, if we end up utilizing a small sound system, we might be able to designate a separated (ie: not connected) corner of the main PCB to attach the sound chip to. The reason there would be four is for each potential subsystem of the hardware. The subsystems we have are the buttons, sound, power, and display.

PCB-1 (Main PCB)
The first potential PCB will focus on being a home for the switches and the hardware microcontroller. This should be our primary PCB. In terms of design, we will have a grid of switches on a large section of the board and probably relegate the hardware controller and other ICs to one side. An approximation is depicted in the figure below.

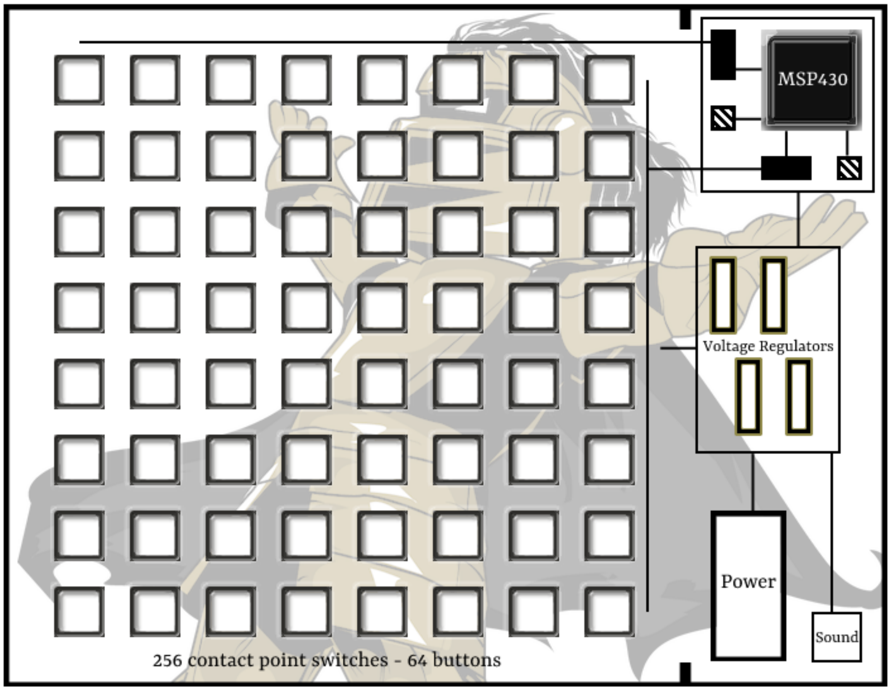


Figure 9 (Main PCB Concept): Rough diagram of the main PCB. Unlabelled components include clock ICs (striped) and shift registers (solid black).

This picture is a not-to-scale representation of the PCB. It is meant to be a block diagram level depiction loosely showing the relationship and components of the main PCB. The amount of space taken on the side of the switches will in reality be much smaller, since the chips and ICs will not be bigger than the size of the switches. The power and sound might have their own separate boards depending on space and amount needed by those circuits. The number of voltage regulators is to be determined. The association between voltage regulators and individual components is loosely represented here.

For the features of the board, we currently need a socket for our MCU, a variety of voltage regulators, a pair of 8-bit shift registers, a pair of clock ICs, and the contact points for our switches. The socket will be determined by the controller we use. Since we are leaning towards an MSP430 at the moment, the socket number will still depend on the individual model of MSP430 we choose. That is, if the MSP430 is the microcontroller line we end up using. The voltage regulators will be a variety of ICs we use to make sure that each of the components on the board receive the correct voltage. Boost converters will be utilized for raising voltage when appropriate and buck converters will allow us to do the opposite - lower the voltage for parts when needed. The plan involves using two 8-bit shift registers in order to read all of our switches. The reasons behind using the two 8-bit register ICs will be explained later in the Hardware section of Design. Specifically it will be covered under “Switches Layout.” Likewise, we will also explain the need for integrated circuits to provide a clock. We need one clock IC for each 8-bit shift register. Lastly, the first PCB will provide the space for our switch functionality, probably via trace contacts that we will complete with conductive push buttons as opposed to component part switches.

Since there is potential error with switch bounce, we would need to accommodate this in our design. The initial PCB design will include room to put in circuitry to handle hardware debouncing if it is needed. These can be slotted with 0 ohm resistors if they are not of use, and removed in future iterations of the design if that is the case.

PCB-2

Another possible PCB would be for the sound. Since the circuit will mostly involve just a simple chip, it might be better to implement this in the corner of the first PCB as mentioned prior. This is so we do not have to fab an extra board. This also lets us minimize wasted space on the first board. If the sound subsystem is too complex or the main PCB does not have space, the sound system will need its own board. The sound chip might not be needed though if we output audio from the Jetson Nano via HDMI. Obviously, this PCB would not be needed if that is the case. It would require voltage regulators on it for its components if the PCB is needed and is implemented as a separate board.

PCB-3

There will likely be cause to have a board to manage the power circuitry. In particular, a PCB designed around charging the battery. The power board would be home to the power button. Like with the sound, a separate board might not be needed depending on the space of the main board. Even with that consideration, isolation could be beneficial. We know that the Jetson Nano runs hot, so we would like

PCB-4

The PCB least likely to be needed is this one - the display board. The Jetson Nano can interface with the display directly via Displayport or HDMI, provided the component we get for display has either of those standards. Should the monitor also have sound capability (say, by HDMI) then this would also eliminate the need for a sound PCB. The display will still need voltage regulators, so this board could be their home. A small disconnected section on the main board could suffice as well if that is all we need for the display circuitry and provided it has room.

Jetson Nano

As a quick note on the Jetson Nano, it will probably be implemented as it comes in its development kit. This lets us utilize the full functionality of the kit rather than trying to replicate these ports and features. There is no need to consider another PCB for it.

5.4.4 Switches Layout

There will be a total of 256 locations for contact switches. These correspond to the four corners of each button. Each set of these four contacts should register as the same button press. The contacts can be tied into a four-to-one manner in order to logically OR them. Even with this, there would be a total of 64 wires to the switch array. The MSP430 does not have that many I/O ports. In order to reduce the number of inputs and outputs to the MSP430, shift registers will be used to feed the information in serially. We chose two 8-bit registers as a way to minimize the 64-buttons into eight wires in and eight wires out, corresponding to the eight rows and eight columns of the device. The fact that we want to do the input and output serially means that we will need one clock for each, or two clocks total. These ICs will be used to keep the timing of the serial in and serial out for the two registers.

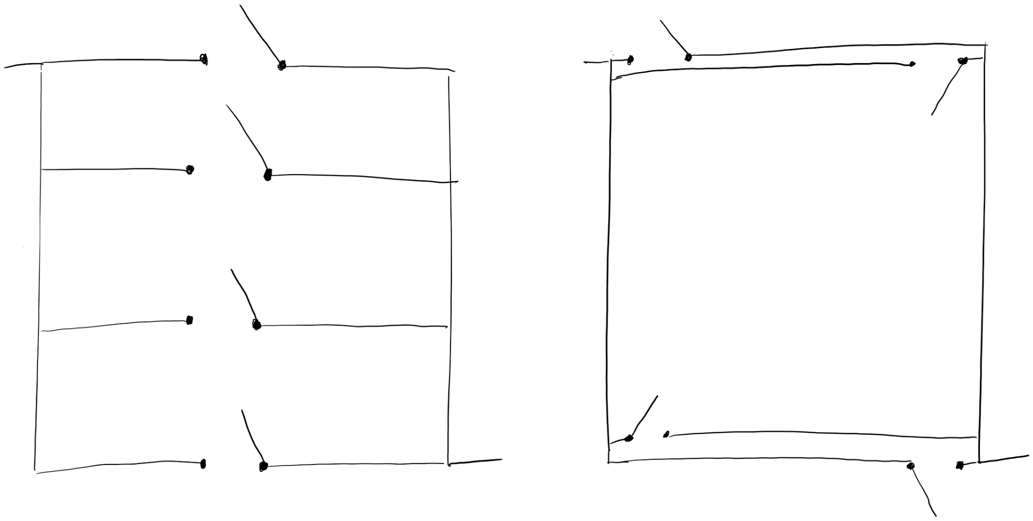


Figure 10 (Hand Drawn Switch Schematic): A basic picture of how each button's switches would be at the circuit diagram level.

In the previous picture the idea of tying the switches together as 4-to-1 is shown. By having four separate switches connecting two nodes we are able to logically OR them as mentioned before without the use of any additional components outside the four switches. The left part of this picture shows an easy to understand implementation of this. Our display must obviously be visible behind the button panels though. On the right is an alternative visual of the same thing. The alternative drawing represents how we might implement this and have a switch in each corner while the middle area is visible for the display behind it.

The potential switch bounce which might occur could lead to a very poor play experience. Additional circuitry attached to the four-to-one buttons can smooth out this interference as our switches open and close. There are several options for resolving this through hardware. A simple RC addition for each group of switches seems like the decision we most likely will choose if we need to rely on a hardware solution. We will first try handling debouncing on the software side.

Another quick note mentioned elsewhere is that it is more efficient and less costly to utilize and complete trace contacts on the PCB for buttons rather than buying 256 individual pin component switches.

5.4.5 Sound

A feature suggested to us was to include a sound subsystem. Sound adds obvious enjoyment quality to games and allows additional games as viable on the device. The current objective does not include this feature, but it would make a good stretch goal. Adding a sound system would add extra complexity and infringe on our existing weight, cost, and power goals. The aforementioned goals hold priority. Potential AI applications are on our mind as we contemplate new additions. A microphone with voice commands is another consideration related to a sound subsystem. Voice commands could be something we could integrate into the Game Frame by having it select the game the user wishes to play. Another feature we could add in if time permits us is we could also have an option so that the user can choose to play chess through the use of verbal commands.

Other features we would like to implement if time permits us is to add sound files to pieces while the game is in progress. We feel like it would add a nice sense of realism to the game to be able to hear small audio cues of the pieces being moved. We also would like the game to be more accessible to others so something else that we can add if time permits us is a feature to allow for text-to-speech to aid those who are visually impaired. This would verbally read aloud the text on the screen and guide the user to be able to select the panel they are trying to reach. The feature could also inform the user where the pieces are located on the board and inform the user the piece that they currently selected and where it is allowed to move.

We have concluded that the quality that sound adds to our machine is worth designing for. The plans are to have, at minimum, a simple sound chip or output. The idea is for the software controller, the Jetson Nano, to be where the sound comes from ideally. Due to various options and not wanting to limit ourselves yet, we are also open to the MSP430 processing the sound.

As mentioned, the Jetson Nano is the ideal place for the sound to come from since it will be processing the sound. The sound design can, however, instead store audio on the MSP430. The

Jetson Nano can feed a signal to the MSP430 for the sound in such a case, but the MSP430 resources might be tied up for other processing. The MSP430 also has a bit more limited memory space, so storing audio data could infringe on this. HDMI from the Jetson Nano seems the most pragmatic design choice for the sound.

5.4.6 Clock and Shift Register

Rather than using a clock IC, output lines from the MSP430 can be used to strobe the shift registers. This would result in three output lines and one input line. Shift register in and out and two clock outs. The frequency is set at a maximum of 25-29 MHz according to the rating on the 74HC595 8-bit shift register datasheet. The clocks are utilized by these aforementioned registers. The range of frequency is determined by the supply voltage to the register IC. Whatever the case, we will doubtfully utilize the maximum frequency. The intended supply voltage is around 3.3V to keep it consistent with the MSP430.

5.4.7 Power

Part	*Voltage Ratings*	*Current Ratings*
MSP430	3.3V	1-3 mA*
Jetson Nano	5V	1A / 2A
LCD Display	3.3V	110mA
LCD Backlight	5V / 12V	650mA@5V / 240mA@12V
Shift Registers	2-6V	1mA

*The current rating of the MSP430 is dependent on the frequency it is run at. Even with varying current it still only operates at 1-3mA for the supply voltage, quite small.

Table 5 (Power Relevant Ratings): This tabulates the various voltage and current ratings for our hardware components.

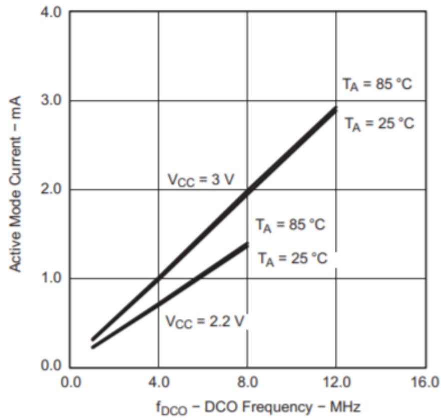


Figure 3. Active Mode Current vs DCO Frequency

Figure 11 (Supply Current Range MSP430): The datasheet provides an excellent graph to show the current draw based on operating frequency and supply voltage. It also tops it off with convenient information on temperature ranges too.

Listed for the sake of denoting option variants, the table shows two voltage ratings and their associated current ratings for the LCD Backlight. The 5V is the minimum voltage rating for the device and probably will not be selected. If we could budget extra amperage, and are willing to allow a dimmer backlight, the 5V rating would allow us to run every component from the switching regulator. We would eliminate the need for a battery that goes up to 12V. The power consumption is a tad more for the 5V rating though.

Voltage Rating	*Total Current Draw*
12V	0.24A
5V	2A*
3.3V	~0.115A
Total Current	2.355A

*Our total currently assumes high power operation mode on the Jetson Nano. The table is easily adjustable should we choose to change this.

Table 6 (Supply Current Range MSP430): A total of the amperage drawn at each voltage level. This is for ease of calculation and reference for our switching regulator.

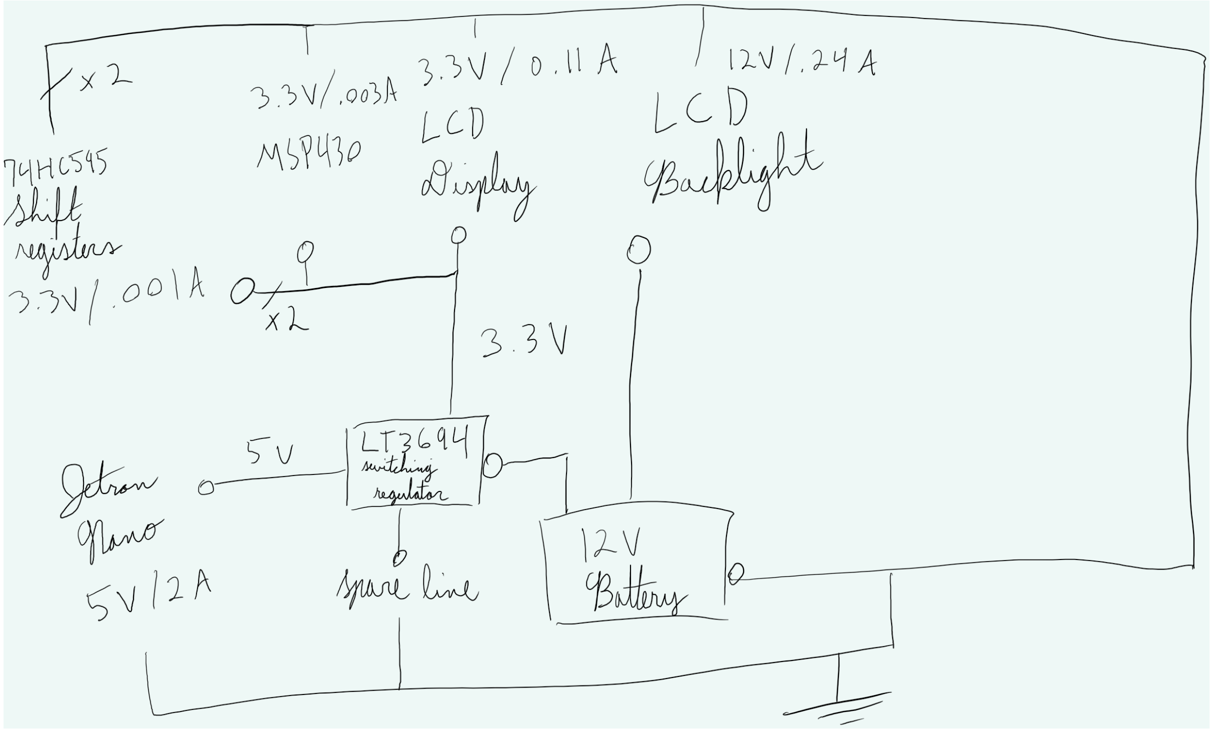
It is worth mentioning that the 12V comes from the 12V battery directly and not the regulator. So even though our 2.355A total is below the 2.6 maximum that the regulator can output, there is actually an additional 240mA that can be utilized from the switching regulator.

Battery

The battery is intended to have 12V so as to be able to supply the required voltage to the LCD backlight without needing any boost converters. Since we are allowed to use a wall wart for our power supply, it is best for safety and compatibility to base our choice for power supply on the battery.

Switching Regulator

The LT3694 allows us to step down our 12V battery to different voltages for multiple output lines. The maximum current output it can supply to various parts is 2.6A. If we run the Jetson Nano in the 10W mode this will leave .6A for the other parts, which is likely feasible. This would leave little room for additional parts or expansion of hardware design though. The 5W power mode for the Jetson Nano is also considered and would leave us with more of a cushion of 1.6A from this regulator alone to power the rest of our components.



<u>Figure 12 (High Level Power Schematic Design)</u>: An abstracted block-diagram level picture showing voltage and current to our device's components from the regulator. Location and layout is not to be considered physically accurate.

5.5 Decision making

5.5.1 Project

Project Name	*Project Complexity*	*Familiarity with Technology*	*AI Requirements*	*Personal Interest*	*Mechanically Challenging*
Smart Blind	Neutral	Neutral	Low	Low	Low
Poker Hand	High	Low	High	High	High
Refillable Station	High	Neutral	Neutral	Neutral	Moderate
GameFrame	Moderate	High (consultant)	Moderate	High	Low

<u>Table 7 (Project Decision)</u>: Project ideas taken into consideration.

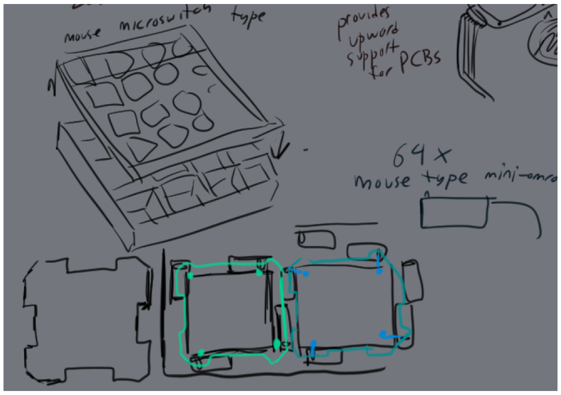
The interest for the poker playing robot was the highest, and it was the project with the most prominent AI development and computer vision. The portable game device ended up having good interest (though admittedly slightly less). The benefits of the game device ended up being that it was complex enough without being overly complex, we had an outside source to consult with, it was not very mechanically challenging, and it had some component of AI with which we could implement. This allowed those who wanted to develop those skills to do so while also maintaining a solid level of project interest. Thus, it is what we decided.

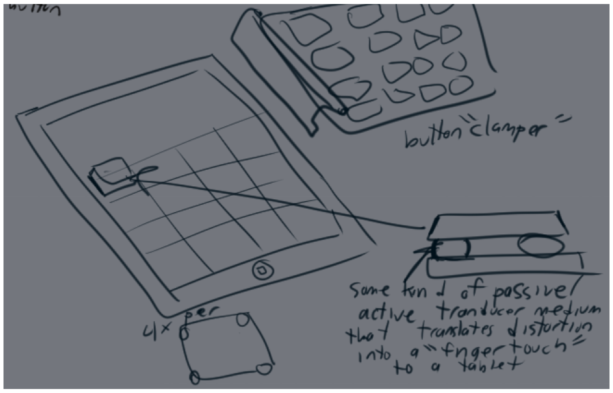
5.5.2 Microcontroller

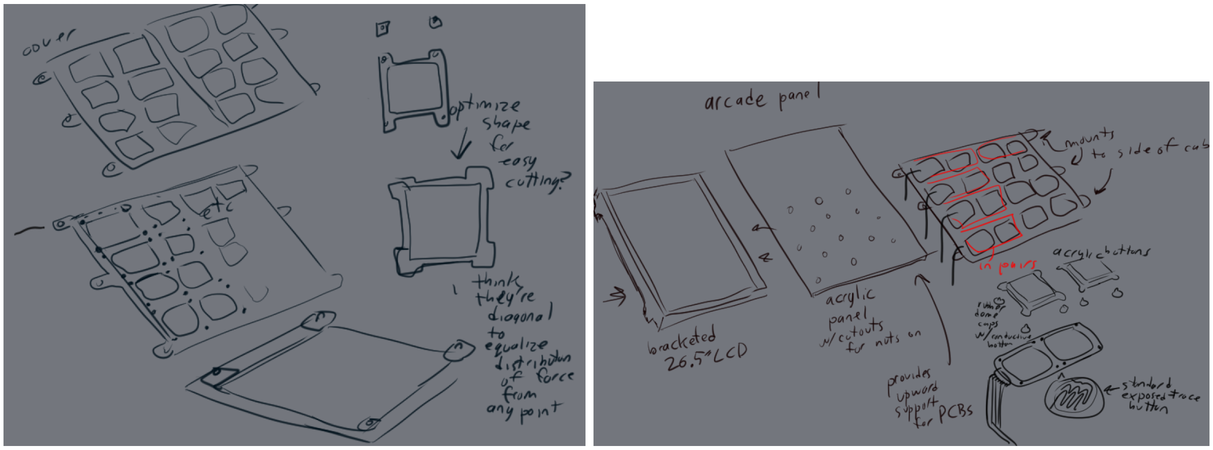
The decision is not set in stone until purchased, but we have honed in on the Jetson Nano as our microcontroller of choice. We believe it will be a great learning opportunity and provide valuable experience to those interested in parallel computing and other technology useful to the AI field. We considered how it has extra cost and power requirements, but decided that it still is within our acceptable range and might actually help with those fronts instead. Since we do not exactly know how much processing power we will need, we figured it was better to be safe and go with a board with more capabilities out of the box. We want to avoid any bottleneck issues during development that can be caused by using a cheaper and less powerful microcontroller. The statement that most summed up our feelings was “it sounds like the pros outweigh the cons.” As we delved into more research on it, we felt that it was suited to our interests and more than sufficient for our goals. The only other thing to mention about a decision regarding computation devices or controllers is that we are using another controller to manage our hardware. The division of the tasks is better accomplished with separate devices. Putting less strain on the Nano also helps alleviate some of its heat generation.

6. Prototyping

6.1 Prototype Overview




button
button "clamper"
4x per
Some kind of passive/active tranducer medium that translates distortion into a "finger touch" to a tablet


cover
etc
optimize shape for easy cutting?
think they're diagonal to equalize distribution of force from any point
arcade panel
bracketed 26.5" LCD
acrylic panel w/ cutouts for nuts on
provides upword support for PCBs
mounts to side of cab
in pairs
acrylic buttons
rubber dome caps w/ conductive button
standard exposed trace button

<u>Figure 13 (Prototype Sketches)</u>: A collection of rough diagram we were given

CAD prototyping is a goal around the corner as we develop more specificity, but our current early-stage visual prototype for the project is based around a video of DIY jubeat assembly which can be found in the appendix and on a rough basic design diagram of the physical components provided to us by our contact (Prototype Sketches).

In order to add more clarity to parts of the diagram a description of the components might help. Of particular note is the PCB and the acrylic panel. The bottom right shows an example of jubeat's PCB. The machine uses eight different two-button PCBs. Due to the large holes and thin PCB, a flimsiness exists. The acrylic back panel helps to keep the LCD screen clean of debris of course, but it also functions to add support to the PCB. One other thing that is not clear just from the diagram is that the buttons have a specific diagonal bevel in the corners. This causes a better slide and feel when the button comes back up. As of the moment, we do not have a cost-efficient way to replicate proprietary acrylic, but this might also not be needed for chess to feel comfortable nor might it be needed for our smaller button size.

Our goals with prototyping focus on cutting costs to create functional controller systems at the moment. It can be noted that this will not allow us to reliably tell if we are meeting the weight or power requirements. If our prototyping breaks either requirement or comes close to the values in those requirements however, we will be able to tell that things will need to be rearranged or reconsidered.

It has been decided that using the microcontroller development kit as it comes will be useful in familiarizing ourselves with the technology and getting things started. As things unfold we might find ourselves gutting features or implementing only the necessary features into our device to save on requirements.

6.2 Software

In order to test the software, a simple graphical interface will be created. This will never realistically be something that is interacted with by the end user, and will eventually be thrown out completely once the full build is completed. This software prototype app serves as a tool to test the functionality, rule implementation, and capabilities of the computer to play against the user in a game of player versus computer. Additionally, this application may be used to monitor training of the artificial intelligence or backtracking algorithm.

6.2.1 Automated Testing

To test certain components of the software, automated testing will be useful. For instance, the artificial intelligence will need to play many rounds of games against itself on different difficulty levels in order to determine whether or not the difficulties are different, and by how much. Automated testing will be performed for this by creating numerous instances of the AI that will play against each other. Many sets of games can be run simultaneously, as parallel computing is one of the key features of the jetson nano board. These tests will record each individual move throughout the games, the time to completion, and win status corresponding to each bot.

Individual aspects of the AI can also be testen nicely through automation. Specific types of games can be configured to ensure the AI does not perform an illegal move, or get gridlocked in its decision making process. A series of games are created for the bot to run through at a few given states that have unique states to them, then the bot is told to make its move, which is recorded into a log. This will give the tester many different examples of individual and finge testing scenarios that allow us to see deeper into the mid of the AI, and understand why problems may be arising.

Automated testing will be created for the game engine as well, to determine if all the components of the game are running properly. The automated tests will include moving all pieces in every square to every possible position, making numerous different capture combinations, such as in chess, a pawn captures a knight, kind captures a queen, pawn gets knighted, etc. For simpler games like tic-tac-toe, all possible games can be run, as only 255,168 total games can be played, a manageable number of simulations for the Jetson nano. Not every possible game needs to be automatically tested, as there are additional different types of tests in place to ensure functionality is available.

Testing the game won checker function is a task that lends itself nicely to automation as well. For this specific task, numerous layouts for a game can be created, fed to the game checker function, and the tester can simply check to make sure the game checker function correctly identifies all the games. This will be done throughout development of the game checker function, as different aspects of the game checker function will be developed at a time.

The automated tests for each component can be run regularly throughout the development process for each component, as well as through the development of certain components which may rely on other components. Automated testing in this way allows the testers to understand quickly which aspects of the software are working and how. These tests should essentially be run habitually, even when it is not necessarily determined that something is broken, or that a new feature is developed. The beauty of automated testing in this way is that plenty of useful troubleshooting data is available to the developer, with little dedicated effort beyond the initial development of the tests.

6.2.2 Sample Menu Prototype

Once the device is turned on the software will begin and will load the main menu software that we will design. It will look something similar to the figure below.

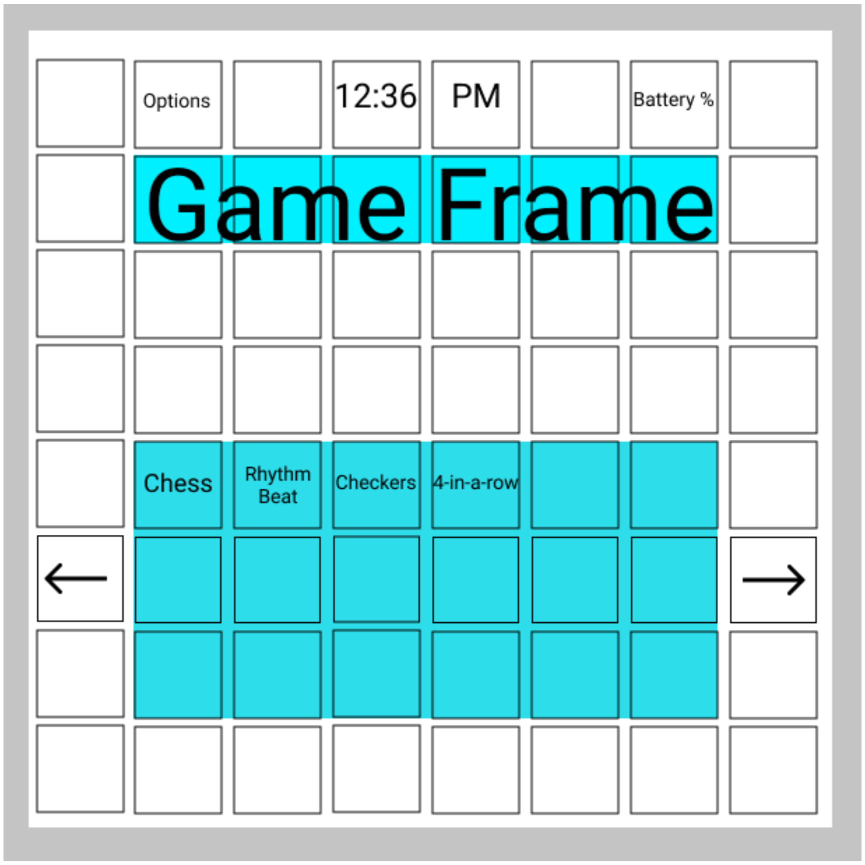
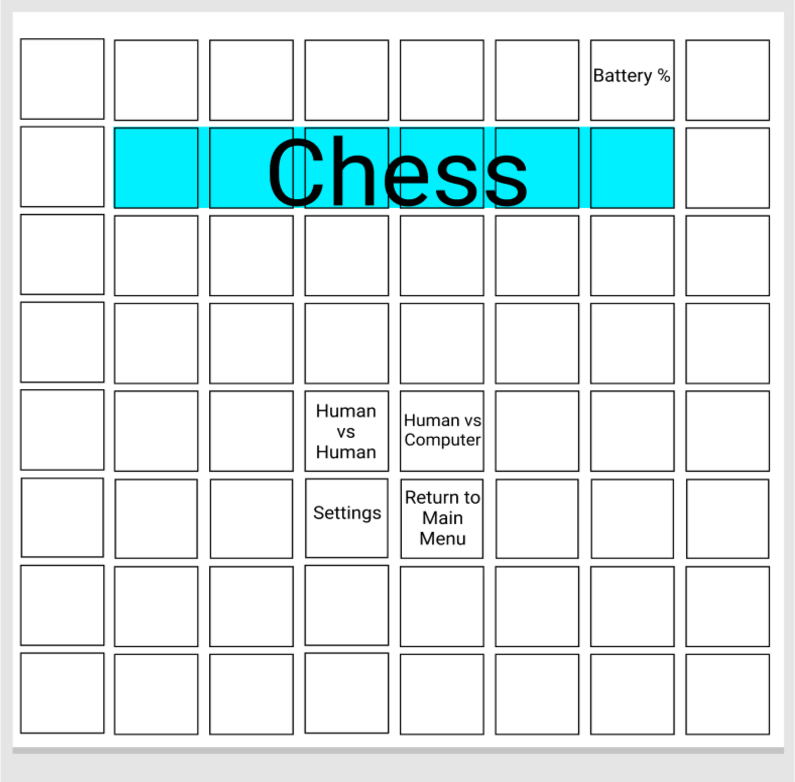


Figure 14 (Sample Main Menu): Pictured is how the display might look for our main menu.

The design of the menu will grow more complex and more features will be added as we see fit based on our time constraint. Key things to point out is that there needs to be some indicator of the battery percentage so the user knows whether or not they need to charge their GameFrame or not. The time will be displayed at the top of the GameFrame and the options will be to the left of the time stamp. The options will allow the user to set the time on the device and it will also have an option to account for daylight savings. We will also implement a unique design to our background so that the background has some sort of moving design. If time allows it, we can always implement more design choices for wallpapers that the user can select. Wallpaper options can vary from something bright so the GameFrame is more visible outdoors to something darker so it's a bit more soothing when playing indoors or in a dark environment.

The title of the product will be displayed at the top and then the library of games will be at the lower half of the screen. This may change based on how many games we end up having on the GameFrame. The games will have a frame around them for aesthetic purposes and each game will be selectable on a button. If we implement a lot of games, for example like 20, then we will add the arrow feature on the sides of the menu screen. The arrows are meant to be able to scroll through the library of games if the amount of games is too large to display on the screen. However if the amount of games is not too large then we can simply display them on the main menu without needing to have the arrow feature.

Once a game is selected, like chess, then the game will open up and it will be brought to that game's respective menu. Like in the figure below.



<u>Figure 15 (Game Menu)</u>: This is an example of how the menu will appear for games.

For the chess game, it will say "Chess" at the top and there will be options for the user to select such as: *Human vs. Human*, *Human vs. Computer*, and *Return to Main Menu*. It will generally follow this outline for each game in the library. If the user selects to play against a computer, the software will make use of the unused buttons on the side and ask it for a level of difficulty for the computer such as: *Easy, Medium, or Hard*. It will also ask whether the user would like to play on the white side or the black side. When the user selects a difficulty it will remain highlighted to show that is what the computer's difficulty will be. Once the user has selected a difficulty and team, or if they selected *Human vs. Human*, then the software will ready up the board. Every piece on the board will have a corresponding button assigned to it. It will look something similar to the figure below.

Another feature we can add on top of this is a tutorial panel where the user is informed how to play whichever game they are on. This would also be a good place to inform the user if we add in other features that are not visibly shown such as an undo feature. Since the only button on the device will be the power switch we can implement a feature where if the user presses certain buttons for a certain amount of time then a feature will happen. For example, if the user presses panel twenty-eight and twenty-nine for ten seconds, then it will undo the last made move. This will be stated in the tutorial panel to inform the user.

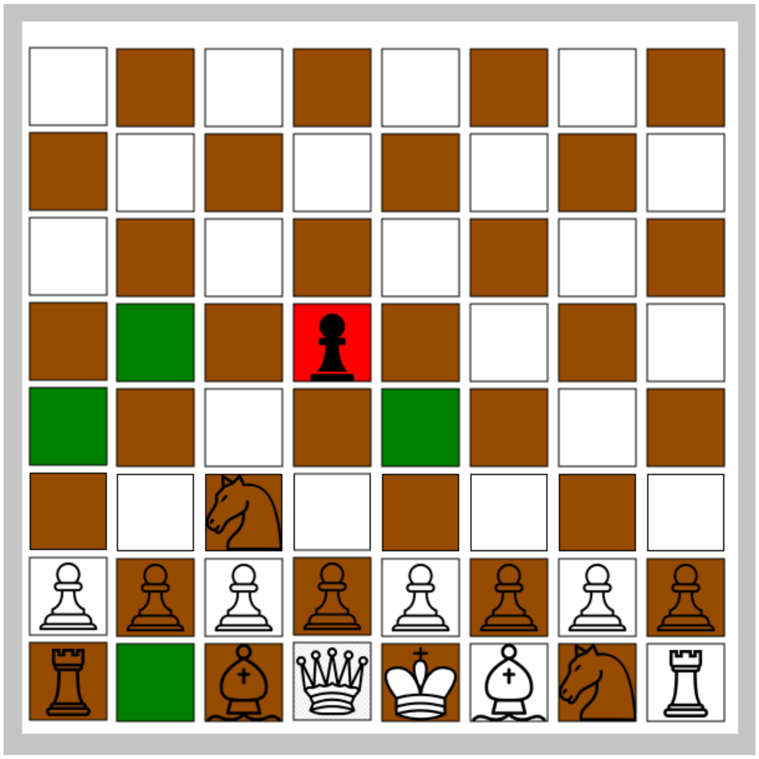


Figure 16 (Chess Gameplay Prototype): A design of how playing chess will be from a top view.

What this figure shows us is that when a piece is selected on the specific player's turn it will highlight green as to where the piece is allowed to move. If the piece is blocked by another piece and is not allowed to move to that square then it will not display anything. If the piece is able to remove an enemy piece then it will be highlighted in red. To move the piece the player will have to select the piece they wish to move and then move to the corresponding green square. Once the game is over, whether it be a victory, loss, or a draw, a small popup will appear asking the player if they would like a rematch or be taken back to the game's menu. Like in the figure below. When this happens, the user has the option to play the game again with the same settings or they can go back to the game's menu where they can change the settings of the game.

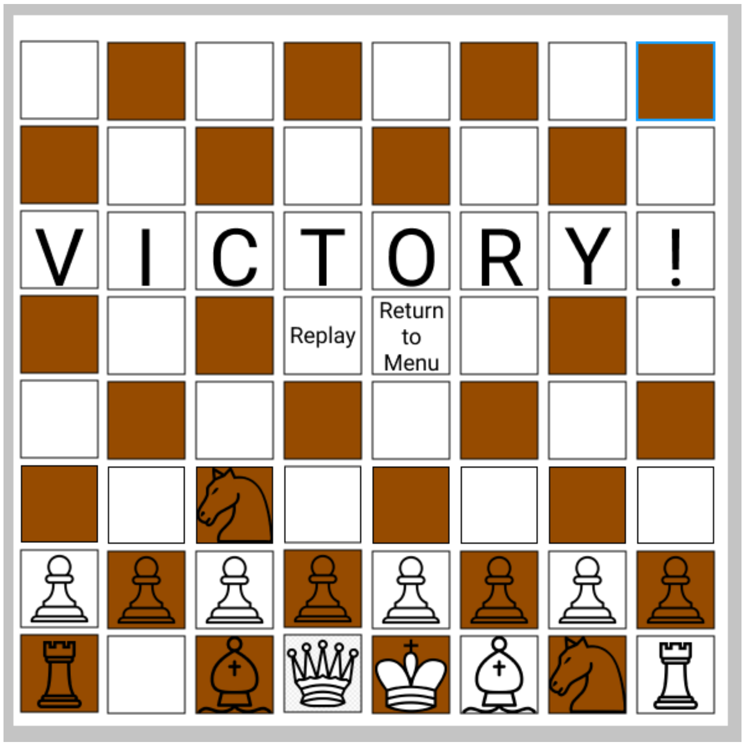


Figure 17 (End Game Screen): *This is an example of how the victory screen will display.*

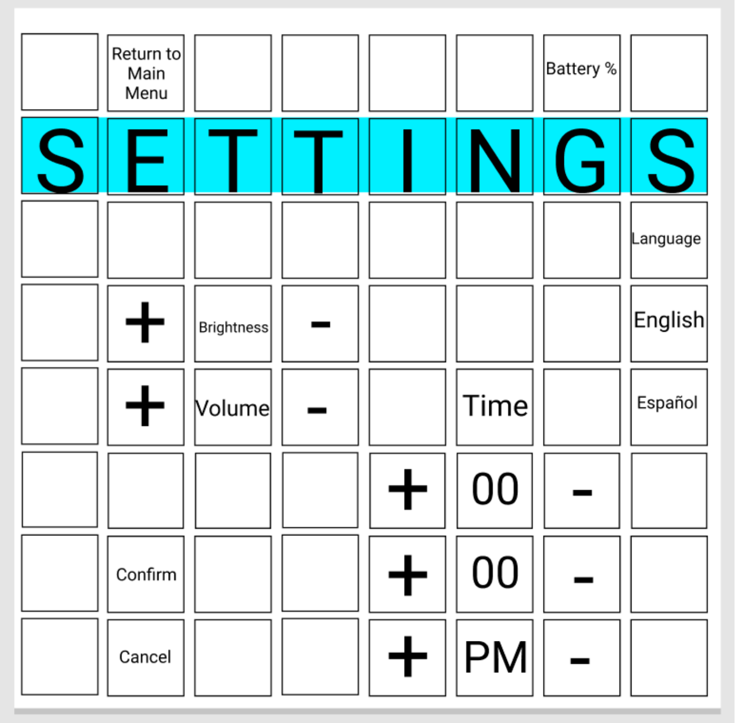


Figure 18 (Settings Prototype): *This is an example of how the options page will be displayed for the user.*

In the figure above we have an example of the settings page. Here, as the figure shows, the user will be able to control the brightness, the volume of the device, adjust the time, and change the language of the device. These are all functionalities that we will add if we have extra time but the main feature we will add will be the user adjusting the time. The priority would be to have the time functionality and it would look similar to the figure above. The user will adjust the time themselves with the appropriate panels and when they are satisfied they will hit confirm to have the time begin. Other features we want to add would also be the language feature to tackle the cultural constraint of having the device be accessible to other people who speak different languages.

The options page would look a bit cleaner than in the figure above. Normally when going to the options page, the screen would only show the settings such as brightness, volume, languages, and time. However when a user interacts with the panel it will open up the options on the side to allow for adjustment of the setting. For example, Volume would appear normally without the plus and minus besides it but when pressed, it would look like the figure above. Here the user can adjust the volume. While adjusting the volume, there will be another panel that displays the percentage of what the volume currently is and how it is changing. The language panel will also work the same, when pressed it will show a list of different languages below. If we implement a grand list of different languages, then we will add the languages onto one panel with arrows next to the panel so the user can scroll through the list of languages,

Once the user is done selecting all the options they wish, they must select the confirm button to keep all their setting options. If they decide they wish to revert to the previous settings, then before they hit “confirm” they can hit “cancel” and the settings will go back to how it was prior to the changes.

6.2.3 Additional Games

Once we are set up with the standard menu and being able to connect games to the device, we want to import as many games as we can. Unfortunately, we do have the design as a constraint to the choices of games. Since the device is a square with sixty-four panels, we are limited in the games we can choose. However this can also be beneficial to us since it can be very good for certain games like chess and checkers since their boards already have 64 blocks. Utilizing this we try to have extra games lined up in the scenario that we have extra time. Below will be some examples of games that we wish to implement if time permits us and a prototype of how they would appear in a top down view.

Paddle Ball
This game is very similar to the hit game Pong, a table tennis arcade game where two players play and attempt to score the ball into the other player's goal. To avoid trademark issues with Atari, we will not copy the original games name and will give it our own unique name. Here two players on opposing sides will be in control of a thin white block that is used to deflect the incoming white ball. Both players will use the leftmost and rightmost panel on their side of the device to move the thin rectangle to parry the moving white ball back to the opposing player. These instructions or instructions similar to these will be included in the game’s menu screen for the players to learn.

Other features may also be added in terms of adding multiple balls for the players to juggle with to make the experience more difficult. Game modes can also be added such as having the game be determined by time or score. For example, the game could keep going until one of the players scores three points or the game could last for three minutes and the player who scores the highest would win. An example of the game is located in the figure below.

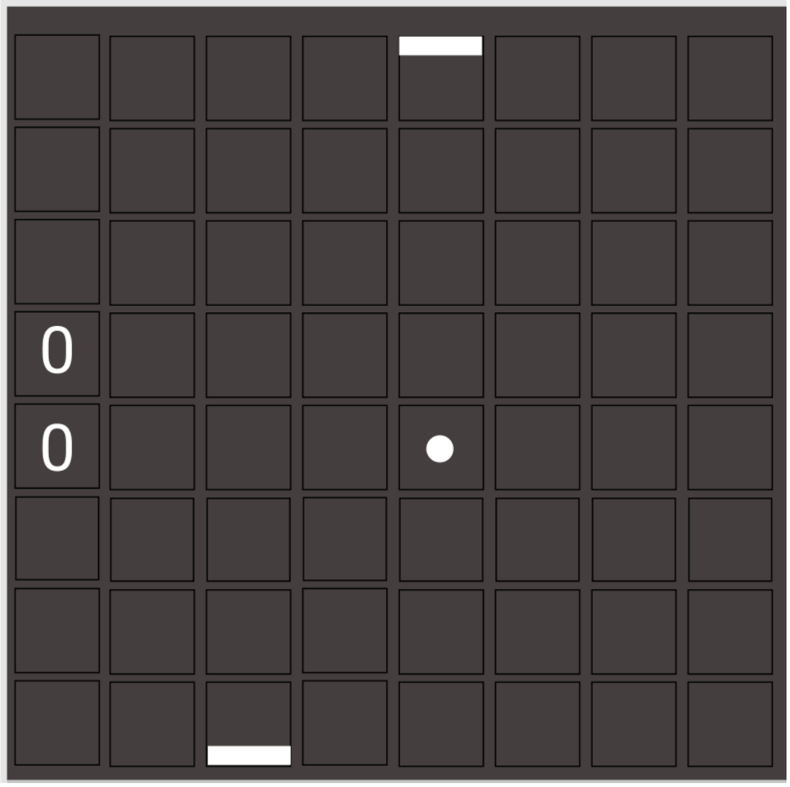


Figure 19 (Paddle Ball): *A prototype of how Paddle Ball would look on the Game Frame.*

Checkers

Checkers is a two player board game that uses the same board as chess. In this game, both players are given twelve pieces and must attempt to capture all of the opponents pieces by having their own piece jump over the opponent's piece diagonally. Both players can only move their pieces diagonally in the direction of the opponent's side and cannot move backwards. However, when one of the player's pieces reaches the furthest row they can, their piece can become a king and is now allowed to move diagonally backwards or forwards. This game is a great addition to the Game Frame as it uses the same amount of spaces as chess so it fits perfectly on our device. An example of the game is shown in the figure below.

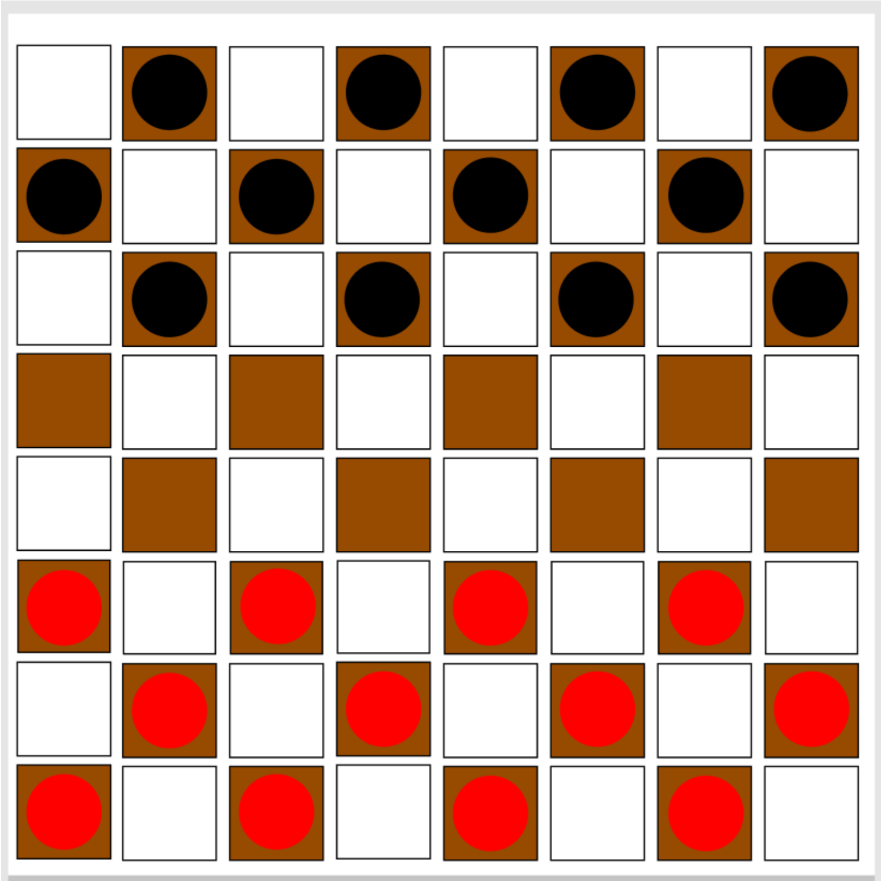


Figure 20 (New Game of Checkers): This is a prototype of how Checkers will start.

This is an example of how the game, Checkers, will be set up once the user has started a new game. Both players will be given twelve pieces and each piece resides on top of a brown tile. Black team will always be the first that is able to move first. Similar to chess, we will be adding in extra features that the user will be explained in the tutorial located in the game's menu. Features will include things such as an undo feature, a pause feature, a feature to return to the main menu. If the user tries to return to the main menu while they are in the middle of the game, it will ask them if they wish to save their progress. These features will be prominent in many of the games we implement.

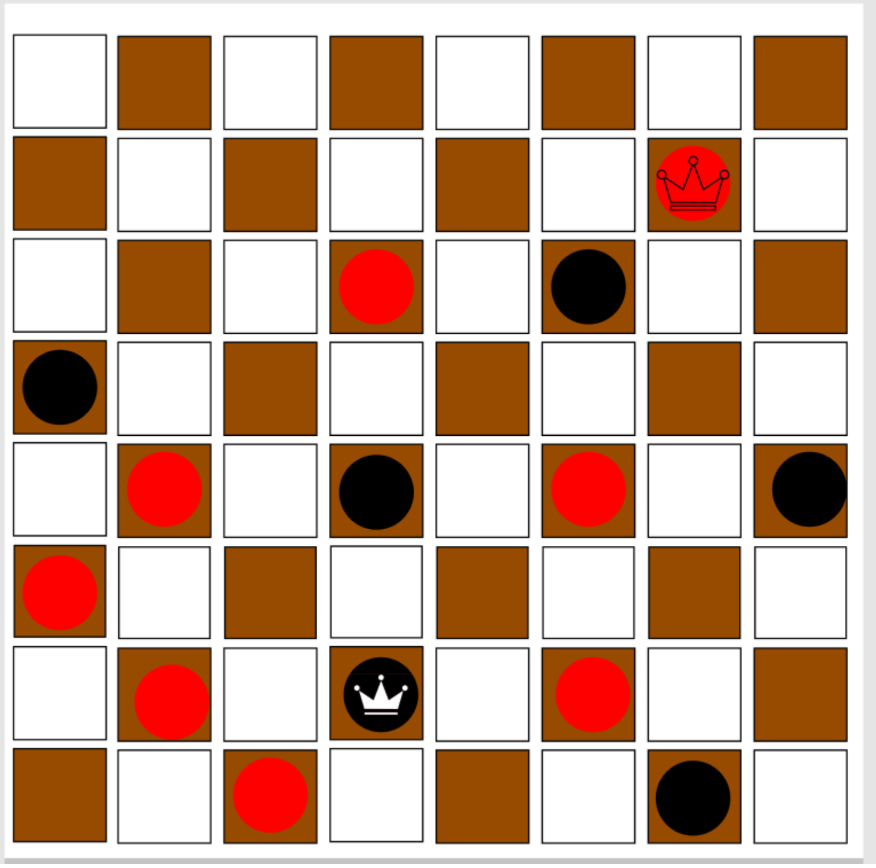


Figure 21 (Checkers Game in Progress): This is an example of a game of Checkers in progress.

The figure above is to show a prototype of a game of Checkers and to show how the king feature will be implemented. It may not be exactly like this and instead of having a white crown on the black piece, it will be an outline of a crown to look a bit more subtle. The reason we couldn't have the same outline on the black piece is because it was not visible so we made it white instead. It will carry a bit more similar features from chess such as selecting a piece and having it's possible moves highlighted as to where the player is allowed to move. Implementing this game will prove to be easier than chess since this game really only has two different pieces in the game, the normal one and the king. Since there is no variety of pieces, the majority of the pieces will all operate the same way and can share the same code. The only condition when it changes, and not by much, is when a piece becomes a king. Even then, it will still be the same code for all of the pieces with an if statement lurking around until the piece reaches the end of the board.

Tic-Tac-Toe

Tic-Tac-Toe is a game where two players alternatively take turns filling out X's or O's in an attempt to get a certain amount consecutively and win. This can be achieved by having symbols appear in a column, row, or diagonally. We decided to give the players a chance to make the game a bit more interesting and customizable to their likings. For example, the player will have the option of choosing the board size whether it be a three by three board, four by four or seven by seven option. Once the user has selected their board size, they also have the option to choose whether they wish to play normally and place X's and O's or they can play a new game mode.

This game mode will make it so when the player wishes to claim a space, they must play another smaller version of tic-tac-toe in order to claim the spot. This game mode would only be eligible for the boards with bigger spaces than three by three and is indicated with every blank space having an empty tic-tac-toe board inside of them. The smaller version will consist of a standard three by three board.

Any panel that is not being used for the game will be fully colored black to indicate that there will not be a use for those buttons and that it is considered out of field. However not all unused spaces will be completely useless as shown in the figure below. Certain panels will be showing the current score between players and will also have buttons that may be beneficial such as an undo feature in case a player accidentally makes a choice. More features will be added if the idea rises during the construction of the game. The figure below shows an example of the game with only one panel showing the empty tic-tac-toe feature just for display.

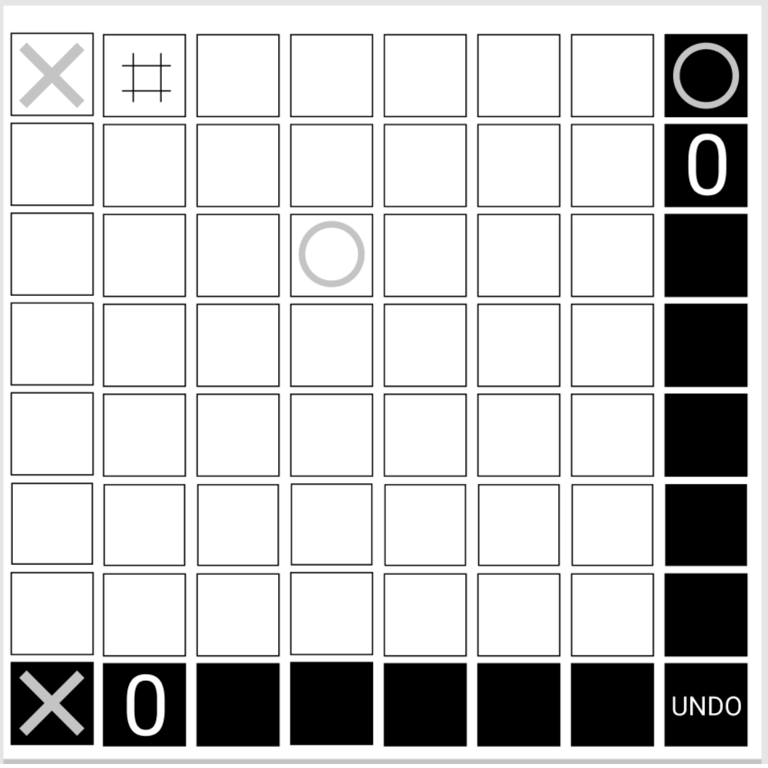


Figure 22 (Tic-Tac-Toe): A prototype of Tic-Tac-Toe being played out.

Rhythm Beat

Rhythm Beat is a rhythm game that we are creating that will utilize the panels of the device for players to press at the appropriate time to sync with the music and achieve a high score. This game is similar to the game, Jubeat, which is what inspired our device the Game Frame. This game will have a selection of songs that the user may choose and play the rhythm game to that specified song. This game is great to implement into our device as it takes advantage of the many buttons on the device to use for pushing. This game can also be played with multiple people. Each player can distribute an area of the board to accurately hit the indicators with the other players. Verbal communication would be key for this to make sure everyone is accurately hitting the panels. If time permits us we may even be able to implement a feature so it can be split and two players can compete against each other.

The panels will all be relatively black to show off the colors more easily to the player. It will either stay like this or we may add some animation of the squares opening up and showing more customized animation depending on how much time we have. The score of the player will also be displayed in the background in a faint color so the player can see it without having it obstruct their view of the game.

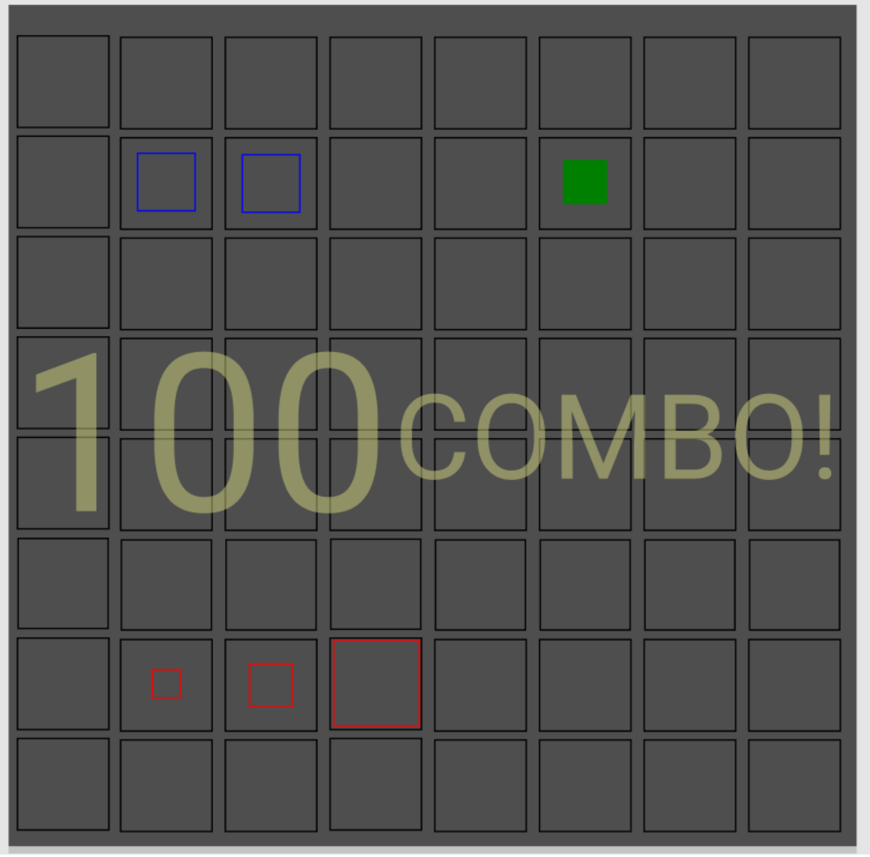


Figure 23 (Rhythm Beat): This is a prototype for the game Rhythm Beat.

The way the game will work is essentially there will be some indicator that shows the player when to press the panel. There will be different types of indicators as shown in the figure above. All the indicators will begin really small and will keep getting closer to the frame of the panel. The player must time it correctly to match the frame with the indicator. The red indicator is a one and down kind of symbol. This means that the symbol will appear and then the user must press it when timed appropriately. These kinds of indicators will be the most prominent ones and scattered across or will sort of lead a trail of red indicators.
The green block will be for pressing the panel and holding it until the green block thins out. This will be to add a bit more features and difficulty into the game so it is not just tapping buttons. Not too many of these will show up in single player mode since it will keep the player occupied.

The other indicator that will be present will be the blue single press square. When one of these squares are visible, players must look around for the second blue square and press both squares simultaneously when lined up with the panel’s frame. The blue square will always appear as a pair to have players have a little scavenger hunt and force them to look for the second blue square to tap in sync with the first blue square. A numbering feature can also be implemented in order to give players a bit more clarity as to which panels they need to prioritize or to give them a heads up as to where they need to go next.

4-in-a-row

This game consists of two players taking turns in placing a ball in the eight columns present. They will take turns trying to get four of their colored balls to be in a row, column, or diagonally. When a user attempts to place a ball with empty spaces below it, the ball will fall down until it lands on top of another ball or the lowest row possible. The software will be continuously checking after the first player has set four balls since it would be impossible to get four in a row prior to this. The software will be checking after each player has completed their turn in order to see if four of the same color ball is in a row, column, or diagonally. Once a new game has begun, the board will be empty and the player who is using the red ball will go first. The order of this will be successible to change, the option will be given in the game’s menu. An example of how the game will be played out is shown in the figure below.

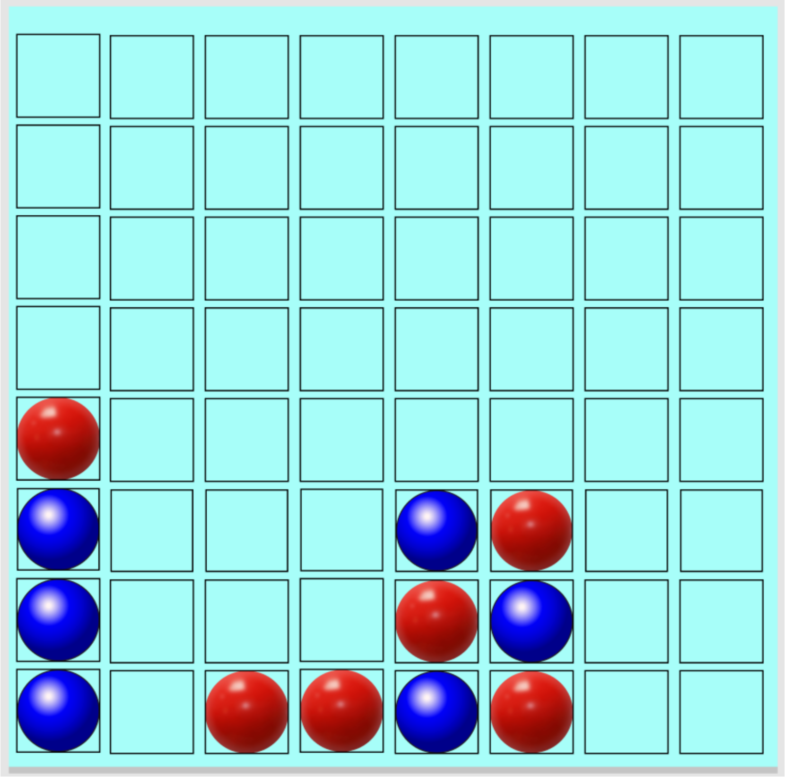


Figure 24 (4-in-a-row): This is a prototype of how 4-in-a-row will be displayed during a game.

Whack-a-Mole

Whack-a-Mole is also another game that takes advantage of the satisfaction of pressing buttons, by whacking moles. In this game the player must “whack” moles by pressing on the corresponding panel where the mole is located. In the game's menu there will be different options available for the player such as a difficulty setting and adding other rules. The difficulty will change the amount of moles that appear and how fast they appear and then disappear. Other rules that the player can add are additional characters that can appear aside from just moles.

For example, moles with little stopwatches can be added. When one of these is whacked, the player will receive an additional time bonus. There is also a cartoon dog that may appear that if whacked, the player will lose points and must try to avoid these. Another character is the mole with the cartoon bomb in his paws. If this character is whacked, the following adjacent squares will also be considered whacked. This character will prove useful if there is a large amount of moles and not enough time to whack them all. However, this character will also be a bit challenging as well since if the mole with the bomb is whacked and is around a dog or multiple dogs, the player will lose points. The current amount of points will also be displayed on the top left screen as shown in the figure below.

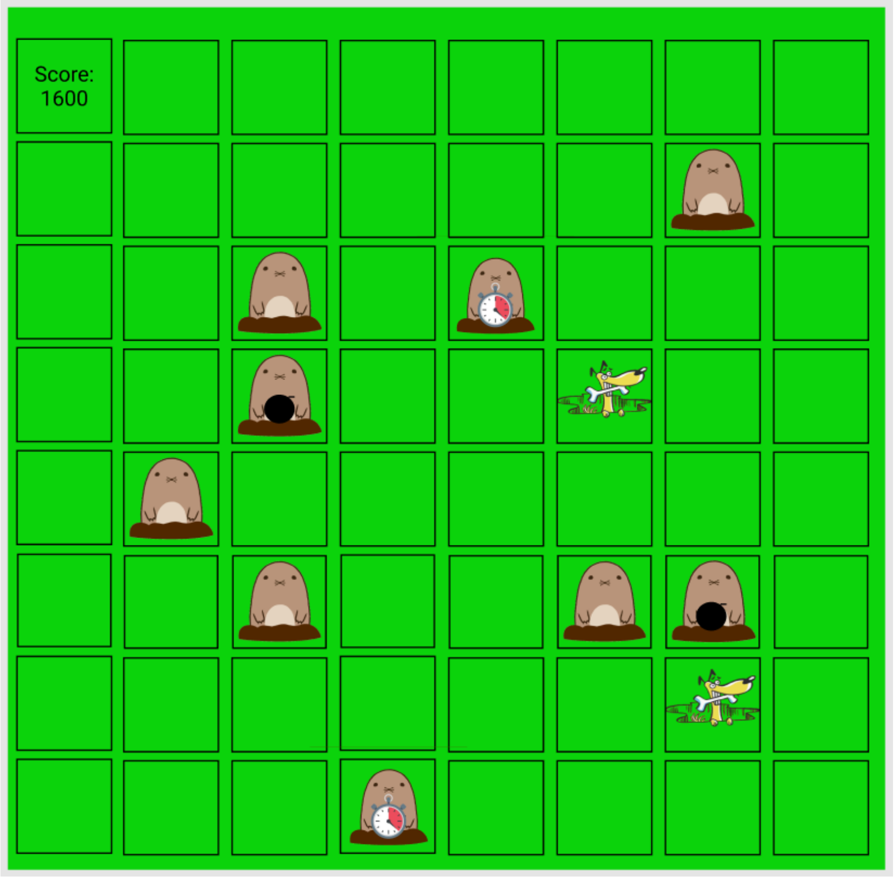


Figure 25 (Whack-a-Mole): This is a prototype of Whack-a-Mole being played.

Alien Invasion

This game is about the player piloting a spaceship that is in space facing endless waves of alien spaceships. The row closest to the player becomes the controls for the game while the rest of the device is used to showcase the game. The top rows will be littered with alien spaceships that keep firing vertically straight down. The aliens' projectiles will be marked in a bright lime green color to show the player where to avoid. If the player is hit with an alien projectile they will lose a life. If the player runs out of lives, they lose the game. The player's projectile will be marked in white to show where the projectile's trajectory is going. The player must use the controls at the bottom to evade the aliens' projectiles and destroy as many aliens as they can to have the highest score they can obtain.

The aliens will have different looks based on different attack patterns. Some aliens will shoot a burst of projectiles, meaning they will fire off three shots in a cone shape, and some will try to fly towards the player. These alien spaceships will have different looks so the player can differentiate them. As shown in the figure provided below, the bottom row will be used for the player's controls. The player will be able to move left and right, fire their own projectiles, and monitor their score and amount of lives.

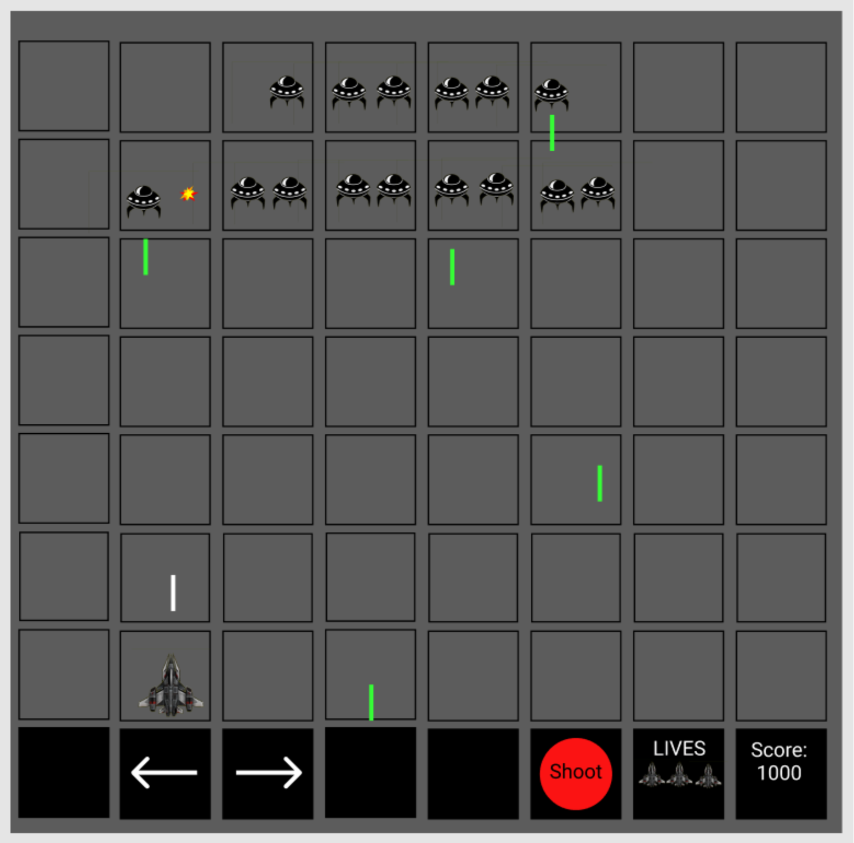


Figure 26 (Alien Invasion): This is a prototype of Alien Invasion being played.

6.3 Hardware

6.3.1 Power

As our device is portable it will probably end up being a DC-circuit. With plans to charge from a wall outlet, we will obviously need a power supply that utilizes AC-DC conversion. For the current prototyping steps, we will focus on using a variable power supply to test our circuit so that we do not have to commit to a battery until we have a better gauge of what is needed. If AC components or AC-DC conversion is in need of testing, we will accordingly use a signal generator.

6.3.2 Estimated Power of Unit

The life span of the battery has been stated as important. Reaching our goal battery-life requires that we understand the amperage for our device in-total. It will be easier to decide upon and look at batteries if we can keep a running estimate of our components' power usage.

The total we calculate as we expand our included components should be the minimum of what our battery is able to output. In the case of current, we need to find the total current first. After we find the total current demand we need to find a battery with two times that amount for its Ah rating in order to last two hours minimum. Likewise, to aim for four hours we need to look for an Ah rating four times as large.

There will be theoretical calculations as we start designing the hardware. These calculations can be tested in the lab once we get components. We will be able to measure with more certainty what our necessary power draw is and what battery might be suitable

A separate hardware controller is determined to be useful and so we have updated our power considerations accordingly.

6.3.3 Estimated Weight of Unit

In a similar fashion to estimating the power, we will be able to reasonably see how our weight stands in relation to our goal if we tabulate the weight. We can do this by listing the parts in the following table as we decide upon the parts or approximate parts used. There is nothing fancy in terms of calculating the total from each part. As this is an estimate, we will overestimate whenever precision can not be perfect. It is safer to overestimate and find that we are under than to underestimate and discover we guessed too low. This is because our weight requirement aims to minimize.

Unit Part	*Estimated Weight*
Jetson Nano	0.55 lbs
MSP430G2553	Less than 0.4375 lbs
Display	~0.399037 lbs
Shift Registers	0.0044092452 lbs
Total (in lbs.)	1.3909462452 lbs

Table 8 (Estimated Unit Weight): A running total of our weight estimation.

The weight of the MSP430 chip used is not provided. The only obtainable weight was that of the LaunchPad kit at 7 oz. The weight is expected to be small, but for the time being it is being referenced as less than 7 oz. Once we have the chip to weigh we can update our table to more accurately reflect its weight.

6.3.4 PCB Prototyping:

Rather than a fully developed and designed PCB, we plan to use a solderless breadboard for connecting various components in the early stages of testing. The first stages require us to have functioning controllers and connections, and a solderless breadboard is the easiest, and least costly way to do this. A solderless breadboard is perfect for prototyping.

Some considerations for prototyping a PCB and its connections can include a conductive pen. This neat modern invention would let us figure the layout and implement it prior to fabbing a PCB. The concern comes with heat production and how realistic it would be to prototype a PCB for our device with this material.

Related implementation involves conductive 3D printer filaments mentioned in our research section. The problem with this and the previous solution is that it is only suitable for low voltage loads. Depending on how low power we can make our device, the filament might work for prototyping. In particular we plan to put off the display part of prototyping in the early stages. Without an LCD or OLED requiring voltage, especially with their backlights, we end up using much less voltage at that point. Obviously this would be inaccurate for power prototyping and implementation, but it could be a cheap alternative early on. As an aside, we plan on utilizing the included microcontroller display outputs when working with the controllers and working through the very early stages of getting things working.

One final option when related to PCB prototyping is to try doing homemade PCB creation. If things are not disposed of properly it can be detrimental to the environment and illegal If proper procedure is followed, it should be safe and could help alleviate costs for a later stage prototype option.

6.4 Testing

6.4.1 Game Engine Testing

Overview

The game engine hosts the actual game being played. It should always make sure that the game has been set up. It will call the check win function on each move. The engine interfaces with the board to show the layout of each piece and should only allow the player and computer to make valid moves. Instructions should be given on when moves are allowed. The game engine should essentially allow for player versus player playing of a game, where eventually the AI will function as the second player in two player mode. The game engine must allow for moves to be made, pieces to move from one square to another, remove old pieces, assign the attributes of each move upon modification of each piece, and initialize the check win function after each player (AI included) makes a move.

Passing tests

The game engine passes the tests when the logic of every implemented game function according to the rules of the game, such as pawns move forward one or two on start and attack diagonally. It also must call the check win function after every move, successfully update pieces and squares after each move, and should be resettable by the setup function when it is called. No permanent changes are made to the board layout. The logic of each game must also hold true, such as bishops moving diagonally, or pieces in 4-in-a-row falling to the bottom. The status of pieces should be updated within the game engine, however is a piece is displayed on the board physically when it is not meant to be, or is not displayed when it should be, this may be an issue with the hardware instead of the engine, so it is not always a failed test for the game engine.

Procedure for testing

Testing the game engine includes testing that every piece makes the correct moves and are not able to move to an illegal place. All squares on the board should be verified as reachable. This will be done by manually placing pieces in each square on the board and verifying that the piece exists and can show up on the graphical interface and eventually physical board. Pieces will be sent the signal to move and a technician will verify that the attributes of the piece have updated as well as verify the tile the piece moved to and from was correct. Many of these tests can be automated, allowing for larger scale testing. The automated tests will be run regularly, printing a debug script of the moves made and status of each affected tile and piece involved in the move. Manual testing will also be performed, as sometimes the tests themselves can be flawed. An entire game will be run on the machine from time to time to verify there are no hidden flaws, but the majority of the time, game testing will be automated and reviewed.

Test Cases

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Chess	Pawns move	Forward one if no piece in front of it Forward two if it is the pawn's first move Diagonal forward one to attack.	Tester initializes a game with the piece in the center, then attempts all possible moves from that point.	Once all moves can be made, the test is successful.
Chess	Knights move	One square one direction + two squares 90 degrees rotated to that (in either order), including its attack.	Tester initializes a game with the piece in the center, then attempts all possible moves from that point.	Once all moves can be made, the test is successful.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Chess	Bishops move	Diagonal any place between itself and an occupied square Diagonal to any piece of the opponent's piece if attacking, provided no piece is between them.	Tester initializes a game with the piece in the center, then attempts all possible moves from that point.	Once all moves can be made, the test is successful.
Chess	Rooks move	Horizontal or vertical any place between itself and an occupied square Horizontal or vertical to any piece of the opposite color if attacking, provided no piece is between them During a castle. (see Castle event)	Tester initializes a game with the piece in the center, then attempts all possible moves from that point.	Once all moves can be made, the test is successful.
Chess	Queens move	Diagonal or horizontal or vertical any place between itself and an occupied square Diagonal or horizontal or vertical to any piece of the opposite color if attacking, provided no piece is between them.	Tester initializes a game with the piece in the center, then attempts all possible moves from that point.	Once all moves can be made, the test is successful.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Chess	Kings move	Diagonal or horizontal or vertical one square, including its attack During a castle. (see Castle event)	Tester initializes a game with the piece in the center, then attempts all possible moves from that point.	Once all moves can be made, the test is successful.
Chess	Pawns do not move	Forward if there is a piece directly in front of it.	Tester initializes a game with the scenario that the piece should not be able to move. This is done for all possible configurations of the piece.	The test is successful when all configurations result in the piece not being able to be moved.
Chess	All pieces do not move	If the move results in its own color going into "check" Off the allotted board Looping around the board To an occupied square of its own color.	Tester initializes a game with the scenario that the piece should not be able to move. This is done for all possible configurations of the piece.	The test is successful when all configurations result in the piece not being able to be moved.
Chess	Piece leaves square	Once the piece is moved to the new square, the piece is no longer displayed on the old square.	Tester initializes a game with a piece on the board. The piece is moved.	The test is successful when the square the piece was previously on no longer displays the piece after the move.
Chess	Piece arrives at new square	Once the piece is moved to the new square, the piece is displayed on the new square.	Tester initializes a game with a piece on the board. The piece is moved.	The test is successful when the square the piece is moved to displays the piece after the move.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Chess	Piece is removed on capture	When a piece is captured by the opponent, the piece should be removed from the board and become unusable for the remainder of the game.	Tester initializes a game where a piece can capture another. The tester then moves the piece to capture the piece.	The test is successful when the piece is removed from the game on capture by the opponent.
Chess	Pawns get knighted	Once a pawn makes it to the other end of the chess board (the opponent's side) the user is prompted as to what piece they want to make the pawn become. The user can choose any piece from the game.	Tester initializes a game where a pawn can be moved to the other side. The tester moves the piece to the knighing position and picks a piece for the pawn to become.	The test is successful when the piece successfully transforms into the piece chosen by the user, and functions as the new piece.
Chess	Castle event is possible	Castling cannot be performed when king or rook have moved Castling cannot be performed to get out of check Castling cannot be performed when pieces are between king and rook Castling results in the king moving two squares right and rook two squares left. (they cross over each other)	Tester initializes a game where a castle event is possible. The tester then castles the pieces.	The test is successful when the castle move results in the pieces moving to the correct squares after the castle event.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Checkers	Pawns move	Pawns move forward diagonally any place between their start position and the next unoccupied square.	Tester initializes a game with the piece in the center, then attempts all possible moves from that point.	Once all moves can be made, the test is successful.
Checkers	Kings move	Kings move diagonally any place between their start position and the next unoccupied square.	Tester initializes a game with the piece in the center, then attempts all possible moves from that point.	Once all moves can be made, the test is successful.
Checkers	Walls are functional	Pieces treat walls functionally as mirrors.	Tester initializes a game where a piece is against a wall. The tester then moves the piece such that it will reflect off the wall.	The test is successful when a piece can be moved in reflection to a wall.
Checkers	Pieces capture the opponent’s piece	Pieces capture the opponent’s piece if there is an opponent piece in a forward diagonal direction and there is an available space located in a forward diagonal direction of the opponent's pawn, then the user's pawn is allowed to jump over it and capture it.	Tester initializes a game where a piece is able to capture a piece from the opposite team. The tester then moves the piece such that it captures the opponent’s piece.	The test is successful when the piece is captured and removed from the game on being captured from the opponent.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Checkers	Pawns become kings	Pawns become kings if a pawn reaches the first row on your opponent's side of the board, it becomes a king. It will have a symbol of a king's crown once it becomes a king.	Tester initializes a game where a pawn can become a king. The tester then moves the pawn such that it reaches the other side to become a king.	The test is successful when the pawn is turned into a king on reaching the opponent's side, and the piece functions as a king.
Checkers	Piece leaves square	Once the piece is moved to the new square, the piece is no longer displayed on the old square.	Tester initializes a game with a piece on the board. The piece is moved.	The test is successful when the square the piece was previously on no longer displays the piece after the move.
Checkers	Piece arrives at new square	Once the piece is moved to the new square, the piece is displayed on the new square.	Tester initializes a game with a piece on the board. The piece is moved.	The test is successful when the square the piece is moved to displays the piece after the move.
Checkers	Piece is removed on capture	When a piece is captured by the opponent, the piece should be removed from the board and become unusable for the remainder of the game.	Tester initializes a game where a piece can capture another. The tester then moves the piece to capture the piece.	The test is successful when the piece is removed from the game on capture by the opponent.
4-in-a-row	Pieces fall	When a piece is placed on the board, the piece will wall all the way to the bottom of the board.	Tester initializes a game where a piece can be placed in a position such that the piece can fall. The tester then places the piece in such a position that the piece falls.	The test is successful when the piece falls to the bottom most available spot, and the location is set to this position.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
4-in-a-row	Pieces do not overlap	When a piece is placed on the board, the piece should fall to sit on top of the highest piece in the row, or the bottom if there are no pieces. Pieces should not occupy the same spot.	Tester initializes a game where a piece can be placed in a position such that the piece can fall on top of a different piece. The tester then places the piece in such a position that the piece falls on top of a different piece.	The test is successful when the piece falls to the bottom most available spot, and the location is set to this position, without overlapping.
4-in-a-row	Pieces cannot be placed above top row	If the column is full, no pieces should be able to be placed in that column.	Tester initializes a game where a column is full. The tester then attempts to place a piece in this column.	The test is successful if the tester is unable to place a piece in this column, and they are still able to place their piece elsewhere, such that they do not lose their turn.
4-in-a-row	Pieces cannot be placed outside game area	Pieces cannot be placed outside the area dedicated to the specific game of 4-in-a-row.	Tester initializes a game in any state and attempts to place a piece in a position outside the play area.	The test is successful if the tester is unable to place a piece outside the play area, and they are still able to place their piece elsewhere, such that they do not lose their turn.
Tic-Tac-Toe	Pieces do not overlap	When a piece is placed on the board, the piece should only be able to be placed in an empty spot. Pieces should not occupy the same spot.	Tester initializes a game where a square is taken. The tester then attempts to place a piece in the taken spot.	The test is successful when the piece is unable to be placed in the occupied spot, and the tester is still able to place their piece elsewhere, such that they do not lose their turn.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Tic-Tac-Toe	Pieces cannot be placed outside game area	Pieces cannot be placed outside the area dedicated to the specific game of Tic-Tac-Toe.	Tester initializes a game in any state and attempts to place a piece in a position outside the play area.	The test is successful if the tester is unable to place a piece outside the play area, and they are still able to place their piece elsewhere, such that they do not lose their turn.
All games	Game does not continue after it ends	Once a game is complete by a win/loss, stalemate, quit, or end condition, the game cannot be continued.	Tester initializes a game that is almost over, then completes the game. After the game is deemed over, the tester attempts to continue the game by moving pieces in the game.	The test is successful if the tester is unable to interact with the game after the game is ended.
All games	Game is playable on start	Once the setup function creates a game, the first player should be able to make their first move.	The tester runs the setup function to create a new game, then attempts to make the first move.	The test is successful if the tester is able to make the first move after the game is set up.
All games	Players only play on their turn	Players should be able to make only one move on their own turn, and unable to make a move when not their turn.	The tester initializes a game and attempts to make a move outside of their own turn.	The test is successful if the tester is unable to make a move outside of their own turn.

Table 9 (Test Cases for Chess): These test cases will help us verify that the software is following all of the rules of chess.

6.4.2 Game Setup Function:

Overview

The setup function is responsible for the creation of new games on startup of a game. The function needs to initialize all pieces in the particular game being played to the default state. To test this, a primitive graphical interface will be created explicitly for the purposes of testing. On initialization of the game, pieces should start at their respective tiles. The graphical interface will then display the location of each piece which will be verified by a technician manually to ensure starting positions are correct.

Passing tests

The graphical display, and eventually board, displays the correct starting condition for the game that is selected to play. The game should also be playable after the setup is complete, meaning that once a move is made the startup function does not reset to the beginning. Games should always be set up to the correct starting position of a given game, and the previous game should not impact the new game. Games should be capable of being quit, reset, lose power, and won by either player including the AI. The correct game should also be set up by the setup function, regardless of what game was played previously.

Procedure for testing

The game setup function will be called with the game type inserted as a parameter. The setup function will execute, then provide a notification that it has completed. The technician will then plug in the intended layout into the graphical interface created for testing to verify that the starting layout has been configured correctly. Numerous different starting conditions will be tested to ensure there are no errors resulting in the startup function requiring the previous game to start in any specific orientation.

Test cases

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
All games	Computer won	The game before the one being set up by the game setup function was won by the computer against a player.	Tester initializes a game where the computer will win. Once the computer wins, the game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the setup function is called.
All games	Player won	The game before the one being set up by the game setup function was won by the player against the computer.	Tester initializes a game where the player playing against the computer will win. Once the player wins, the game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the setup function is called.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
All games	Player 1 won	The game before the one being set up by the game setup function was won by the user at the "Player 1" position.	Tester initializes a game where player 1 will win. Once player 1 wins, the game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the setup function is called.
All games	Player 2 won	The game before the one being set up by the game setup function was won by the user at the "Player 2" position.	Tester initializes a game where player 2 will win. Once player 2 wins, the game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the setup function is called.
All games	Draw	The game before the one being set up by the game setup function ended in a draw.	Tester initializes a game where the game will end in a draw. Once the game ends, the game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the setup function is called.
All games	Stalemate	The game before the one being set up by the game setup function ended in a stalemate.	Tester initializes a game where the game will end in a stalemate. Once the game ends, the game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the setup function is called.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
All games	Quit	The game before the one being set up by the game setup function was quit.	Tester initializes a game then quits. Once a new game is created, the game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the setup function is called.
All games	Reset	The game before the one being set up by the game setup function was reset during the middle of the game, but not quit.	Tester initializes a game then selects reset. The game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the setup function is called.
All games	Power loss	The game before the one being set up by the game setup function was interrupted by a power loss.	Tester initializes a game then disconnects power to the device. The game setup function is called to reset a new game, and the tester verifies the status of the game that is set up.	The test is successful if the game is set up after the function is called automatically.
All games	Repeated calls	The game reset function is called multiple times consecutively without modifying the game between calls.	The game setup function is called to set up a new game. Immediately, the reset function is hit, then again immediately. The tester verifies that the final game is set up.	The test is successful if the game is set up after the function is called.
All games	Correct setup	The game set up by the setup function is set up in the correct zero state (the first move has not yet been made) of the corresponding game.	The tester calls the setup function and verifies that the game is set up correctly, and in a playable state.	The test is successful if the game is set up correctly after the function is called.

Game	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
All games	Change game type	The game previously played was different to the game currently being set up by the setup function.	Tester initializes a game, then uses the setup function to set up a different game. The tester verifies the status of the game that is set up.	The test is successful if the game is set up correctly after having a different game played previously.

Table 10 (Test Cases for Setup): These test cases are to verify that a game sets up correctly.

6.4.3 Game Solver Testing:

Overview

The game solver is the "AI" that will be used to play against the user in single-player mode. This AI will need to have 3 (or more) difficulty levels available for the player to play against, and be able to win more games against the same user as the difficulty is increased. The AI needs to be able to function as the second user in a game, interacting with the game engine's input output system. The AI needs to make only valid moves, and should have a failsafe for if somehow it accidentally makes an incorrect move to which the game engine tells it is illegal, so the AI will make a new, legal move. The AI should be able to make its moves within less than one second of computational time

Passing tests

One second is the maximum allowable time for finding a new move and executing it, however longer times may be allowed for if an "extreme" difficulty level is implemented. If played with a user at the same skill level, the AI should demonstrate a noticeable increase in wins as the difficulty level increases. If a defined solving algorithm is used, multiple games should be sent to the AI to play against and it should win only a few on low difficulty. More should be won on medium. Most should be won on hard. The AI must be able to make legal moves and can move again if the gaming engine tells it that its move was invalid.

Procedure for testing

To test the AI's ability to move, automated tests will be created that feed the AI a given gameboard, then the AI will be told to make its move. A technician will check the graphical interface or board to verify the move was performed legally and within the time constraint. Technicians will also play games against the AI to ensure the AI "feels right" and to see if any patterns emerge that may give the user a clear upperhand. The AI will also play against itself on different difficulty levels and the win loss statistics will be monitored to ensure the higher difficulty AI wins more than the lower level ones.

Test cases

Plays against	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
All players	Easy win	The computer has a game where it can win in just one move.	Tester initializes a game where the game can be won by making one move. This is done for the easy, medium, and hard difficulty settings. It is expected that all settings should be able to win in this scenario.	All difficulty settings make the moves that win the game.
All players	Medium win	The computer has a game where it can win in one third of the total number of average moves for the given game.	Tester initializes a game where the game can be won by making one third of the total number of average moves for the given game moves. This is done for the easy, medium, and hard difficulty settings. It is expected that the medium and hard settings should be able to win in this scenario.	Medium and hard difficulty settings make the moves that win the game.
All players	Hard win	The game is not in a position that has a well defined end path to it.	Tester initializes a game where the game does not have a known path to win. This is done for the easy, medium, and hard difficulty settings. It is expected that the hard setting should be able to win	Hard difficulty setting makes the moves that win the game.

Plays against	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
All players	Start board layout	The game is initialized to the status of the start of a normal game.	Tester initializes a game to the default start of the game. The tester then analyzes the computer's moves to ensure it is playing the game as expected.	The computer makes moves on its turn, and plays through to the end of the game.
All players	Given board layout	The game is initialized to the status of a game that has already played out such that the AI does not see the start of the game.	Tester initializes a game to a given random position. The tester then analyzes the computer's moves to ensure it is playing the game as expected.	The computer makes moves on its turn, and plays through to the end of the game.
All players	Blank board layout	The game is completely void of pieces (or filled in entirely)	Tester initializes a game where there are no pieces on the board. The tester analyzes the computer's response to ensure it does not generate bugs. *This test is a failsafe test.*	The computer makes no action when no moves are possible, and does not attempt to generate false moves.
All players	No moves possible	There are no possible moves for the computer to make, from lack of pieces, lack of open spaces, or no moves being legal to make.	Tester initializes a game where the computer is unable to make any moves. The tester analyzes the computer's response to ensure it does not generate bugs. *This test is a failsafe test.*	The computer makes no action when no moves are possible, and does not attempt to generate false moves.

Plays against	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
All players	No win possible	There is no way for the computer to win the given game.	Tester initializes a game where there is no possible way for the computer to win the game. The tester then analyzes the computer's response to ensure it plays out to the end of the game anyway.	The computer plays the game through to the end, even when it cannot win.
All players	1 second to think	The computer takes no longer than one second to make a decision to move in any state of the game.	Tester initializes a game where the computer needs to make a move and records the amount of time it takes for it to respond and make the move. This is repeated in many scenarios.	The computer does not take longer than 1 second to make its move from the end of the player's move to the end of it's move for any recorded test.
Other computer	Easy v. medium	Two computers play against each other, one being easy, the other medium difficulty setting.	Two different instances of the computer are set up to play against each other, one set to the easy difficulty level, the other set to medium. The tester records the moves, and analyzes the win results of numerous matches played in this way.	The medium difficulty level wins against the easy difficulty level 75% of the time or more.

Plays against	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Other computer	Easy v. hard	Two computers play against each other, one being easy, the other hard difficulty setting.	Two different instances of the computer are set up to play against each other, one set to the easy difficulty level, the other set to hard. The tester records the moves, and analyzes the win results of numerous matches played in this way.	The hard difficulty level wins against the easy difficulty level 95% of the time or more.
Other computer	Medium v. hard	Two computers play against each other, one being medium, the other hard difficulty setting.	Two different instances of the computer are set up to play against each other, one set to the medium difficulty level, the other set to hard. The tester records the moves, and analyzes the win results of numerous matches played in this way.	The hard difficulty level wins against the medium difficulty level 75% of the time or more.
Player	Easy difficulty	The computer plays against a player at the beginner, intermediate, and expert skill levels on the easy difficulty setting.	Human players who are self described as beginner, intermediate, and expert skill levels play a few games against the computer on its easy difficulty level. The tester records the number of wins/losses for each player at this level.	+/-20% the player should win against the computer at a rate of Beginner: 50% Intermediate:75% Expert: 95%

Plays against	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Player	Medium difficulty	The computer plays against a player at the beginner, intermediate, and expert skill levels on the medium difficulty setting.	Human players who are self described as beginner, intermediate, and expert skill levels play a few games against the computer on its medium difficulty level. The tester records the number of wins/losses for each player at this level.	+/-20% the player should win against the computer at a rate of Beginner: 20% Intermediate:50% Expert: 75%
Player	Hard difficulty	The computer plays against a player at the beginner, intermediate, and expert skill levels on the hard difficulty setting.	Human players who are self described as beginner, intermediate, and expert skill levels play a few games against the computer on its hard difficulty level. The tester records the number of wins/losses for each player at this level.	+/-20% the player should win against the computer at a rate of Beginner: 5% Intermediate:25% Expert: 50%

Table 11 (Test Cases for Solver): These test cases are to verify that the AI is working and works at various difficulties.

6.4.4 Input/Output Testing

Overview

The inputs and outputs should be responsible for correctly transferring data between hardware and software. The hardware should read any input signals properly. In return, the controller should be able to receive the corresponding result from the software and display the desired output in the proper part of the screen. Both the software and hardware should interface correctly so that interaction with the device is seamless. The majority of the communication between the user and the Game Frame will be through the use of buttons. If time permits it, we can also add in a microphone so that the device can also take in voice commands from the user.

Passing tests

The microcontroller should be able to detect when a button is pressed, and specifically which button is pressed. All 64 buttons should be able to send the right signal to the processor and the processor should identify them all separately. Since multiple buttons will be connected to a single pin, the microcontroller should be able to differentiate all the different buttons serviced by the same pin. This should indicate that the button matrix is working properly.

Procedure for testing

Press each button separately and see if they are all registered by the embedded software. The screen should also respond properly to each button being pressed individually.

Test cases

Input/ output	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Input	Processor receives input	All of the 64 input buttons are able to send a signal to the processor.	Each button will be tested individually by pressing one button at a time and seeing if the controller correctly reads the 1 or 0 when pressed.	Each button correctly sends a signal to the controller that it has completed a circuit and displays the input. For testing, it is likely that the button will be coded to turn on an LED when pressed and off when not pressed for each individual button.
Input	Unique identifiers	Each button has a unique number to identify.	Each button should all be individually recognized by the controller. The controller should know exactly where in the switch array a button is pressed once a signal is detected. Each button will be coded to send a unique identifier back to the computer during the test. For simplicity, each button will be labeled in the code 1 to 64, and when a button is pressed, the correct number should be printed into the screen. It needs to also determine if multiple buttons were pressed at the same time and know which ones.	To pass this test, each number must be individually recognized and also be able to read them in any order they are pressed. It needs to also be able to detect when multiple buttons are pressed at the same time. When multiple buttons are pressed at the same time, the numbers those buttons were assigned to should display at the same time together.

Input/ output	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Output	Correct output screens	Screens display the correct square of information.	Divide up the pixels on the LCD evenly and assign each button a part of the LCD. Pressing a button should turn on it's assigned pixels and nothing else. This should work with simultaneous buttons pressed	To pass this test, each button must only turn on the pixels it has been assigned. Every other pixel where a button is not pressed should be turned off.
Output	Sound	Audio comes out of speakers properly	Code a sound to be played when any button is pressed. Assign this sound to any button and press the button to see if the intended sound is played from the speakers.	Pressing the button the sound was assigned to should be immediately followed by the sound coming out of the system.

Table 12 (Test Cases for I/O): Procedures to verify the functionality of I/O.

6.4.5 Game End/Won Testing

Overview
After each move by either the player or the computer, the game end function should be run. Win and lose scenarios, stalemates, draws, quits, cases of over moving, and running out of time are all things that could make the game end. The end game function should check for each of them. Once a game is determined to be over, the end game function will send a signal to display the end game screen and the game engine will be sent a signal to stop allowing players to make moves. The way the game has ended must also be sent to the end game screen, but the score and other details about the game that need to be displayed will be handled by the game engine. The checker should only test the recently affected pieces in order to have a fast runtime and should never take more than a tenth of a second to run. The game should not be playable until after the checker has completed checking the status.

Passing tests
The end game checker should be able to indicate that the game is over for all possible scenarios and test for a win or stalemate in any situation that the game can be configured for. Even if it is not possible to arrive in the state, like having 10 queens on the board in chess, it should be able to be checked. The checker should be able to indicate whether player 1 or player 2 has won or lost, but the end game screen will be responsible for deciphering if either player is the AI. The checker also keeps a timer running for timed games and must end the game by sending an "out of time" signal to the end game screen for timed games. Upon being told by the input panel that the players choose to quit, the end game checker should treat it similarly to a win, loss, or stalemate by sending the end screen a signal that the game was quit.

Procedure for testing

Since the game engine will have many tests of handling end game scenarios, the end game checker will also be tested a lot with the game engine. The checker will only work when the game engine is working, but they can both be refined at the same time. Games will be set up manually in winning and losing cases, and the checker will be run. Only when the checker has 100% accuracy in determining the win status of a game will it be determined complete. There is no margin of error. Games may also be played out and automated testing can be run in player-player mode. Large quantities of games can be run and can have output logs sent to be analyzed. Games will also be played out for a few moves then quit to verify this function works as well.

Test cases

End type	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Win	Win player 1	Player 1 makes the move that wins the game.	Tester initializes a game where player 1 is about to win. The tester then plays through such that player 1 wins. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Win	Win player 2	Player 2 makes the move that wins the game.	Tester initializes a game where player 2 is about to win. The tester then plays through such that player 2 wins. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Lose	Lose player 1	Player 1 makes the move that loses the game.	Tester initializes a game where player 1 is about to lose. The tester then plays through such that player 1 loses. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.

End type	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Lose	Lose player 2	Player 2 makes the move that loses the game.	Tester initializes a game where player 2 is about to lose. The tester then plays through such that player 2 loses. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Win	Win computer	The computer makes the move that wins the game.	Tester initializes a game where the computer is about to win. The tester then plays through such that the computer wins. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Lose	Lose computer	The computer makes the move that loses the game.	Tester initializes a game where the computer is about to lose. The tester then plays through such that the computer loses. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Stalemate	Stalemate player v. computer	A player v. computer game ends in stalemate.	Tester initializes a game against the computer which is about to end in a stalemate. The tester then plays through such that the game ends in a stalemate. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.

End type	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Stalemate	Stalemate player v. player	A player v. player game ends in stalemate.	Tester initializes a game against a player which is about to end in a stalemate. The tester then plays through such that the game ends in a stalemate. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Quit	Quit player v. computer	A player v. computer game is quit.	Tester initializes a game against the computer which is then quit by the tester. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Quit	Quit player v. player	A player v. player game is quit.	Tester initializes a game against a player which is then quit by the tester. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Out of time	Out of time player v. computer	A player v. computer game ends because the time limit for the game is hit.	Tester initializes a game against the computer which is about to run out of time. The tester then plays through such that the game runs out of time. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.

End type	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Out of time	Out of time player v. player	A player v. player game ends because the time limit for the game is hit.	Tester initializes a game against a player which is about to run out of time. The tester then plays through such that the game runs out of time. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Over move count player v. computer	Over move count player v. computer	A player v. computer game ends because the move limit for the game is hit.	Tester initializes a game against the computer which is about to run over the maximum amount of moves. The tester then plays through such that the game runs over the maximum number of moves. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.
Over move count player v. player	Over move count player v. player	A player v. player game ends because the move limit for the game is hit.	Tester initializes a game against a player which is about to run over the maximum amount of moves. The tester then plays through such that the game runs over the maximum number of moves. A new game is then initialized, and the tester records the results.	A new game is set up at the correct starting state after the setup function is called, and the game is functional.

Table 13 (Test Cases for Game End/Won): Procedure to verify if the game is over.

6.4.6 Power testing

1. *Power output (voltage, current)*

2. *Battery life*
3. *Overheating*
4. *Recovery after power loss*
5. *Power noise (consistent input)*

Overview

Power for the device will be supplied via battery and fully standalone. The battery will be a rechargeable lithium-ion battery, similar to one used for a laptop or smartphone. This battery pack should be capable of providing consistent DC voltage to the components of the system which require it, should remain cool, last 3 hours or more, and have low noise output for the battery, and power system. The power system should also be capable of recovering after power loss, and have fail safes in place to ensure the system software is power-loss safe.

Passing tests

The system must provide consistent voltage and current within the required range at each point along the tested areas. The system should also be capable of fully recovering to a state where the player can play a new game as normal after power loss.

Procedure for testing

The power system will be measured in its outputs to see what kind of power is output. Oscilloscopes and voltmeters will be used to determine the voltage, current, and stability of the battery output, and the voltages and currents at different points along the internals of the system. To test the recoverability after power loss, the system will have its main power source disconnected, as well as have individual power carrying components disconnected to verify the system's ability to recover from these specific cases.

6.4.7 Error testing

1. *Power loss*
2. *Electric shock/static shock*
3. *Recover from software errors*
4. *User errors*
5. *Hardware errors*

Overview

During the course of operating any electronic device, errors are expected to occur. The GameFrame must be capable of running into a variety of common errors and recover from them with seamless operation. Possible errors for this device include power loss, bad or incorrect input to the squares and functional buttons, static shock from normal handling of the device, software errors, user error, and hardware errors.

Passing tests

The game board should recover to the startup state after all potential errors.

Procedure for testing

For each test, an error will be created manually, or via automated testing for the case of software testing. The failsafe of the system then should kick in and correct the error. The tester then records the recovery state.

Test cases

End type	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Power	Power loss	Power is suddenly gone	While running the device, turn off the power or remove the battery	The device reboots without a problem
Shock	Static discharge	Static discharge causing electrical shocks	Introduce an outside source of electricity such as an extra voltage	The device reboots without a problem
Software	Segmentation fault	The game goes out of bounds in memory	Code the AI to input a move that is out of bounds from the grid of the buttons	The system returns with a segmentation fault error.
User	Inputs an invalid move	User tries to use make a move in the game that is invalid	Play a game of chess and try to move a pawn backwards	The system should return with a message saying the move is invalid

Table 14 (Test Cases for Error testing): Procedure to verify error handling capabilities

6.4.8 Performance testing

1. *Battery life*
2. *Power-up*
3. *Hardware performance*
4. *Drop-safe*
5. *Shock-safe*
6. *Signal noise*

Overview

The GameFrame has numerous different components to it that are expected to act in a certain way and within a certain range of operational values. Many of the systems interact with each other, and one failing system may lead to another reducing in performance. Such systems as battery life, power-up ability, hardware performance, droppability, shock resistance, and signal noise are components of the system that need to be regulated to a specific level of performance.

Passing tests

Passing tests should result in our measurements matching the goals we set out in our requirements

Procedure for testing

For each test, put the device under a worse possible use scenario.

Test cases

Type	*Test Case*	*Details*	*Testing Procedure*	*Passing Test*
Battery	Battery Life	How long the Battery will last	Keep the device powered on until the device runs low on battery and record the time	The device stays on past or turns off past the 2 hour mark
Power-Up	Boot time	How long it takes for the device to power up and ready to use	Turn on the device and record how long it takes to go to the main menu screen.	Reaches the menu screen within 30 seconds.
Hardware	?			
Durability	Drop-safe	Drop the enclosure without the actual electrical components	To ensure that the enclosure will not break, pre-assemble it and drop it while standing up to test durability	The enclosure does not come apart
	Shock-safe			
Noise	Signal noise			

Tables 15 (Test Cases for Power Testing and Performance Testing): Procedure to verify performance

6.5 Evaluation Plan

6.5.1 Evaluation of Hard Requirements

Requirement Type	*Name*	*Test*	*Results*
Power	Battery Life	To test the battery life, games against AI should be simulated until the device turns off or stays on past the battery life goal.	Satisfactory: >2 hours Unsatisfactory: < 2 hours
Physical	Dimensions	While this should not require any extensive test besides measuring, the dimensions should be met as long as it was built around this from the beginning. The only possibility that can cause unforeseen changes to the dimensions is if a different component is physically demanding it.	1-foot width or length maximum
	Weight	Using any weight measuring device	Satisfactory: ≤4-lbs Unsatisfactory: > 4-lbs
Monetary	Unit Cost	Evaluate the cost of each material used in a single build and add them all together	Satisfactory: ≤$400 Unsatisfactory: > $400
Software	Chess	Play through the one game either against AI or against another player without interruption	Satisfactory: No glitches Unsatisfactory: Any bugs or crashes

Table 16 (Evaluation Plan): A set of tests to evaluate if our hard requirements are met

6.6 Facilities and Equipment

6.6.1 Texas Instruments Innovation Lab

- Laser Cutter: May potentially be used for cutting acrylic.
- 3D Printer: May potentially be used if we decide to 3D print the housing for our build.
- Soldering stations: For soldering pins
- Senior Design Lab equipment: For testing and implementing prototypes

6.6.2 Off Campus

As a result of COVID, it is likely that most of our work will be done from home using communication platforms such as Discord to coordinate any work necessary. All of us have the necessary computers to do most of our work from home. However, it is possible for meet ups if needed.

6.6.3 Linux

Most of our coding for the hardware and software will likely be done using Linux or a Linux distribution such as Ubuntu. While there are IDE's and other development tools provided for many of the micro controllers we have considered, they also recommend developing using a terminal for interacting with the hardware.

7. Operation Guide

7.1 Start up

When a user operates the Game Frame for the first time, they must activate the power switch in order to turn on the device. Once the power is on the user is greeted by the main menu which is shown in the software prototype in figure 5 main menu. Here the user can select the panels on the screen to navigate between the options and the games that will be located on the lower half of the screen. Each game will be located on it's own corresponding panel and the options will also have it's own panel. If we add in a plethora of games, then the user will be able to use the panels with arrows on them to cycle through the different games available on the device. Once the user has traversed the list of games, the user selects which game they are interested in playing and the main menu will lead them to that game's menu.

7.2 Menu

Once the game's menu has loaded, the user will be presented with settings for when they intend to play the game they have chosen. If the user is unfamiliar with how to play the game, there will be a panel labeled tutorial that will elaborate on how to play the game and any unique bindings that there are.

7.2.1 Game Select

Here, on the game menu, the user will specify whether they will be playing with a computer or with another person. If the user chooses a computer to play with, then the menu will ask them what difficulty to select for the computer. If the user selects to play with another person then it will simply load in a new game. Once the game begins, the player will continuously play until someone results in a victory or if there is a draw.

7.2.2 Gameplay

While the game is being played, the software will be continuously checking to see if the state of the game results in a victory or draw. As players select and confirm where they wish to move their pieces, the software will be double checking to see if that move is allowed or not. The software will see if the selected piece is allowed to move and if it can it will either inform the user by placing a colored square where acceptable or by allowing them to move.

7.2.3 Save

While the game is being carried out, the user has the ability to return to the game's menu. Once this is done, the software will ask if they wish to save the current game or else the game will be lost. If the user saves the current state of the game, the software will create a save file that stores the information of the game. When the user returns to the game's menu, they will have the option to load the previously saved file and they can resume the game whenever they desire.

7.3 End Game

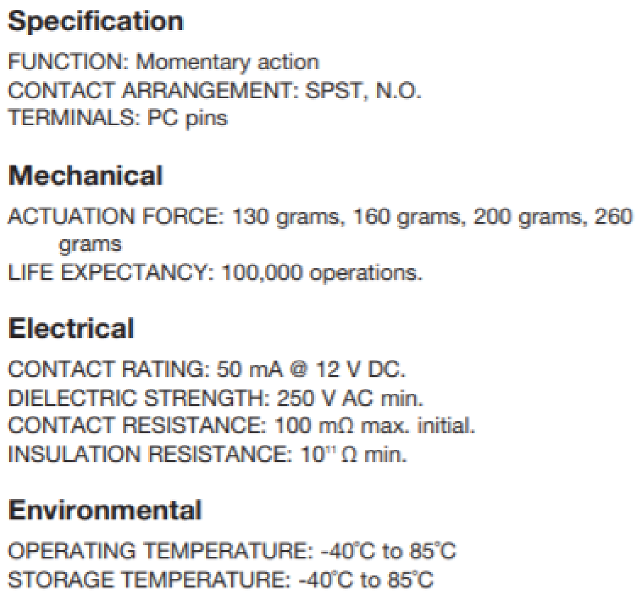
Once the game inevitably reaches a victory or a draw, the user will be presented by a results screen that shows the result of the game and will ask the user if they wish to reset and play the game again with the same settings or if they wish to return to the game's menu.

8. Research and Tradeoffs

8.1 Switches

In order for the buttons to even operate we need switches. There are several options we have looked into thus far. At the moment, the plan is to utilize four switches for each of the 64 buttons - one for each corner. We might be able to implement the device with fewer as the project and its design unfolds.

8.1.1 Tactile Switches



Specification

FUNCTION: Momentary action
CONTACT ARRANGEMENT: SPST, N.O.
TERMINALS: PC pins

Mechanical

ACTUATION FORCE: 130 grams, 160 grams, 200 grams, 260 grams
LIFE EXPECTANCY: 100,000 operations.

Electrical

CONTACT RATING: 50 mA @ 12 V DC.
DIELECTRIC STRENGTH: 250 V AC min.
CONTACT RESISTANCE: 100 mΩ max. initial.
INSULATION RESISTANCE: 10^{11} Ω min.

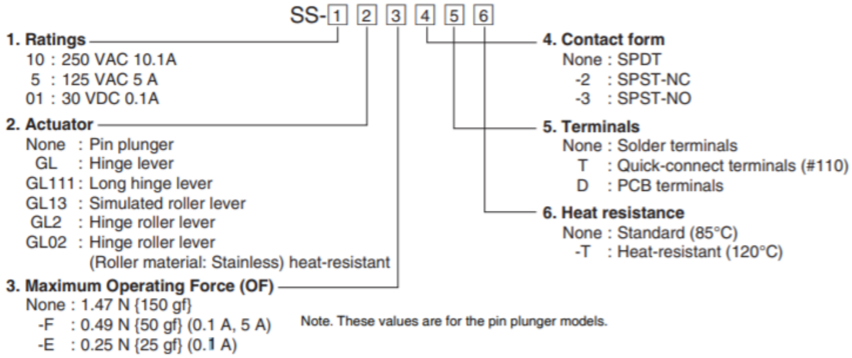
Environmental

OPERATING TEMPERATURE: -40°C to 85°C
STORAGE TEMPERATURE: -40°C to 85°C

Figure 27 (Tactile Switch Sample Specs): A list of specifications for an example tactile switch. Taken from the PTS645 datasheet.

These small switches are around 18-25¢ and are cheap in cost compared to other options making them good for prototyping. Because they are the cheapest, they might also lead to a device with the least responsive feedback and lead to being the cheapest feel amongst the choices. These have a lower voltage rating and minimum current than the next option that we'll go over, making them better for a portable device. The datasheet graciously provides an expected durability of 100,000 uses so that we know its lifespan and can further estimate potential repair costs down the road. Though repair costs are not an important decision-making factor for our project, when referring to the cheap cost of these switches, it would also have a positive benefit to lowering repair costs.

8.1.2 Snap Action Switches



SS-1 2 3 4 5 6

1. Ratings
10 : 250 VAC 10.1A
5 : 125 VAC 5 A
01 : 30 VDC 0.1A

2. Actuator
None : Pin plunger
GL : Hinge lever
GL111 : Long hinge lever
GL13 : Simulated roller lever
GL2 : Hinge roller lever
GL02 : Hinge roller lever
(Roller material: Stainless) heat-resistant

3. Maximum Operating Force (OF)
None : 1.47 N {150 gf}
-F : 0.49 N {50 gf} (0.1 A, 5 A)
-E : 0.25 N {25 gf} (0.1 A)

Note. These values are for the pin plunger models.

4. Contact form
None : SPDT
-2 : SPST-NC
-3 : SPST-NO

5. Terminals
None : Solder terminals
T : Quick-connect terminals (#110)
D : PCB terminals

6. Heat resistance
None : Standard (85°C)
-T : Heat-resistant (120°C)

Figure 28 (Snap Action Sample Specs): A list of specifications for an example snap action switch. Taken from the SS-x series datasheet.

Snap action switches are opposite of the previous option in many ways. It is by far more expensive, but has potential for the best tactile response and feel when used concerning purchasable switch components. The DIY video of someone making a jubeat machine used these switches. A huge problem is that it requires much more voltage for operation. The amperage is still manageable, however it is higher. Higher amperage demand would mean that the battery could not last as long. The datasheet has information pertaining to running the device on lower loads. The tradeoff might help the battery life positively while negatively impacting the durability of the switches.

8.1.3 PCB Trace Switches

One of the best options is probably implementing electrical contact buttons with PCB trace membranes for the switch. Jubeat uses this method of implementation. PCB prototyping is the most expensive prototyping option leading to an obvious downside. For unit cost this would probably be the cheapest to produce since it is part of the PCB cost.

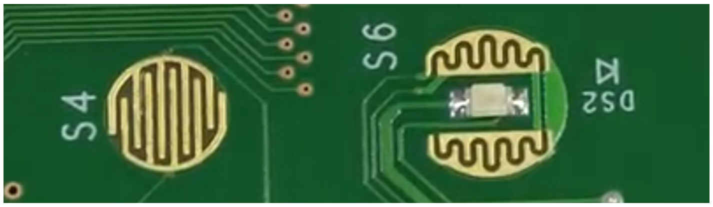


Figure 29 (Folded PCB Traces): A picture provided to visually describe what is being talked about. Origin of image unknown and used under fair use. The picture was acquired from StackExchange.

8.1.4 Arcade buttons

It has been discussed that auxiliary buttons on the side might be useful in our design. They could operate not only as UI or power control, but could also be used by various games down the road. They might not be necessary and might be an additional feature. It details an expected 10,000,000 uses on the kind we looked at.

**It should be noted that the switches linked in the budget section do not contain bulk costs. Since we would have a potential 256 switches, bulk costs are actually a viable consideration. One such example of bulk cost is provided in the same table.*

8.2 Controllers

There are a large variety of controllers to choose from. The plan for now uses two. A microcontroller will work for managing the hardware and the array of switches while also interfacing with another controller or computer which manages the software. Another viability is to use a microcontroller or other controller with two processors on it. The benefit of the separate controllers is the ability to facilitate simultaneous progress and work on the hardware and software side of things.

8.2.1 Raspberry Pi

Raspberry Pi of course is known for their variety of products and options to use as controllers. The obvious downside of one of these is that they are not as impressive for inclusion on a resume compared to other controllers.

Raspi 4
A Raspberry Pi 4 is monetarily more expensive than other controllers that might have less powerful specs in them. The Raspberry Pi 4, however, has specs powerful enough to implement Python as a default language. The power is not free and might affect our desired battery life. For the time being, it seems the safest choice for handling the software until we know more.

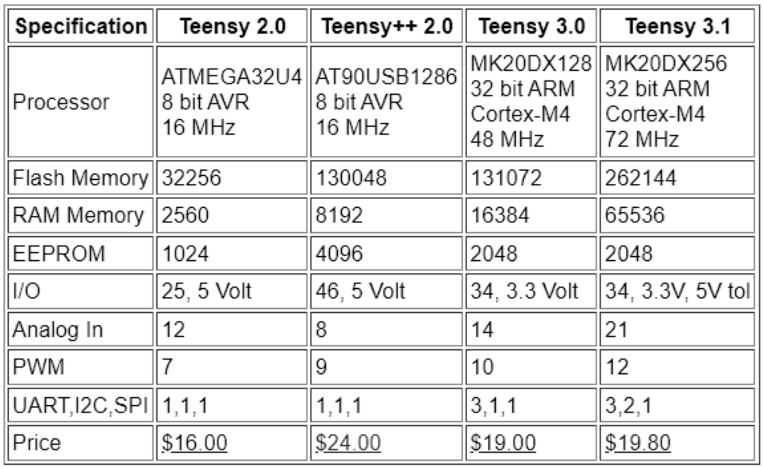
Raspi Pico
The Pico is an appealing incredibly cost efficient option at only $4. The size and lower specs make it desirable as a microcontroller, but if it is sufficient to handle our software, it seems the superior choice.

RP 2040
It should be noted that the microcontroller chip can be bought by itself and designed around. This is an even cheaper option and allows our computer engineers the potential to develop their architectural skills.

8.2.2 Teensy

An honorable mention is the Teensy++ 2.0. We stumbled upon this through a DIY jubeat video that our consultant shared with us of another individual. In that video, the person utilized this microcontroller for the panels, but we decided that it wasn’t for us. The largest benefit is how compact it is. The person in this video utilized a separate computer entirely to run his game. For the software development, it uses a built-in Teensy Loader to load your code and reboots to run the code. To run code, specifically C or C++, you will also need to install programs such as the AVR-gcc and the AVR C library.

Touted features directly from the manufacturers include: compatibility with arduino software and libraries, versatile USB connectivity with any type of device, single-push push button programming, easy loader application, free development tools, interoperability on the big three operating systems, a teensy size, and the option to get a model with pins for solderless breadboards.



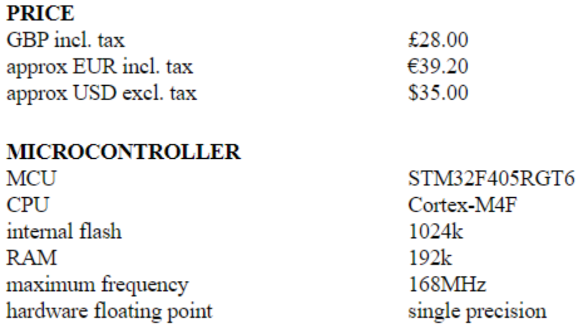
Specification	Teensy 2.0	Teensy++ 2.0	Teensy 3.0	Teensy 3.1
Processor	ATMEGA32U4 8 bit AVR 16 MHz	AT90USB1286 8 bit AVR 16 MHz	MK20DX128 32 bit ARM Cortex-M4 48 MHz	MK20DX256 32 bit ARM Cortex-M4 72 MHz
Flash Memory	32256	130048	131072	262144
RAM Memory	2560	8192	16384	65536
EEPROM	1024	4096	2048	2048
I/O	25, 5 Volt	46, 5 Volt	34, 3.3 Volt	34, 3.3V, 5V tol
Analog In	12	8	14	21
PWM	7	9	10	12
UART,I2C,SPI	1,1,1	1,1,1	3,1,1	3,2,1
Price	$16.00	$24.00	$19.00	$19.80

Figure 30 (Teensy Model Features): Convenient information table provided by Teensy's website for quick easy comparison between the models.

We originally decided that we wanted a controller we would be sure could handle both the buttons and the game instead. Now that we're considering two devices this has changed, however. The size, lower power requirements, and cost make this a viable and desirable option as a microcontroller for managing the hardware.

8.2.3 Pyboard

This is the official board of MicroPython. Python essentially acts like an Operating System for the board. It also contains its own built-in file system. It also has modules for accessing low-level hardware that is specific to the Pyboard. The board is not far in price from a Raspi at about $35, but it has considerably less specs. There are cheaper microcontroller options for handling our hardware, but this could be a consideration for managing software. MicroPython might be a good high level language to work with and the power consumption might be more manageable than the Raspberry Pi.



PRICE	
GBP incl. tax	£28.00
approx EUR incl. tax	€39.20
approx USD excl. tax	$35.00
MICROCONTROLLER	
MCU	STM32F405RGT6
CPU	Cortex-M4F
internal flash	1024k
RAM	192k
maximum frequency	168MHz
hardware floating point	single precision

Figure 31 (Pyboard Features): An excerpt of the most essential features of the PYB v1.1 features.

8.2.4 Nvidia Jetson Nano

Pardon the informal language, but this thing is an absolute beast. This microcontroller touts the highest costs and specs we would even consider for our project. This could do and handle everything, including putting a strain on our goal unit cost. A positive aspect is the specific skill of knowing how to code for this looks fantastic to employers. The other most notable factor is how this would raise the ceiling on our game library having a GPU bound constraint. This product might also allow us to forgo a microcontroller for a screen panel and handle that itself. This really would be an all-in-one, which would make it a more manageable cost, but less divisible project. This controller has two versions with different amounts of RAM: a 2GB version and a 4GB version. For our use, it is likely that we will not need 4GB of RAM to run any of the games we intend to run on this.

With further deliberation we have considered that the Nano could actually save costs since it can handle an LCD output with higher fidelity *and* manage our software. The technology also facilitates AI development should we find a reason to expand into that domain further. The controller boasts being highly power efficient with its 5 watts (also has a 10 watt mode). There would still be a desire for a controller to handle the switches and other hardware though. With the high cost of the Jetson Nano, we would require a highly budget friendly option for the hardware controller. If we decided to go with this board, it is very likely that we would go with the 2GB version since 4GB is unnecessary, and it would save us an extra $40.

More discussion has us lean even more towards the Jetson Nano. We decided as a group that the pros outweigh the cons on this controller for us. The best non-feature of the controller by far is how perfect it is to develop everyone's resume. Nvidia designed the Nano with AI in mind. Many AI applications and projects utilize the microcontroller. Even if we were to have minimal implementation of any AI, we would be working with a device that would put valuable experience in our repertoire which employers of AI positions would find useful. This aligns with the desires of our group members. The architecture focuses on parallel processing, which might end up being useful for implementing a grid based device and definitely will be useful for the display aspect of our device.

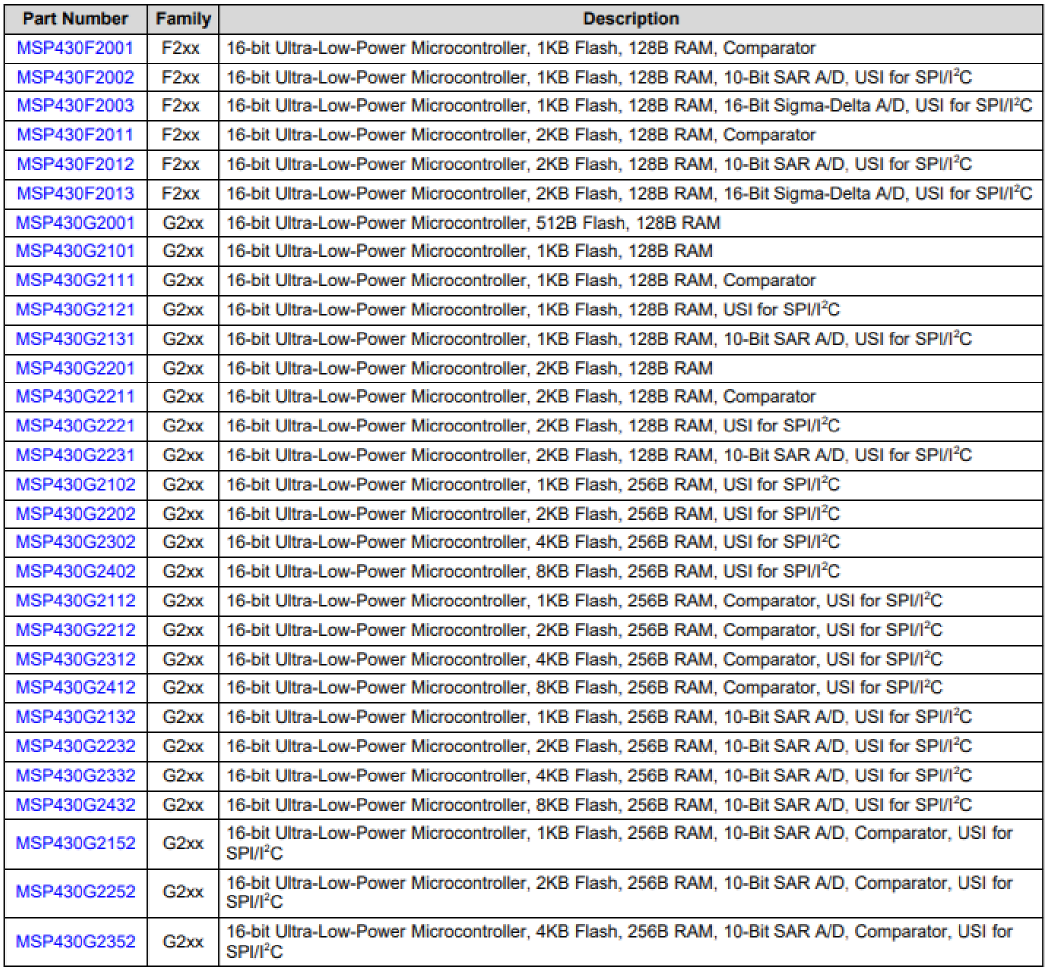
8.3 MSP430

We are including a separate section for research on the MSP430. Unlike other products like the Jetson Nano, the MSP430 is a line of models, so we need to select one and go from there. We have one serial in and one serial out planned for the MSP430, two clocks (one associated with each serial), and a need to communicate with the Jetson Nano. This means that we need at least four I/O pins and some standard for communication. For example, an RS-232 port could handle inter-controller communication. An additional I/O or two might be desirable in case we end up wanting to handle sound with the MSP430. Many of the products in this family come in series which allow choices of things such as Flash Memory, ROM, RAM, GPIO, ADC choices, and additional peripherals.

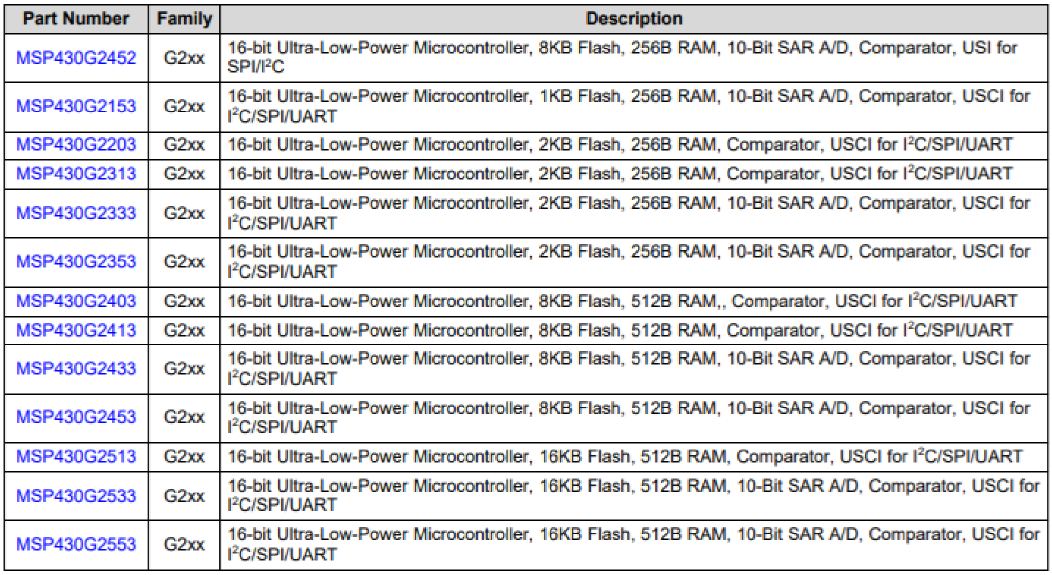
8.3.1 DIP Compatible Launchpad

One of the nicest features we can look for in terms of convenience when selecting a chip is one that has a development environment with a dual in-line package socket. Using a chip that can fit into a DIP socket lets us develop conveniently with a LaunchPad that has one. We can then equip our PCB with such a socket and transfer to and from the PCB and LaunchPad. This makes flashing our program onto the microcontroller and updating or modifying it a breeze. As for repair considerations, it also makes it easier to repair should the chip fail.

If we prioritize such a convenience we already can hone in on a few chips. Namely, anything in the MSP430G2xxx series and the MSP430F20xx series that has either 14 or 20 pin packages. This table from the TI datasheet explicitly lists the models we could use:



Part Number	Family	Description
MSP430F2001	F2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 128B RAM, Comparator
MSP430F2002	F2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 128B RAM, 10-Bit SAR A/D, USI for SPI/I^{2}C
MSP430F2003	F2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 128B RAM, 16-Bit Sigma-Delta A/D, USI for SPI/I^{2}C
MSP430F2011	F2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 128B RAM, Comparator
MSP430F2012	F2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 128B RAM, 10-Bit SAR A/D, USI for SPI/I^{2}C
MSP430F2013	F2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 128B RAM, 16-Bit Sigma-Delta A/D, USI for SPI/I^{2}C
MSP430G2001	G2xx	16-bit Ultra-Low-Power Microcontroller, 512B Flash, 128B RAM
MSP430G2101	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 128B RAM
MSP430G2111	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 128B RAM, Comparator
MSP430G2121	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 128B RAM, USI for SPI/I^{2}C
MSP430G2131	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 128B RAM, 10-Bit SAR A/D, USI for SPI/I^{2}C
MSP430G2201	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 128B RAM
MSP430G2211	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 128B RAM, Comparator
MSP430G2221	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 128B RAM, USI for SPI/I^{2}C
MSP430G2231	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 128B RAM, 10-Bit SAR A/D, USI for SPI/I^{2}C
MSP430G2102	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 256B RAM, USI for SPI/I^{2}C
MSP430G2202	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 256B RAM, USI for SPI/I^{2}C
MSP430G2302	G2xx	16-bit Ultra-Low-Power Microcontroller, 4KB Flash, 256B RAM, USI for SPI/I^{2}C
MSP430G2402	G2xx	16-bit Ultra-Low-Power Microcontroller, 8KB Flash, 256B RAM, USI for SPI/I^{2}C
MSP430G2112	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 256B RAM, Comparator, USI for SPI/I^{2}C
MSP430G2212	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 256B RAM, Comparator, USI for SPI/I^{2}C
MSP430G2312	G2xx	16-bit Ultra-Low-Power Microcontroller, 4KB Flash, 256B RAM, Comparator, USI for SPI/I^{2}C
MSP430G2412	G2xx	16-bit Ultra-Low-Power Microcontroller, 8KB Flash, 256B RAM, Comparator, USI for SPI/I^{2}C
MSP430G2132	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 256B RAM, 10-Bit SAR A/D, USI for SPI/I^{2}C
MSP430G2232	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 256B RAM, 10-Bit SAR A/D, USI for SPI/I^{2}C
MSP430G2332	G2xx	16-bit Ultra-Low-Power Microcontroller, 4KB Flash, 256B RAM, 10-Bit SAR A/D, USI for SPI/I^{2}C
MSP430G2432	G2xx	16-bit Ultra-Low-Power Microcontroller, 8KB Flash, 256B RAM, 10-Bit SAR A/D, USI for SPI/I^{2}C
MSP430G2152	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 256B RAM, 10-Bit SAR A/D, Comparator, USI for SPI/I^{2}C
MSP430G2252	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 256B RAM, 10-Bit SAR A/D, Comparator, USI for SPI/I^{2}C
MSP430G2352	G2xx	16-bit Ultra-Low-Power Microcontroller, 4KB Flash, 256B RAM, 10-Bit SAR A/D, Comparator, USI for SPI/I^{2}C



Part Number	Family	Description
MSP430G2452	G2xx	16-bit Ultra-Low-Power Microcontroller, 8KB Flash, 256B RAM, 10-Bit SAR A/D, Comparator, USI for SPI/I^{2}C
MSP430G2153	G2xx	16-bit Ultra-Low-Power Microcontroller, 1KB Flash, 256B RAM, 10-Bit SAR A/D, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2203	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 256B RAM, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2313	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 256B RAM, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2333	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 256B RAM, 10-Bit SAR A/D, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2353	G2xx	16-bit Ultra-Low-Power Microcontroller, 2KB Flash, 256B RAM, 10-Bit SAR A/D, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2403	G2xx	16-bit Ultra-Low-Power Microcontroller, 8KB Flash, 512B RAM,, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2413	G2xx	16-bit Ultra-Low-Power Microcontroller, 8KB Flash, 512B RAM, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2433	G2xx	16-bit Ultra-Low-Power Microcontroller, 8KB Flash, 512B RAM, 10-Bit SAR A/D, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2453	G2xx	16-bit Ultra-Low-Power Microcontroller, 8KB Flash, 512B RAM, 10-Bit SAR A/D, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2513	G2xx	16-bit Ultra-Low-Power Microcontroller, 16KB Flash, 512B RAM, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2533	G2xx	16-bit Ultra-Low-Power Microcontroller, 16KB Flash, 512B RAM, 10-Bit SAR A/D, Comparator, USCI for I^{2}C/SPI/UART
MSP430G2553	G2xx	16-bit Ultra-Low-Power Microcontroller, 16KB Flash, 512B RAM, 10-Bit SAR A/D, Comparator, USCI for I^{2}C/SPI/UART

Figure 32 (MSP430 Model Feature Table): Comprehensive list of all models compatible with the MSP430G2ET LaunchPad for easy development. Taken directly from the datasheet for the LaunchPad.

These tables are also useful in the fact that they list the varying features between the processors for easy comparison for the following considerations.

8.3.2 Flash Memory, ROM, and RAM

How much we need is hard to determine without the software being written. In general, more is better with the only meaningful tradeoff being cost. For developing, it is probably best if we overestimate how much we will need to avoid problems or not having enough memory. As we measure how much memory our programs use, we can either decide to buy a different controller with less memory or just continue using the controller we already have in hand during development in the final build.

8.3.3 GPIO

We already established that we need at minimum four GPIO and possibly two more to accommodate potential extra design. Luckily for us all MSP430 come with at least 10 GPIO. We did not plan to have 64 different wires for our buttons, but some models go up to 80 or more pins. Any MSP430 in the table has at least 24 pins.

8.3.4 ADC

Should we have audio input both in our device and that utilizes the MSP430, to have a good quality analog-to-digital converter would be useful. The F20xx series offers some 16-bit Sigma Delta options for better quality, but the 10-bit successive approximation register might suffice. It is doubtful that this would matter much since the audio processing would likely need to be done on a separate chip since the MSP430 is busy processing our switches.

8.4 Programming Languages

The language we use will mostly be learned and determined based on the controllers that we find to be most appropriate. There were some considerations we have regarding a couple languages though in terms of hopes and preference. Preference of which obviously will influence our decision for controllers alongside the tradeoffs of the controllers. Whatever the high level language a controller has, the architecture specific assembly language it has is bound to be necessary to learn as well.

8.4.1 Python

Python is an expensive language requiring more processing resources than others. It also lacks the resource management and speed provided by an alternative like C. Python is desirable for us despite its shortcomings because of its power and ease of use. Python has access to a wide array of open source AI libraries, so if we augment our game engine with AI or think of additional features that utilize computer vision, Python will already have access to the tools needed to facilitate their actualization.

Since the CUDA has a Python wrapper for its architecture, we have discussed staying with Python as our primary high level language in all likelihood. It is what we were partial to before. We do have a bit of flexibility now with our language choice since there are several high level languages that are compatible with the CUDA. That is, since we are heavily biased towards using the Jetson Nano.

8.4.2 Micropython

Another highly favored language might be MicroPython - a Python based programming language designed for the limited resources of microcontrollers. MicroPython is a language that includes a small subset of the Python standard library. Using less resources might allow saving on unit production costs. If we were to go with the Pyboard itself, Micropython also has libraries specific to the Pyboard that may prove useful. Some of these include Drawing text, primitive shapes, pixel access methods, and touch screen methods. Since we are not yet sure of the extent of MicroPython library capabilities, Python has more appeal for our group. If MicroPython proves compatible and it is feasible within reason to port our work to a more desirable controller, then we might do so as the progress unfolds.

8.4.3 C

This language is one that everyone on the project is familiarized with and it has unparalleled convenience with access to the low level through memory manipulation. Transferring to and from the low level language also becomes easier with assembly directives. C also allows for especially lightweight and optimized software, allowing for more efficient battery usage, and faster runtimes. C is lower level than most of the high level coding languages, and has very direct interfacing for embedded programming. Many microcontrollers have their own development platform with their own C header files for C integration. The language allows for immediate implementation of I/O and tends to have much more transparent memory usage and hardware interfacing. C/C++ is also heavily supported by the CUDA Toolkit, having a comprehensive development environment for C/C++.

The primary drawback with C is its primitiveness. It has very few features and tools in the language itself, and most of the time involves lots of reinventing of logical systems. C would be a very time-consuming language to code in, especially for the AI portion of the GameFrame, but it can be done, and when done correctly can be especially efficient.

8.4.4 Assembly

Although it has already been mentioned that we will probably have some amount of assembly for any of our choices, it should be noted that which assembly we work with can be a valuable asset for job applications. The other thing to bring up is that assembly is very close to the hardware. As such, we expect that using assembly will definitely be more prominent for our hardware controller, while higher level languages will probably remain most preferable for the software device.

8.5 Code Composer Studio

Code Composer Studio is an integrated development platform for TI's embedded processors such as the whole family of MSP430. The IDE already includes the C libraries necessary to program their microcontrollers. CCS also contains a bunch of tools for development and debugging purposes, and is optimized to run C/C++.

8.6 CUDA Tools

CUDA has a variety of compilers and tools for developing in different languages. Looking at these tools gives us a sense of which language we might want to develop in. The tools we are looking at are those that are officially recognized and suggested. The aforementioned tradeoffs under languages also apply here in our considerations.

8.6.1 CUDA Toolkit

The CUDA Toolkit is a set of tools provided by NVIDIA for developing C and C++ code with an included compiler. It seems to be the leading set of tools provided and also recently added support for the NVIDIA Ampere architecture. The toolkit has a suite of useful tools including GPU libraries, debuggers and optimization tools. Aside from varied tools that we might find useful, most of the tradeoffs for this one would be based on the benefits for using C and/or C++.

8.6.2 PyCUDA

With our current bias towards Python due to its ease of use and libraries, these tools in the form of a Python wrapper for the CUDA driver and runtime APIs stick out to us. The worst part of PyCUDA is that it is a preview software. It does add the functional ability to code in Python on the architecture and compile with the same parallel processing based programming for CUDA-capable GPUs (eg: Jetson Nano). Being in development means that it does not have many of the bells and whistles the other toolkits might afford us. Though the extra features might add convenience, so too might the ability to code in Python. How optimized it compiles and runs is something to keep in mind as well, since we do not want poorly optimized code to ruin the user experience.

One of the biggest advantages to using PyCUDA is it's automatic memory management. While CUDA has a lot of memory management features, it is complex and can cause a lot of difficulties since you have to manually manage its memory. When using PyCUDA, the memory is automatically managed. When Python goes out of scope, PyCUDA automatically frees the allocated memory. Another issue with CUDA that PyCUDA solves is error handling. Errors captured by CUDA can be complex because some operations can run concurrently with code on the CPU. PyCUDA automatically checks CUDA errors and turns them into Python exceptions that can be more easily understood.

8.6.3 Alea GPU

If we wanted to write .NET applications and use F# Alea GPU would be our goal. This is an easy to install package that has JIT compilation as a feature. There is some debugging functionality within the tools. The problem with this is that using the .NET framework and scripting is less than ideal for our code. Running precompiled software would be more preferred on a portable embedded computer.

8.6.4 NVIDIA HPC SDK

Since our device does not plan to do any intensive modeling or simulation, this might prove a bit too robust. It supports similar functionalities as the base toolkit for similar languages and frameworks, but it focuses on high performance optimization. These intense applications often utilize a lot of power. Thus, development kits which might expect high powered operation do not seem optimal for our low power portable device. One thing that does seem appealing is this toolkits advertisement for portability. This of course is a different kind of portability referring to its ability to facilitate software porting for if we ever felt the need to compile our code across multiple devices. As our goal is to code and design features best for our device, while also being able to tailor hardware to the software as we need, software portability does not seem all that appealing.

8.6.5 Altimesh Hybridizer

What differentiates this is that it focuses on generating vectorized code. In addition to C++, C, and .NET assemblies, it also can do so for Java. There is no huge benefit to bloating our machine with the JVM in order to develop in Java. Python has similar or better advantages to Java at that rate. Vectorized code might deserve consideration as useful when our entire device is based on an 8x8 array. Additionally AI benefits from vectorization of calculations. I suppose at that point it is obvious why NVIDIA would provide an option that generates vectorized code then.

8.6.6 OpenACC

These directives focus on ease of implementation and programming for faster application performance. Like the HPC SDK, there is also an emphasis on portability. Portable code is not a priority nor is ease of implementation. Though ease is nice, it lends towards not being as great of a tool to use when we want to build skills for our resumes. As we have more extensive programming backgrounds, we are not the target audience for OpenACC.

8.6.7 OpenCL

OpenCL stands for Open Computing Language and is actually a low-level API. OpenCL might be useful for us to be able to work on the hardware because it is lower-level. It is intended for CUDA GPUs like everything on this list, so it would lend itself as a good API for low-level handling on the Jetson Nano. If we end up using a separate controller for the hardware, the use of OpenCL drops dramatically. Our software definitely wants to utilize a high level language for coding.

8.7 Displays

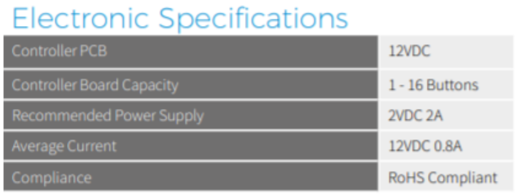
Rather than a single LCD panel with the 64 different buttons we considered using 64 partitioned displays for each button instead. This leads to two primary options for the display - partitioned or unpartitioned.

8.7.1 Single LCD Panel

It might prove an easier feat to implement the partitioning of the segments on the software side, so a single panel might be preferred. Also, due to the cost of the individual panel being considerably lower than buying 64 units of the various segmented options we prefer a single panel. Both of our inspiration sources utilize one LCD. The handheld and portable chess devices on the market also have an individual LCD screen albeit smaller. One screen also allows for more versatility on the software side, especially if creating additional games. The operating voltages and current for this seem reasonable for a portable device, but there is concern with the additional amount required for the backlight.

Another less convenient but much cheaper solution to implementing one screen is buying a microcontroller designed for display control and buying the screen separately.

8.7.2 Separated Display Options



Electronic Specifications

Controller PCB	12VDC
Controller Board Capacity	1 - 16 Buttons
Recommended Power Supply	2VDC 2A
Average Current	12VDC 0.8A
Compliance	RoHS Compliant

Figure 33 (Data Excerpt for SuzoHapp LCD Button): Though it can operate up to 16 buttons, the concern stems from needing 0.8A for only a fraction of our total buttons.

Aside from complicating the software and hardware design, especially at the interfacing level, quality panels for segmentation boast a significant cost increase due to needing 64 of them. The prices of the particular LCD buttons provided are unavailable unless contacted, so an accurate price estimate of these are not yet known. SuzoHapp as a manufacturer, while providing quality parts and giving good ideas for components that might fit in our device, tend to have costly products. Using 64 of these drawing .8A each is quite worrisome for portable device design too. A positive is that none of the housing or frame space between buttons will be wasted pixels on screen. The cost of repair and maintenance might also be better should one of these fail compared with an approximate 10” LCD display.

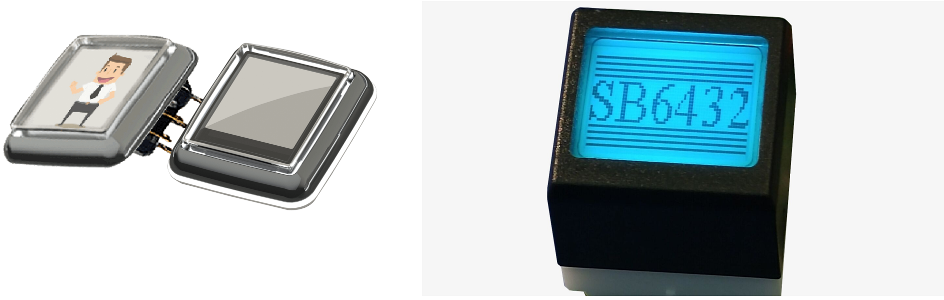


Figure 34 (Sample Segmented Options): Two pictures to help depict how the separated displays might look. The left is the aforementioned SuzoHapp product. On the right is one produced by LCD-Keys (I/O Universal Technologies).

8.7.3 OLED

Some sources suggest OLED can be a cheaper alternative. Investigation leads to seeing roughly the same prices as LCD. OLED has the ability to be more power efficient than LCD, but it requires the brightness be turned down, thus reducing the image quality. A lot of LCD screens draw most of its power for the backlight. OLED might be able to have a more desirable power to brightness. This will reveal itself more as we look into specific makes and models of screens rather than general tradeoffs. It should also be noted that OLED has the potential to be incredibly thin. Being so thin would have an advantage towards the weight requirements. There are the same tradeoffs for a single OLED panel or 64 separate panels.

8.8 Housing

Not many options came to us in regards to housing our production. We are fortunate to be in an age of 3D-printing, so it seemed an obvious choice for us. Still, we want to leave room for other considerations should they arise.

8.8.1 None/Cardboard

We obviously have no desire to create a device housed in cardboard or without housing. The reason this option is mentioned is simply for early prototyping. The cost is as minimal as it gets. At some point in the process we might even use this option as a way to verify dimensions.

8.8.2 Crafted housing

Similar to buying acrylic for mounting the PCB or for making buttons, we could fashion our own housing out of some plastic (pre-cut or cut ourselves) and screw them together ourselves. This could be cheaper, but also requires some handiwork and might end up looking less professional than a 3D-printed alternative, which should be comparable in price. Professionalism might not be as important in prototyping stages, but at that rate nothing or cardboard is cheaper.

8.8.3 Plastic Enclosures

Some companies, like Polycase, sell premade plastic housing. These can be quite affordable, but they are less customizable. Instead of using these, it might be better to just 3D-print a more custom tailored one ourselves.

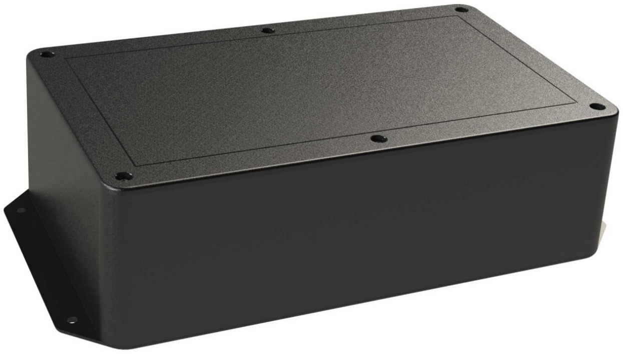


Figure 35 (Sample Polycase Enclosure): The plastic enclosures feel best described by a picture.

8.8.4 3D-printed housing

3D-printers are able to create good prototype housing while also being reasonably accessible. The monetary cost is reasonable, but prototyping and mistakes definitely cost more time. CAD knowledge is required to create something with this method, but this is seen as a boon for the project, since this kind of experience is nice for our resumes.

We only have access to a 200x200x250mm (~7.87x7.87x9.84inch) for the time being. So for all of the options listed, we would either need to design our housing to fit or be melded together in some way, or we need to use an alternative to 3D printing for the case.

One such alternative is to use a CNC router to develop a housing for our device out of acrylic or other solid plastic. This approach might require some extra handiwork if used in order to smooth some of the rough edges.

3D Printer Filaments

3D printers come with a variety of usable materials. The advantages and disadvantages are accordingly diverse. The materials focused on for the beginning of the project are those that are more suitable for prototyping. Some materials that might be considered higher end or more industrial are not that much more expensive.

PLA

PLA is considered one of the easiest to print with due to its low printing temperature requirement and its low flexibility. PLA is susceptible to distortion if left in heat. For example, leaving a portable game device in the backseat of a car, especially in Florida, might cause warping if the device were made of PLA. It is also a bit more brittle as a material compared to other options due to being inflexible. There are a variety of colors and even some options of filled materials for certain textures as a superfluous benefit. The filled materials include wood-filled or metal-filled filaments to name a few. One can also acquire PLA with material that glows-in-the-dark as another option. Most of the properties of these diluted materials are not beneficial for us and just a tad worse than the non-filled PLA alternative. PLA seems rather appealing as a prototyping filament though, especially as it is one of the materials we already have access to a 3D printer for.

ABS

ABS demands a tad more difficulty when it comes to printing. The benefit is that it is more durable and more heat resistant. ABS is seen as a tad superior to PLA. It has a tendency towards shrinkage or warping while printing. There are also toxic fumes to be wary of while printing. These are downsides during production however, and would not affect our device negatively. With a similar cost, ABS might be better than PLA for us, but not for prototyping. While first building it, a material that is prone to errors and might need reprints is much worse and will cost more while prototyping.

PETG

PET or its related materials PETG and PETT lie somewhere in between the benefits of PLA and ABS. It is a bit harder to print well than PLA, but easier than ABS. It is less brittle than PLA due to its flexibility and durability. The downsides include being susceptible to moisture and having a surface that can scratch easily. PETG has a similar cost to the aforementioned two filament types, but it also is a material that we have access to a 3D printer already set up for.

TPE

Not much is necessary to say for TPE in regards to our plans. TPE is highly flexible and has properties akin to rubber. For our case, we need a more rigid material. No components we have planned at the moment would need the features this material offers, but it's flexibility and durability are incredibly high and are good points to keep in mind.

Nylon

Nylon beats a lot of other materials in flexibility, durability and strength. The cost of it sits above the rest of the materials as well. This probably is not a material that we want to use, especially for early prototyping. It also must be kept dry like PETG.

PC
Polycarbonate is as durable if not more durable than Nylon. The main difference comes in the fact that it is less flexible. It requires an incredibly high temperature to print. It resists damage from both impact and heat quite well. It seems a bit more durable than necessary, but it also is not that much more expensive than other options. Its heat resistance should not be needed, but if temperature becomes an issue, we know which material to use.

Biodegradable
Though these are worthy of at least a mention because of being environmentally friendly, they tend to be more expensive and have worse qualities. The battery and other electronic components inside our device would need to be properly recycled and disposed of anyways, so having a biodegradable housing is moot.

Conductive
It is quite an interesting option to consider conductive carbon combined with another filament on this list. Though it is limited to low-voltage applications, this might be something to consider elsewhere in our design. Obviously, we do not particularly want conductive filament to be used for our housing.

8.8 Batteries

One of the most important factors for making our device portable will be the battery. Because the device is portable we have leaned heavily towards recharging as an option almost to the point that it's our decision. The downside is that cost is on the development and fabrication sides, rather than offloading it to the consumer's wallet. Disposable batteries also have the benefit of allowing the customer more playtime on the go without the need to recharge. That said, despite disposable batteries being better for monetary design, it is worse for the consumer's pockets and for the environment. We would not be knights if we did not have this bare minimum consideration for the environment in our design. That leads us to the most common materials for rechargeable batteries as our first consideration.

8.9.1 Alkaline

Alkaline batteries boast a decent energy density and are relatively cheap. If not charged properly these are more susceptible to exploding. They are not as lightweight as lithium options. Alkaline batteries tend to have higher internal resistance as well.

8.9.2 Lead-Acid, Nickel-Cadmium

Of the battery types both Lead-Acid and Nickel-Cadmium are older technology. They have a cheaper price point, but end up worse in energy density. Lead-Acid is a good consideration for being able to handle high currents. High current supply is good for a car, for example. Our device will not need this, so this material seems less desirable.

8.9.3 Nickel-Metal Hydride

Nickel-Metal Hydride is virtually a more modern version of Nickel-Cadmium batteries. Nickel-Metal Hydride have better capacity, retain the ability to keep fully charging, and are more environmentally friendly than Nickel-Cadmium. The tradeoff is that they are more expensive.

8.9.4 Lithium-Ion

Lithium-ion batteries are increasingly popular for portable devices, and for good reason. This material boasts a high energy density as one of its biggest advantages. It has a lower self-discharge rate while not in use, especially when compared to its Nickel-x counterparts. The disadvantages of this technology is that they require some extra protection. They must have protection from being overcharged or exceedingly discharged and also must maintain safe current limits. Some lithium-ion batteries include this protection themselves, but this disadvantage is necessary to consider since we would need to create the protection for it if not included and if we use this material type battery. If we go with a rechargeable version, it will also need an extra module to manage the battery. The last, and probably obvious, disadvantage that one would assume from its advantages is the cost. Lithium-ion batteries are more costly than other materials, but the cost might be worth it to attain our desired battery life.

8.9.5 Lithium-Polymer

These types of batteries are more of a subset of Lithium-ion. They have many of the advantages and disadvantages of Lithium-Ion batteries. These end up being a slightly safer technology with even lower self-discharge. They have a lower capacity overall, but they do have slightly better energy density.

8.11.1 Jetson Nano options

HDMI/Displayport
One plausible option for outputting sound is to use the HDMI or Displayport functionality included with the Jetson Nano. This is by far the simplest solution to do sound (and video). The problem comes in how it will limit our selection of displays to ones that can do audio and that have these standard connections. This also will limit or entirely remove our possibility to read audio input from users, should that be a feature we want to add.

USB
If the built-in video output options are not used for audio, especially as the display might not be sound capable, our other option is via USB. There are various plug-and-play USB sound cards that can be used with the nano. The problem with these is obviously that they are not included, will have to be purchased separately, and take up space.

Process with Nano or with MSP430
The Nano has no native support for audio. The only standalone option is the aforementioned video ports that also do audio. The MSP430 however has options to do audio output with its DAC. It also can use PWM to achieve the same thing. Texas Instruments even provides robust information in a pdf titled “Low-Cost Speech With MSP430 MCUs (Rev. B).” Any information

spelled out for us in such a way is incredibly useful for implementation. If we are not using HDMI/Display, the MSP430 seems more appealing. In such a case, the sound should be included on the same PCB as the MSP430. The sound would likely be stored and processed by the Jetson Nano, so it would be an extra step of convolution to have to send the information over the MSP430 first.

8.11.2 Speakers/Sound Chip

If we do use the MSP430 for processing the sound, or depending on the USB adapter used for the Jetson Nano if we use that, we would need to use speakers or a sound chip for our design. Some USB adapters will already handle this, but others would not. For the MSP430, it would be able to act as the sound chip, process the sound, and then need speakers to output. However, the plausibility of acting as a sound chip is doubtful, since it will already be handling the switches. The Jetson Nano would need an extra chip or IC for processing and digital-to-analog conversion *and* speakers. Since we are not leaning towards this option at this time we did not want to spend much time yet investigating chips and speakers.

8.12 Switch Debounce

There are several options for dealing with the error that comes from switch bouncing. It might not be entirely needed, but it is something we would like to design for while prototyping. This way we can decide on an option if it is needed, and to what extent.

8.12.1 Hardware solutions

NAND
This hardware implementation will lead to the best non-IC result. It uses two NAND gates together to implement an S-R flip flop. The self feedback of the circuit results in preserving the output while the contacts cause bounce. The trade off of this is that the rising edge of the transition is curved. It also requires the most components to create.

IC
There are dedicated integrated circuits for reducing or eliminating switch bounce. The debouncing ICs are a bit pricier of a solution but might result in the overall best debounce on hardware. It is unlikely we will need to use one of these, but it is worth considering in case we have room for it in our design. It is also good if we decide that we want to improve the responsiveness of our device.

RC
Using a simple RC circuit we can get rid of the error from bouncing. Using a simple resistor and capacitor this is the most heuristic solution, while also being the lowest quality of our hardware options. Additionally a diode can be used to speed up how fast the capacitor would charge and gives us the option to speed up debouncing if we need.

8.12.2 Software solution

Rather than using any of the hardware solutions, the cheapest and easiest solution comes at the cost of memory and performance. It relies on implementing it in software. As we are running our loop on the hardware microcontroller, we are sampling our inputs from the switches. As we sample inputs, we can make sure that a switch input only changes a value if it consistently reads the same over a number of iterations of the controller loop. That is, we can keep a counter and only update the actual value if the counter reaches a threshold. Thinking about this, it is practically the software equivalent of the flip flops relying on feedback from prior values. Instead of a flip flop, it utilizes the dynamic memory on the microcontroller to keep track of the previous values.

8.13 PCB Development Software

As our project develops, we will need to come up with a schematic and layout to actualize our PCB. With options present, there was little to do but to look into them so we could research the features they provide. This way, we can choose which software to utilize when we are constructing our PCB.

8.13.1 SnapEDA

The first thing to bring up is not software for PCB design. SnapEDA is a library of footprints and CAD symbols that other PCB development programs can use. These are premade and as an article from the official SnapEDA blog says, “this allows them to focus on innovation and design optimization rather than the more mundane and tedious aspects of the design process.” (natasha, 2015).

8.13.2 Eagle

One of the two most popular PCB design softwares used at our university is Eagle. Members of our group have experience with Eagle from junior design. Typically Eagle is not free, but it does have licenses available for educational use. There is robust library support for Eagle. Rich library support is always welcome when choosing development environments.

8.13.3 KiCAD

This is the second most popular software at UCF from what we could gather. KiCAD is open source and thus always free. Similar to Eagle it has many options from its libraries. The idea of any and all functionality of KiCAD being free with no limitations definitely makes this option a top contender.

8.13.4 Ultiboard

This program from National Instruments is part of the Multisim suite of products that we are familiar with, and thus compatible with them. One of its touted features is its ability to do design rule checking in real time. A big drawback to this is that it is currently not supported by SnapEDA. There is an official response from the SnapEDA team which states this is on their roadmap (Team SnapEDA, 2016). We can keep an eye on any updates from SnapEDA to see if they end up adding support before we get to work with a program in the coming semester, should this be a program we are interested in using. The response was posted in 2016, so perhaps it is to be completed soon.

8.14 Voltage Regulators

Regulating voltages throughout our circuit will be a must. We will dive into and expand upon the specific ICs for regulating voltage as specific parts are picked. More generally we can contrast linear voltage regulators and DC-DC converters.

8.14.1 Linear voltage regulators

The most prominent thing to consider for linear voltage regulators is that they can only act as buck converters. They are also known for being power inefficient with most of this inefficiency due to heat. Heat is already of concern with our device, and with battery life as an important design point for us, inefficiency is also undesirable. Linear regulators are cheap, resilient to ripple, have a fast response and are not susceptible to switching noise.

8.14.2 DC-DC converters

Most likely we will choose from DC-DC converters to regulate voltage. These converters allow step-up converting and are more power efficient. These two criteria alone makes them better for our device. At the moment, we do not need step-up conversion, but a single DC-DC switch regulator can regulate all of our necessary voltage and current levels.

8.15 Cooling

Among the many parts researched, one of the biggest problems for any electronics is heat generated by components. One of these heat-producing parts is the microcontroller due to its processor. There were some microcontrollers that came with some form of cooling solution already pre-installed, such as the Jetson Nano with its pre-installed heatsink. It is important to have our components properly cooled for them to run efficiently. It is also necessary to prevent damage to our components and reduce the risk of a system failure as a result of heat. There are however, there are a few main cooling options to consider:

8.15.1 Air cooling

Air cooling utilizes the ambient air to carry heat away from the source and out of the system. It does this simply by using a fan or even multiple fans to circulate air. Ideally, proper air cooling solutions involve pulling in cool, fresh air into the system and passing that air through the heat source, which will transfer some of the heat to the air, and another fan will exhaust the warm air out.

The main benefits to air cooling is that they are relatively inexpensive and do not require some form of liquid solution to carry the heat away. This results in less parts to utilize air cooling, which also leads to less points of failure. If the fan breaks, it is easy to replace instead of having to replace the whole cooler if there is a leakage. It also has greater flexibility due to simplicity in nature air cooling is.

However, air cooling isn't without its limitations. Air has a low specific heat, which means that it has less capacity to absorb and transfer heat. In typical scenarios, air cooling also requires an extra element such as a heat sink or a heat pipe to initially absorb the heat. Air cooling can also be completely ineffective if air flow is not properly planned ahead of time. If done incorrectly, it's possible to recirculate heated air inside the system. Without proper circulation, heat is just transferred back into the source.

8.15.2 Heatsinks

A heatsink is a heat transfer device that acts like a middleman to transfer heat from the source to a fluid medium, such as air or water. It utilizes a highly thermal conductive material to absorb heat from a heat-generating source, which can be either passively or actively dispersed. Passive disbursement relies on the physics of air or water to transfer heat away. How this works is when heatsinks absorb heat, the surrounding air touching the heatsink will also have some of that heat transferred to it. This will cause the air to warm up, and air circulation is passively generated by warm air rising. As warm air rises, it carries heat away from the source, which then causes cooler fresh air to take its place to absorb heat from the heatsink again. The same principle can be applied to a water medium where water rises and passively circulates cool water to the source.

However, sometimes the heat source can generate too much heat and passive air circulation is not enough to properly cool a component like a CPU. If passively circulation is not fast enough to transfer heat away, then an active heat disbursement is needed. In a heatsink using air as a fluid medium, a fan would be attached to the heatsink to actively cycle cool air in and warm air out. In a heatsink using water, you would need to have a pump and radiator combo to pump water away from the source which is then cooled by the radiator.

There are many metals with different thermal conductivity, but the two main metals used in heatsinks are aluminum and copper. Aluminum is the cheaper of the two since it is less thermally conductive than copper. Copper is almost three times denser than aluminum, which results in a higher cost for its better performance and higher thermal conductivity. Selecting which material to use will depend on how much heat needs to be transferred.

8.15.3 Heat Pipes

A heat pipe is a heat transfer device that is very similar to a heatsink. Instead, a heat pipe consists of an envelope with a fluid and a wick structure to carry heat away from the source. Heat vaporizes the liquid inside the wick, which then carries the vapor with its heat towards the cooler section of the pipe away from the heat source. The cooling area contains a condenser where the vaptor loses its heat by condensing back to liquid form. The liquid then returns to the heat source through the wick and the process is cycled.

Some of the main benefits include a way higher thermal conductivity than air, can potentially be passive, low cost, and resistant to shock. Since it can be passive, implementing heat pipes as our cooling solution adds the benefit of a quiet system. If we so desire, you may also install a fan to combine air cooling with the heat pipes for an even better cooling solution. However, this does add back noise into the system due to the fan.

One of the main drawbacks is that heat pipes are not suitable devices that demand high power usage. The passive cooling on the heat pipe is not enough to cool most high power demands. However, this drawback is mostly negligible because this project aims to have low power usage. It may be a bigger consideration depending on what parts we pick. For example, the jetson nano might need a different cooling solution since it does generate a lot of heat.

Based on our options, it is likely that we will lean towards a solution that utilizes a mixture of these cooling solutions with air as our primary fluid medium to transfer heat. Water cooling, while more efficient, is definitely not necessary for our potential needs. Water cooling will add a lot more unnecessary bulk to our device. Since we are leaning towards using the Jetson nano, it helps that an aluminum heatsink is already pre-installed onto the board, which means copper will not be considered. Depending on how much heat is generated by the Jetson nano, one fan might be installed on top of the pre-installed heatsink. Luckily, there are fans already specifically designed for the Jetson nano with mounting already set up on the board. If air cooling is necessary, the design of the enclosure must be changed to include vents for warm air to leave the device. Every other component will be passively cooled with heat pipes or be cooled by air circulation generated by fans. As of now, the Jetson nano should be the main heat generating component.

8.16 Algorithms for the Computer Player

To play against the computer, a type of AI or bot will be used. Throughout the paper, the terms AI, bot, and computer have been used interchangeably to address the algorithm that will play against the human player in 1-player mode, however it is not necessarily determined that an AI is the best solution. The game of chess for instance has more possible games than could ever be played, or stored in any realistic hard drive, let alone the small factor drive that will be not larger than 256GB, likely much less. This being the case, utilizing something like an artificial intelligence may be the best route to take in order to solve a game of chess. On the other hand, a game of tic-tac-toe has a relatively small number of total games, and is not especially complex in the decision making process. For the computer, a combination of AI and algorithmic solving, such as backtracking, will be used to play against the human player.

8.16.1 Artificial Intelligence

Artificial intelligence is one of the primary uses of the jetson nano board. This being the case, AI may seem like the all around best option for all solutions of each game. AI however has the downsides of being an incomplete system, and does not always choose the absolute optimal solution. AI is further limited by the fact that lots of useful input needs to be created and input into the training models for the AI to gain any ability to function. This is timely and may require lots of time and resources to train. A system utilizing AI is also prone to errors, as an AI tries to get to its goal given only exactly what the rules are it has to abide by, and nothing more. This means for instance if an AI is incentivised not to lose, it may simply choose not to play at all, making it impossible to lose.

A key upside of artificial intelligence is the modifiability of difficulty that the algorithm plays the game on, and the lack of need to fully program in the solution to the game. The primary way that an AI is trained is essentially it is told how to do something, and what its goal is, then seeded with some randomness to do that, and rewarded once it gets closer to a goal. For instance, if the goal of an AI is to drive from point A to point B as quickly as possible, it may be told how to spin the wheels of its body, use its arms, and change the speeds of each component. Then as it gets closer to point B it may start receiving rewards, or sometimes only once the goal is achieved does it get a reward. The AI then makes a map of what types of things resulted in getting to a reward through a process called convolution. This means that the AI abstracts each thing it does into "ideas" and puts the best ideas at the highest priority and the ones that worked the least at the lowest. The AI will then continue trying to solve the problem by raising and lowering ideas in its priority convolutional network (called a convolutional neural network due to its resemblance to neurons).

For the sake of this project, this is a useful feature to have, as not every aspect of a game of chess for instance will need to be programmed in. It is known to chess players such things as moving a king to a corner can be a good strategy for success, having a castle at the bottom or 2nd to bottom row is beneficial, certain pieces are worth more than others, pawns are typically worth more as just sacrificial pieces, certain opening moves result in more favorable outcomes, etc. but this is quite cumbersome to program into a computer, especially with the fact that many of these cases have numerous caveats and exceptions. AI allows for the programmers to not know much about the strategy of chess and simply reward the AI when it is doing well.

8.16.2 Backtracking

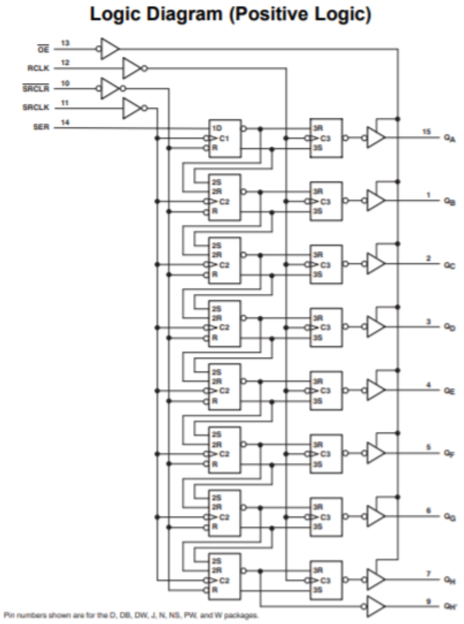
A useful strategy to solve problems that have lots of discrete moves is an algorithm called backtracking. This process essentially sets up many instances of simulated moves, where the algorithm "explores" lots of different paths, then once it gets to a point that is undesirable, backs up a bit until it hits a place where it can make a change in decision and keep going. A good example of this is something like a maze solver, where the backtracking algorithm may choose to turn right at every intersection. Once it hits a dead end, it backs up to the last right, then makes a left. It then continues making rights until it hits another dead end, in which case it repeats the same problem. If no additional right exists, it backs up another step. Ultimately, this should result in a solution, but this is not always the case

In backtracking, there exist numerous gridlock states. In the maze solver example from earlier, if the end is surrounded by an island, the backtracking algorithm may never be able to move there, as it could back up beyond the place where turning would get it to the end. With game solving, this is a concern, as sometimes the optimal solution to a game logically may not be accessible, and the algorithm could lose out on many more optimal solutions. Additionally, it may take a substantial length of time to run through every solution to a game, or at least enough to make a good next move. Backtracking needs to have a certain type of solution set for it to be able to run well.

For games like tic-tac-toe, 4-in-a-row, and potentially checkers, backtracking is probably still the better bet. As mentioned earlier, AI is very resource intensive to get up and running, however a backtracking algorithm can be functionally complete in very little time. These algorithms simply need to be told what the rules of the game are, how to play, and a basic strategy that will allow them to test through the game, then they are ready to go. This is good for a game that has few rules and limited potential for cases where the algorithm may not be capable of arriving at a favorable end state. Additionally, the Jetson Nano will make it very possible to optimize these algorithms, as numerous different instances of paths can be explored in parallel, resulting in the desirable 1-second time to move.

8.17 Shift Registers

There is not much to consider when it comes to shift register ICs. The only important factor is how many bits the register has. The 74HC595 is an 8-bit shift register that would work for us. In case we would like additional bits to work with a 16-bit register in the form of the 74LS674 is listed. The extra bits would allow expansion of hardware design without loading more onto our controller's other I/O pins. In the event of expanding the hardware design, we would probably just utilize the other I/O pins that we have. Even more likely, we might need an additional microcontroller to handle other hardware processing.



<u>Figure 36 (74HC595 Logic Diagram)</u>: Depicts the logic gate architecture of the 8-bit registers. Taken from the part's datasheet.

9. Budget and Financing

Item	*Cost*	*Source*	*Description*
3D-printed Housing	$20 (about one roll of ABS or PETG)	Amazon	ABS Filament
7-in LCD Display	$30-$60	Digikey	
Acrylic Squares/Buttons	$3 - $ 28 (depending on thickness)	US Plastic	
Acrylic mounting panel	$3 - $28 (depending on thickness)	See above	
PCB	$0.50 - $300	Gerber Labs	
Snap Action Switches	$64 - $192	TE Connectivity	Estimate is for many switches
Arcade-style Buttons	$2.30	Northeast Coast Customs	Only need 1-2
Software Controller	$35	Raspberry Pi	Raspi 4 Model B 2 GB
Hardware Controller	$12	Digikey	MSP430G2553
Battery	$13	Amazon	12V (1 AH)
Screws	$14	Ace Hardware	100 count
Total	$208.8 - $716.3		

Table 17 (Estimated Unit Cost): A tabulation and estimate total of the various parts to make up one unit

Item	*Cost*	*Source*	*Description*
No Housing	$0		
LCD Display	$26+$12 = $38	eBay Amazon	
Loose Wire	$24	Bulk Wire	

Tactile Switches	$0.18	Digikey	64 minimum needed, 256 preferred
Software Controller	$35	Raspberry Pi 4 Mobel B 2GB	
Hardware Controller	$12	Digikey	MSP430G2553
Power source	$13	Amazon	12V (1 AH)
Total	$145.52 - $180.08		

Table 18 (Estimated Prototyping Cost): This is like the prior estimated unit cost but with the goal of using cheaper materials or excluding others to prototype.

Item	*Cost + Shipping*	*Arrival Status*	*Purchaser*
MSP-EXP430G2ET	$21.28	Arrived	Frank
Total	$21.28		

Table 19 (Billing Materials): We have not yet started to spend on our project so this table is currently a placeholder. In the future it will denote the actual expenditures of our project.

Switches

Name	*Cost*	*Source*
Tactile Switch	$0.18	Digikey
Snap Action Switch	$2.91	Digikey
Silicon Push Buttons	$0.19	Alibaba eBay
Bulk Example	$0.001	Alibaba
Arcade-style Buttons	$2.30	Northeast Coast Customs

Controllers

Name	*Cost*	*Source*
Raspi 4	$35	Raspberry Pi
Raspi Pico	$4	Raspberry Pi
RP2040	$1	CanaKit
Teensy	$24	pjrc
Pyboard	£28 ($33.5)	MicroPython
Nvidia Jetson Nano	$99	Nvidia

Displays

Name	*Cost*	*Source*
Single LCD	$53	Digikey
Microcontroller/ Display Separate	$26+$12 = $38	eBay Amazon
64 Individual Displays	Not Yet Available	Suzohapp

Housing

Name	*Cost*	*Source*
None	$0	
Cardboard	$0.24	Packaging Prices
3d-Printed Housing	$20	Amazon
Crafted Housing	Unknown	
Plastic Enclosure	$15 (+$4 to include screws)	polycase

Table 20-23 (Research Cataloging): Costs of researched parts catalogued in the budget section for ease of use. Descriptions are excluded and explained in the research section of this document.

10. Initial Milestones

Milestone	*Milestone Type*	*Due Date*	*Details*
Functional input/output	Hardware	09/05/21	The microcontroller should be able to read user input from buttons on a 64-panel grid. Input should be sent into the microcontroller for usage in the game engine.
Application can run a game	Software	09/05/21	The software application should have the basic structure of a game (chess) set up, and logic/rules to the game implemented and interactable. Ex: bishops move diagonally, pieces are captured on being attacked, captured king wins the game
Application can play against user	Software	09/19/21	Given a working application, the AI should be able to play a game of chess against the user. It is responsible to both make a decision as to what piece to move, as well as send the game engine a signal to move the piece. It should stop processing moves on capture of a king, draw, or reset.
Physical board layout and encasing	Hardware	09/19/21	The playing board on which the users will interact with the game and AI should be laid out in such a fashion that the corresponding game pieces are in the position they correspond to in the game engine. Power switches, reset buttons, and menu buttons should also be attached in their permanent location.
Software to hardware integration	Hardware -Software	10/10/21	The hardware input from the completed player board should fully interact with all components of the application. No computer aid should be required to change game-modes, reset, power on/off, or enable/disable AI or two-player modes.
Fully battery powered	Hardware	10/24/21	The entire game board should fully run off of the battery pack. No external power input should be required, and battery life must last at least 2 hours (as per the requirements).
Fully standalone	Hardware -Software	11/07/21	The game board should not need any aid from wired or wireless computers, internet, or power cords, and it should be able to be transported freely without issues in stability. Ex: If the board is tilted while mid-game, this should not affect performance.
Working Product	Hardware -Software	11/30/21	Game board on startup should boot to an interface which allows the user to start a game, choose game modes, and play a game. Menu should be accessible at the end of a game, and when the menu button is pushed. Board should recover to a usable state after power drain.

Table 16 (Milestones): A general outline of our project development milestones throughout the two coming semesters

Appendices

Works Cited

Not directly referenced

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

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/.

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

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

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

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

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.

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

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.

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.

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.

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

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

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

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

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

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

Directly referenced

natasha. (2015, March 24). *What is SnapEDA? A Primer for Non-Engineers*. SnapEDA Blog. https://blog.snapeda.com/2015/03/21/what-is-snapeda-a-primer-for-non-technical-folks/.

Team SnapEDA. *Why no NI MultiSim/Ultiboard ?* SnapEDA. (2016, December). https://www.snapeda.com/questions/question/why-no-ni-multisimultiboard/.

Other Links

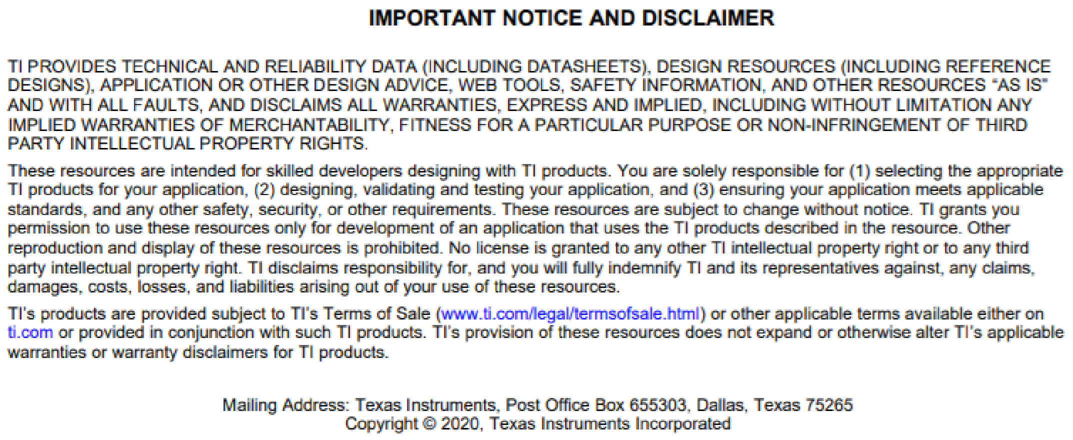
Switch Trace Image Source:
https://electronics.stackexchange.com/questions/281921/pcb-touch-button

Example of jubeat from a top-down view:
https://youtu.be/zfmbBTBlP4Q

Video of a DIY jubeat:
https://www.youtube.com/watch?v=bemd8mn8H4E&ab_channel=MaikeeMaous

Permissions

TI Datasheet Legal Disclaimer:



IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2020, Texas Instruments Incorporated

Datasheets and Reference Documentation

ICs or Controllers

https://www.ckswitches.com/media/1471/pts645.pdf
https://omronfs.omron.com/en_US/ecb/products/pdf/en-ss.pdf
https://www.pjrc.com/teensy/
https://www.ti.com/lit/an/slaa405b/slaa405b.pdf?ts=1626850739943
https://store.micropython.org/pyb-features
https://developer.nvidia.com/embedded/jetson-nano
https://oem.suzohapp.com/wp-content/uploads/2020/06/Snapshot-LCD-Button_Flyer_EN.pdf
https://www.ti.com/lit/ds/symlink/sn74hc595.pdf?ts=1627888610317
https://www.ti.com/lit/ds/symlink/sn74ls674.pdf?ts=1627957879522&ref_url=https%253A%252F%252Fwww.google.com%252F
https://www.analog.com/media/en/technical-documentation/data-sheets/36941fb.pdf
https://www.ti.com/lit/ds/symlink/msp430g2553.pdf?ts=1627963592296&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430G2553

Standards

https://ieeexplore.ieee.org/document/6204026
"IEEE Standard for System and Software Verification and Validation," in *IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004)* , vol., no., pp.1-223, 25 May 2012, doi: 10.1109/IEEESTD.2012.6204026.
https://ieeexplore.ieee.org/document/5930304
"IEEE Standard for Rechargeable Batteries for Cellular Telephones," in *IEEE Std 1725-2011(Revision to IEEE Std 1725-2006)* , vol., no., pp.1-91, 10 June 2011, doi: 10.1109/IEEESTD.2011.5930304.

https://ieeexplore.ieee.org/document/8320570
"IEEE Standard for Environmental and Social Responsibility Assessment of Computers and Displays," in *IEEE Std 1680.1-2018 (Revision of IEEE Std 1680.1-2009)* , vol., no., pp.1-121, 19 March 2018, doi: 10.1109/IEEESTD.2018.8320570.