Final Project Report

Virtual Reality Assisted Rehabilitation for Stroke Patient Therapy

Denovus

Nathan Barton – ETL
Dennis Gaona
Chase Gonzales
James Huang
Wongani Kalengamaliro
Neil Whitney

December 7, 2020

Nathan Barton
Email Address: nbarton0001@gmail.com
Phone Number: (972) 971-2100

Dennis Gaona
Email Address: dennisjgaona@gmail.com
Phone Number: (469) 350-3919

Chase Gonzales
Email Address: chase1.gonzales@gmail.com
Phone Number: (361) 355-5237

James Huang
Email Address: jhuan44044@gmail.com
Phone Number: (281) 725-2170

Wongani Kalengamaliro
Email Address: wkalengamaliro@gmail.com
Phone Number: (281) 763-3385

Neil Whitney
Email Address: neilwhitney67@gmail.com
Phone Number: (972) 809-0781

Acknowledgements

- UT Southwestern Physical Therapists
- We would like to thank Dr. Staci Shearin, Dr. Todd Polk, and Dr. Robert Hart for their contributions to the conception and design of this project. This project was supported by UTDesign.

Executive Summary

Loss of upper limb motor function is a common consequence of stroke injuries. Recent clinical evidence indicates that only 10 percent of people who have a stroke recover without any significant impairments [1]. Common issues in stroke patients undergoing rehabilitation through physical therapy include a lack of motivation to continue the physical therapy (PT) program and a general apathy towards rehabilitation due to its exorbitant price tags. These factors play an enormous role in a stroke victims' initial decision to commit to physical therapy. Unfortunately, the luxury of time is not one stroke victims are privy to as the small window of two weeks to three months is a deciding factor in the level of rehabilitation a stroke patient can expect to achieve[17]. Thus, a cheaper Physical Therapy alternative to in-patient rehabilitation that is easily accessible, and research backed is crucial for those stroke patients who are lacking in motivation to start, or continue, treatment as well as stroke patients with financial limitations.

Our program provides an inexpensive and engaging source of at-home Physical Therapy that targets the most common long-term impairment experienced by stroke patients, upper-limb paralysis. There are two wireless devices that are included with our system, an Oculus Quest VR headset and a custom-made sensor glove that is interlaid with 10 sensors to accurately track the user's movements. Utilizing VR enables the implementation of a physical therapy technique known as discordant visual feedback, which involves showing the user hand functions in the VR space that are different from what they are physically doing in real life. This PT technique is proven to increase activation in the motor regions of the brain and speed up recovery. The four mini games included in our program incorporate a variety of hand exercises that help improve finger dexterity, strength, and promote neuroplasticity. More research is necessary to determine the effectiveness of Denovus PT on actual stroke patients, and whether research backed claims of increased motivation and ADL ability are applicable to Denovus PT. Using this Physical Therapy in conjunction with outpatient Physical Therapy is recommended for maximum potential rehabilitation. Furthermore, Denovus recommends minor adjustments to both the sensor glove and the VR application. The sensor glove should utilize a smaller microcontroller and battery pack effectively making the design much sleeker and compact while still maintaining full functionality. The VR application should include more mini games of different exercise variety for a full clinical rehabilitation experience. Data output by the mini games should also either send to a companion phone Denovus app or can be sent to an excel file for easy access by licensed physical therapists. While these adjustments would increase the functionality of Denovus PT, the current prototype is a fully functional, standalone PT alternative ready for market or clinical sponsorship.

All software source files, CAD files, electrical schematics, and other electronic files associated with this project are contained in an electronic repository maintained by UTDesign.

Table of Contents

List of Figures

List of Tables

Introduction

Every year, approximately 800,000 Americans have a stroke [1]. A stroke is a sudden interruption of the blood supply to the brain often caused by blockage of arteries leading to the brain. Symptoms of a stroke include numbness or paralysis of the arm, face, and leg on one side of the body as well as trouble walking, speaking, and understanding. The symptoms vary on a case-by-case basis and are dependent on the location of the blood supply interruption, as well as the severity of the blockage. Sixty to eighty percent of stroke survivors experience physical impairments that require physical therapy (PT) to correct [2]. Only ten percent of stroke patients in PT will fully recover [3]. A stroke patient's recovery is dependent on their access to quality PT, their ongoing motivation to continue PT, and the time at which they begin PT post-stroke.

Physical therapy is a long, difficult, and expensive process. As shown in **Figure 1** below, the average cost for one year of PT for stroke patients is $11,689 and rehabilitation can take years to fully recover [4]. The exercises a stroke patient does in PT are physically exhausting, repetitive, and usually offer no immediate rewards or gratification to encourage rehabilitation. This can lead to a lack of interest and depression which is experienced by approximately one third of stroke patients at some point throughout rehabilitation [5].

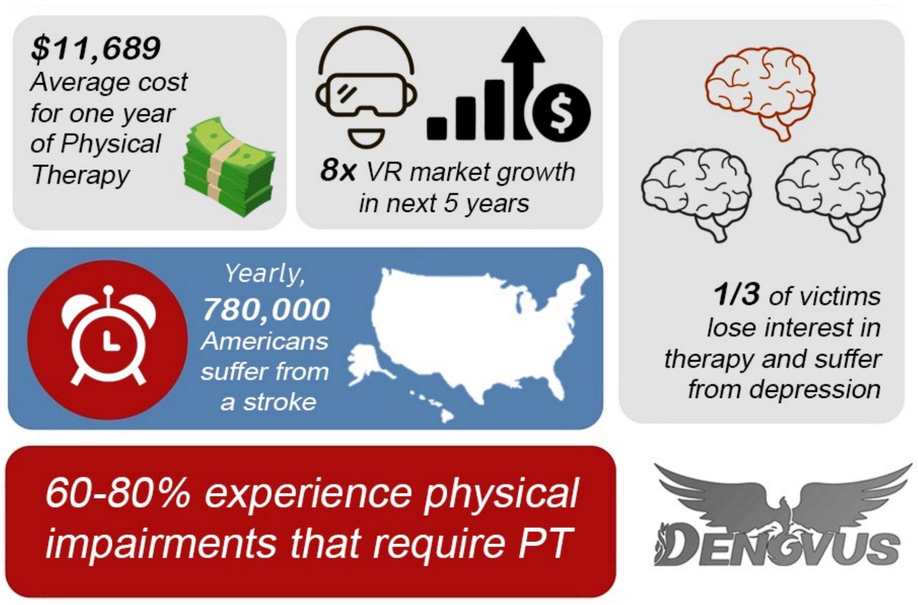

$11,689
Average cost for one year of Physical Therapy
8x VR market growth in next 5 years
Yearly, 780,000 Americans suffer from a stroke
1/3 of victims lose interest in therapy and suffer from depression
60-80% experience physical impairments that require PT
DENGVUS

Figure 1: Graphic displaying stroke data in America

As a rising number of individuals suffer from strokes (**Figure 2**), staying motivated during traditional PT exercises is a problem that some stroke patients struggle with, particularly stroke patients whose non-dominant hand was affected. To alleviate this problem, it is important to incorporate additional sources of motivation aside from just their physical progress. Providing stroke patients with entertaining exercises where they can play games, finish tasks, and achieve goals is more effective than conventional exercises because it provides motivation and helps to maintain constant concentration during the exercise [6]. Games help inspire, motivate, and trigger pleasure and interest in rehabilitation by providing stimulation and feedback that fulfills a natural desire for competition and interaction [6]. Furthermore, the inherent success and failure metric present within video games increases a patient's motivation to continue PT as they feel as if they have an active role in their own rehabilitation [6].

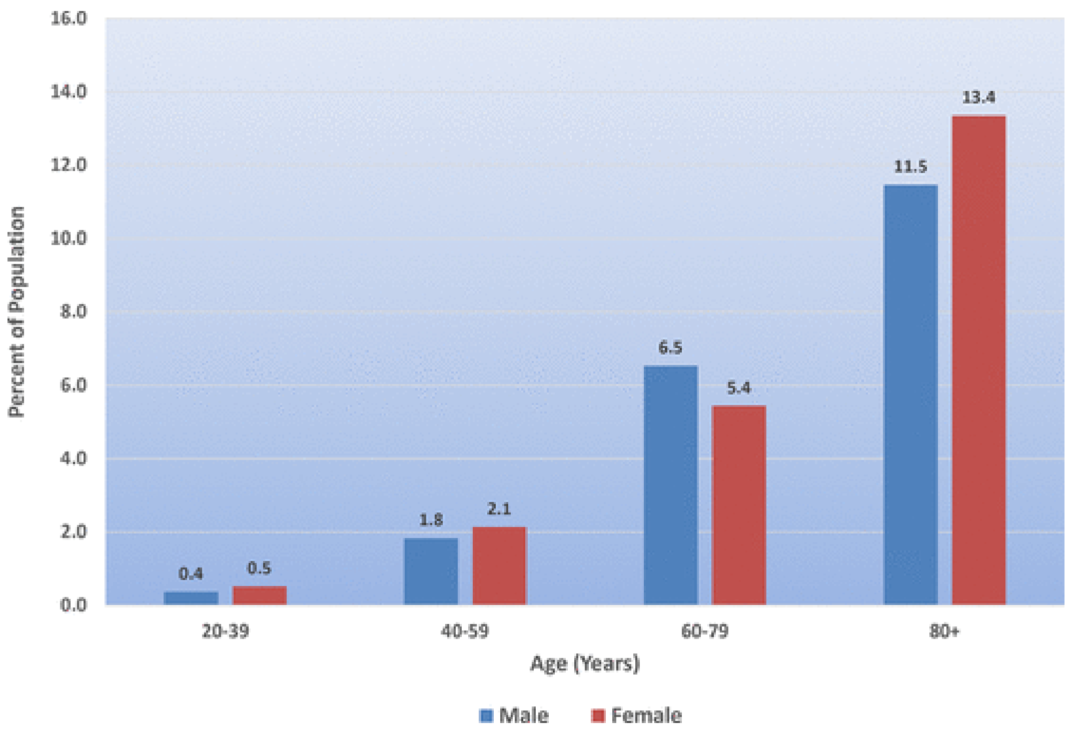


Figure 2: Plot for prevalence of stroke by age and sex [7]

Products that provide interactive VR games for stroke patient PT exercise exist on the market today. Companies like the NeuroRehabVR and GestureTek create VR devices that cost thousands of dollars and are only available at treatment centers. Patients must leave their homes and pay the treatment center fees to use the devices. There is a need for an at-home, cost-effective VR device for stroke patient PT to help supplement therapy in treatment centers during the middle and late stages of rehabilitation. The most common long-term impairment for stroke patients is hand and arm paralysis. An at home VR device designed to exercise a patient's hand and arm with interactive objects could help patients continue to progress through PT while significantly decreasing the amount of time spent at a treatment center.

Problem Statement

For many stroke survivors, tasks which were once considered simple become difficult and frustrating. Often the aftereffects following a stroke event change the lives of those affected by preventing them from performing everyday tasks, living independently, or returning to professional activities. In recent years there has been a rising number of elderly stroke patients increasing the demand for specific, targeted physical therapies (PT) [8].

Traditional stroke physical rehabilitation methods do provide modest benefits; however, patients often find the repetitive exercises involved monotonous and frustrating. If stroke patients become disinterested in the PT exercises, it can become difficult for them to maintain the motivation to continue their rehabilitation. The first six months following a stroke event is most often when stroke patients see the most improvement [9] and their ability to improve during this crucial time is often largely dependent on their individual effort. Maintaining motivation is an obstacle that stroke survivors must overcome to regain mobility and function; this imbalance between affordable therapies and effective therapies can be seen below in **Figure 3**.

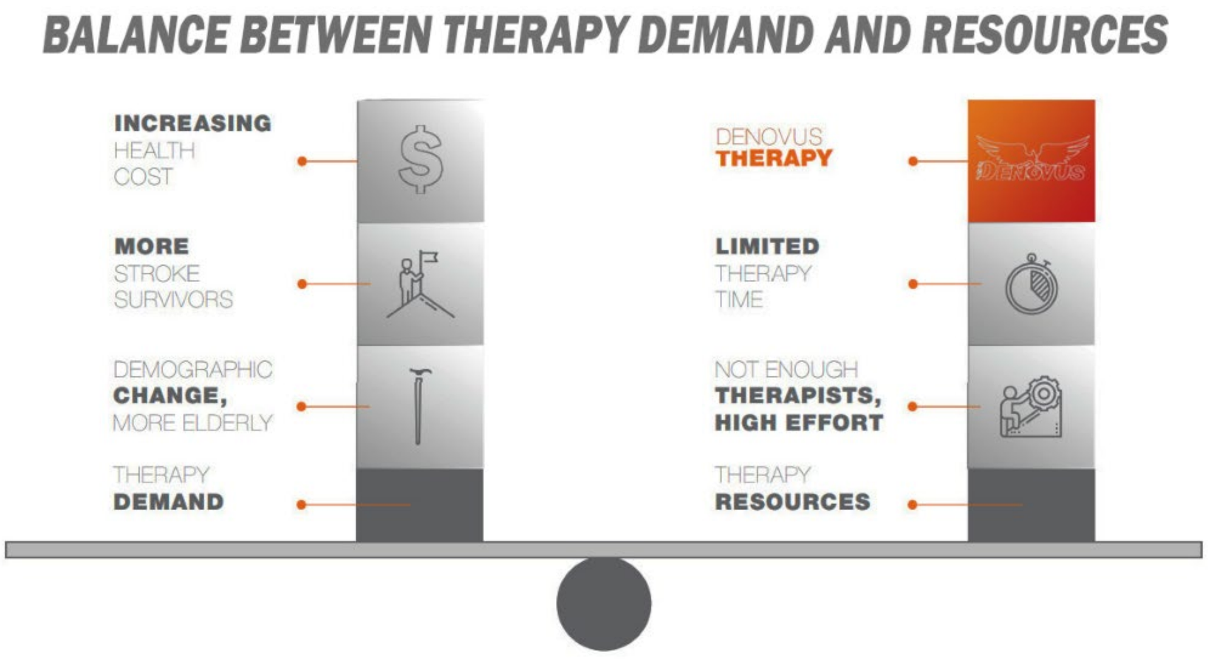
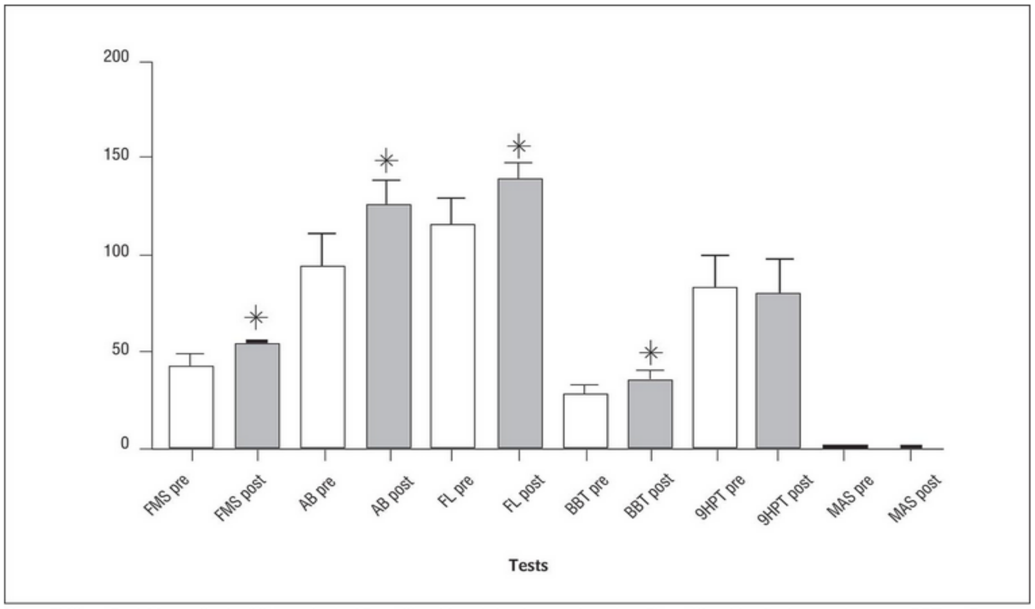

BALANCE BETWEEN THERAPY DEMAND AND RESOURCES
INCREASING HEALTH COST
MORE STROKE SURVIVORS
DEMOGRAPHIC CHANGE, MORE ELDERLY
THERAPY DEMAND
DENOVUS THERAPY
LIMITED THERAPY TIME
NOT ENOUGH THERAPISTS, HIGH EFFORT
THERAPY RESOURCES

Figure 3: Illustration showing the balance between the demand for stroke therapy and available resources [10]

Virtual Reality has proven to be a valuable tool in increasing effectiveness of stroke patient PT [11]. Existing VR solutions help address the issue of maintaining patient motivation during PT by making therapy more entertaining; however, they are relatively expensive, often costing stroke patients' families thousands of dollars. Current solutions are not optimized for at-home usage, often requiring the stroke patients to have outside assistance to initially set up and operate. Six months post-stroke event, 40-62% of stroke survivors are still dependent on outside assistance for some aspect of their daily living activities. Stroke survivors often lose their sense of independence due to the aftereffects of a stroke and it is important to promote and encourage survivors to do tasks independently when possible [12]. Additionally, current solutions lack exercises which incorporate interacting with physical objects. Sensory and muscular stimulation through physical object interaction has proven to help restore motor function following stroke [13] (**Figure 4**).



Note: FMS = Fugl-Meyer Scale (UL session); AB = Shoulder abduction goniometry; FL = Shoulder flexion goniometry; BBT = Box and Blocs Test; 9HPT = Nine Hole and Peg Test; MAS = Modified Ashworth Scale. ✳ $p \leq 0.05$.
Source: Research data.

Figure 4: Plot showing the results of Virtual Reality for upper limb rehabilitation of hemiparetic Stroke patients before and after 20 sessions between 15 and 30 minutes of duration [11]

The Denovus VR-Assisted System will be an inexpensive alternative to current VR solutions that will allow patients to utilize sensory and muscular stimulation through interaction with physical objects, promoting upper extremity motor recovery. The Denovus VR system will be an entertaining alternative to traditional PT exercises and will be optimized for at-home usage. Stroke patients will be able to set up and operate the system independently, with little to no outside assistance when setting up for first time users. The aim is that the Denovus VR system will effectively help maintain exercise motivation in stroke patients through an immersive and interactive experience.

Specifications

Physical therapy treatment encompasses a large range of exercises and techniques that aim to help certain affected areas of the body to recover their neuro-muscular connection. Most often, motor impairment at the hand is commonly affected after stroke. At 6 months after severe stroke, one third of people develop wrist and hand contracture (loss of passive joint range of motion) [2] and more than 50% of people with hand impairments do not regain function" [14]. Due to the necessity of having full control of one's hands for daily activities, Denovus PT is pursuing its efforts in tackling hand grip and flexion strength recovery. Below is a list of specifications for completing a VR Physical Therapy treatment with a primary focus on patient hand recovery.

1. Physical therapy session within a maximum time frame of 2 hours to prevent dizziness and nausea from extended VR usage
2. Final product costs less than $800
3. Accurately track hand and finger movement within a tolerance of ± 10mm
4. Document and manage progress of PT that can be accessed by the user
5. Measure user finger flexion and extension with a sensitivity range of ± 5 lbf
6. Measure user finger flexion and extension within a range from 0 lbf to 100 lbf
7. Track physical objects with a minimum refresh rate of 2 Hz to a maximum error of 3 cm in x, y, and z directions
8. Increase difficulty by increasing number of exercise repetitions and resistance force
9. Device improves hand strength and dexterity
10. Headset weighs less than 2.5 lbs.
11. Includes a user's manual
12. VR application includes a user's tutorial

Due to the debilitating nature of strokes, only stroke patients who have undergone initial clinical based therapy are recommended to participate in at-home VR PT treatment. Furthermore, to prevent potential patient injury, the VR Environment will be setup to be performed on a flat surface, ideally a table. Lastly, this PT treatment is meant to be supplemental, not a substitute, for current PT treatment. By using this system, users will have the ability to perform hand exercises at home in a fun and stimulating way, eliminating the need for daily visits to expensive PT clinics.

Design Alternatives

Several important factors must be considered when designing an outpatient, supplemental stroke rehabilitation program as patient improvement is often attributed to the quality of the rehabilitation. Factors are further added when this therapy is supplied through a VR game setting, requiring at minimum a basic understanding of game design and development. With so many different systems expected to function simultaneously, an overlying system block diagram, shown in **Figure 5**, was created that displayed how individual sub-systems would need to communicate: raw user data, data packaging, and therapy software.

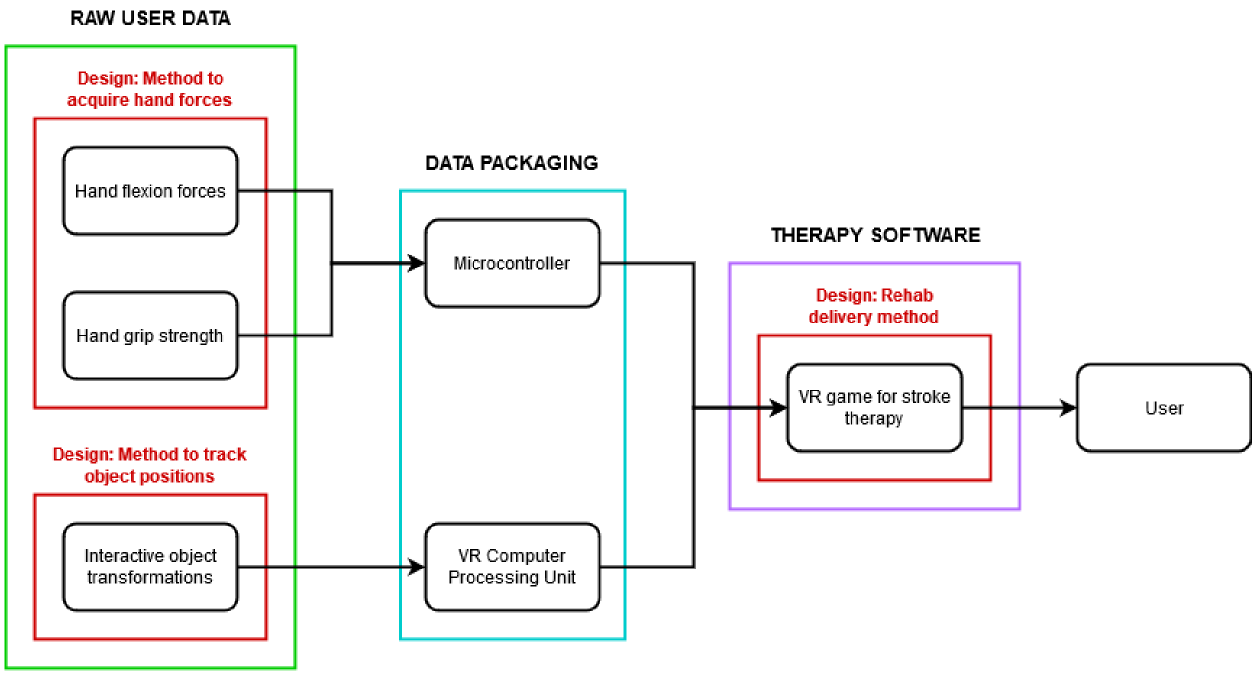

RAW USER DATA
Design: Method to acquire hand forces
Hand flexion forces
Hand grip strength
Design: Method to track object positions
Interactive object transformations
DATA PACKAGING
Microcontroller
VR Computer Processing Unit
THERAPY SOFTWARE
Design: Rehab delivery method
VR game for stroke therapy
User

Figure 5: High-level design flowchart for Denovus stroke therapy

The Raw User Data sub-system contains the user's input or interaction with the device, be that force/flexion readings and object interaction. Second, the Data Packaging sub-system is where the raw user input data is delivered and processed into usable data. Finally, the Therapy Software sub-system encompasses the user's therapy experience and how the predetermined exercises will be displayed. Since specific hardware was associated with each category of the design, some hardware features were determined to be a simple purchase and implementation requiring minimal modifications (microcontroller and VR headset), while other hardware aspects required careful research behind their design selection, such as hand flex and force readings, object detection, and the rehab delivery method (as outlined in red in **Figure 5**).

Method to Acquire Hand Forces

The first design concept analyzed was how to collect and deliver accurate and relevant user hand force and flexion data. This facet of the system was the most important consideration in the design phase as it would fully demonstrate the possibilities of the program. Much research and debate went into choosing between one of the following concepts: minimalist glove with attached sensors, objects with imbedded sensors, or supplied third-party devices for manual data collection. The glove concept would possess a simple design in which multiple sensors would be secured to the front and back, as seen in **Figure 6**. Although this design would accomplish everything needed in the program, the two major drawbacks of this design were the unnecessary complexity of its design and the difficulties of users equipping it.

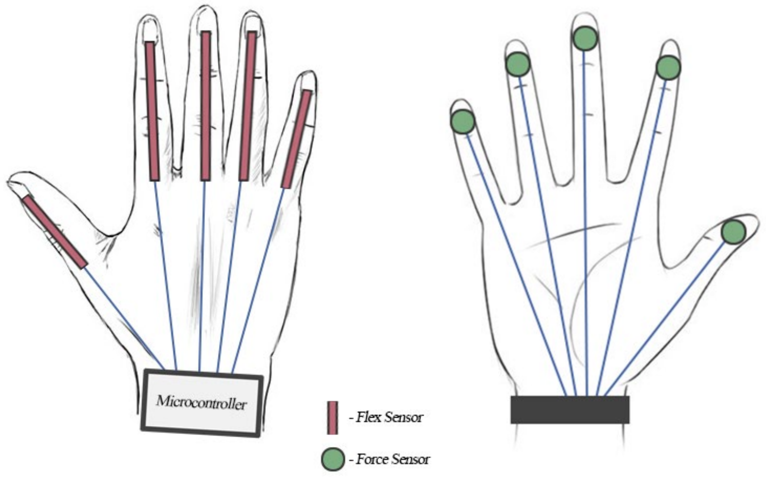


Figure 6: Outline of glove flex and force sensor placement

Alternatively, allowing users to input this data with objects with imbedded sensors in the second option would eliminate both glove concerns but would instead introduce the added difficulties of designing and molding objects, along with requiring users to interact with specific objects, thus creating exercise limitations. Finally, allowing users to monitor this data manually with supplied sensor devices would satisfy the system requirements and specifications, however, this option was determined to be the crudest solution and would sacrifice immersion for simplicity. Ultimately, the specification regarding the scalability of exercise difficulty was an important consideration when deciding how the system was going to receive user input, and thus, the design concept with sensors integrated into a glove allowed for more freedom in potential exercises and the variation of objects used during therapy making it the team's recommended design in collecting user hand force and flexion data.

Method to Track Object Positions

The second design concept was interactive object tracking and input, which determines how a patient interacts with real world objects while maintaining immersion. Users will be performing tasks and exercises that require the handling of various tools, making it important to determine how users will see these tools without removing the VR headset. The proposed ideas for interactive object input were either infrared tracking or visible detection through XR/AR (Augmented Reality). Through IR tracking, points would be attached to objects and their position would be tracked in real-time. Although this method was the most accurate and immersive choice, implementation would be time-consuming and may add delays in data processing. The other option was utilizing augmented reality features and allowing users to see objects while maintaining immersion, as seen below in **Figure 7**.

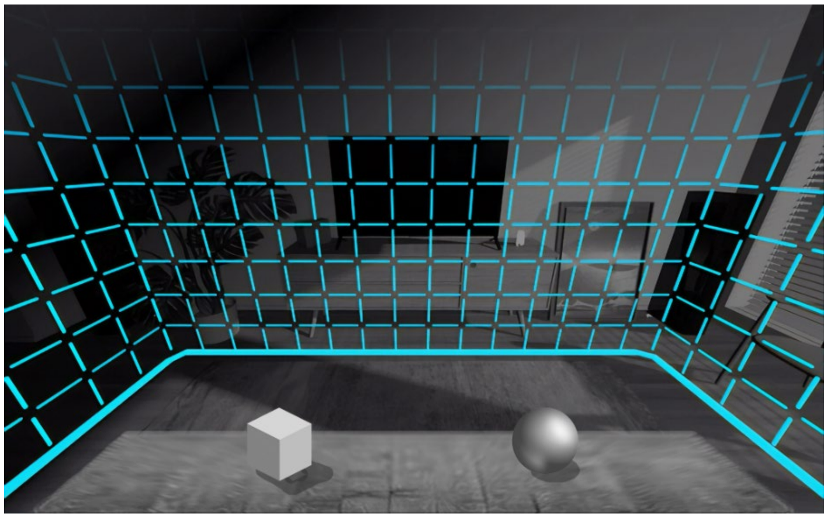


Figure 7: Demonstration of AR object detection

This method would reduce development time and still meet the program requirements but would slightly reduce immersion and only allow for estimating object positions. Although more benefits would come from IR tracking, implementing this method would prove to be unmanageable in the time frame given for this project; as a result, the team decided XR/AR visible tracking would effectively allow users to interact with objects while mostly maintaining full VR immersion. It is important to note that the IR tracking method was not eliminated but rather included as a stretch goal and will be revisited if time allows.

Rehabilitation Delivery Method

The third design concept being analyzed was the rehabilitation delivery method, which determines how the tasks will be executed during treatment. The proposed ideas for VR development were either mini-game tasks or task introduced through role-playing game progression. When comparing each option, the main goal was incorporating a physical therapy session within a maximum time frame of 2 hours as extended VR usage can lead to dizziness and nausea in certain individuals [11]. Implementing a mini-game based therapy mini-sessions would minimize the amount of time needed to complete a full physical therapy session and therefore minimize the amount of time needed to be spent in VR; this option would also much easier for users to understand as options would be laid out in a tile-like format as shown in **Figure 8**.

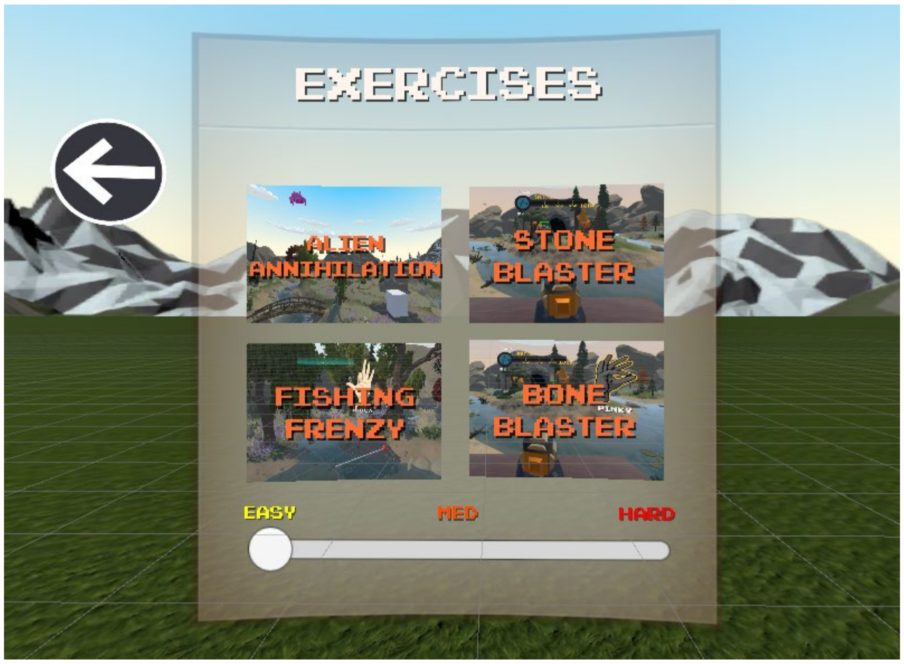


Figure 8. Mine-game Exercise Menu

Research has also shown that the mini-game concept design in similar physical therapy applications has proven successful in maintaining user interest and engagement in the target user demographic age group [6]. However, providing an incentive can be difficult in task-based exercises since users will invest less time into each activity. Alternatively, a story-driven RPG could increase the user's feeling of immersion within the game since the progression of the user's virtual character through the story would be tied to the user's physical strength progression. Users could see their increase in physical strength reflected in the game in ways such as improved physical attributes of their virtual character and increases in exercise difficulty. But due to the complexity of RPG-style games, the option would include user character aspects/stats which are not exercise/therapy related and would therefore prolong the amount of time the user would need to complete a physical therapy session. Furthermore, a story-driven RPG would not have clear treatment stopping points and as such, could confuse the user into believing they have not accomplished enough therapy within a given time-period, or lead them to believe they have leveled up sufficiently enough without completing the desired two-hour treatment session.

Another major consideration in this design concept area is the amount of time needed for development; an RPG requires the creation of a narrative and decision-making processes, which would require more time to develop than exercise focused mini games. Lastly, there are concerns that there would be a lack of appeal for a story-driven RPG in the target user demographic as they might prefer shorter, more condensed therapy sessions. Ultimately, due to the complexity and unappealing nature of RPG games towards the target demographic, the group decided that mini game tasks would be the preferred rehab therapy method and would provide a more condensed therapy session while meeting all listed specifications.

Final Design Solution

Description of Design

The final Denovus therapy supplementary program includes all necessary rehabilitation tools to enable users the comfort of improving their lives at-home. Since the goal was to make a simple and easy-to-learn therapy program, the entire program can be divided into three parts:

1. <u>Sensor Glove</u>
    - Wireless (Bluetooth Connectivity)
    - Easy to Equip, Custom-Sewn Design
    - Safe and Comfortable
    - Accurately Reads 10 Hand Data Points
    - Auto Bluetooth Reconnect Feature
2. <u>VR Headset (VR Program)</u>
    - Oculus Quest Wireless VR Headset
    - Customizable In-Game Table
    - Easy to Navigate UI
    - Calming Natural Environment
    - 4 Unique Exercises
        - Utilize Proven Therapy Techniques
        - Simple, Easy to Learn Game Mechanics
        - Tutorials
        - Encouraging Victory Messages
        - Various Sound Effects, Including a Subtle Nature Ambience
        - Incentivizing Game Elements
        - Realtime Glove Feedback Screen
        - Post-Exercise Glove Data Update
    - Detailed Exercise Data Breakdown via In-Game Menu Option
3. <u>Additional Rehabilitation Tools/Devices</u>
    - 1 Firm Therapy Egg

Each subsystem listed above is broken down into more detail below.

Sensor Glove Overview

Figure 9/Table 1 below shows a detailed breakdown of each component in the Sensor Glove.

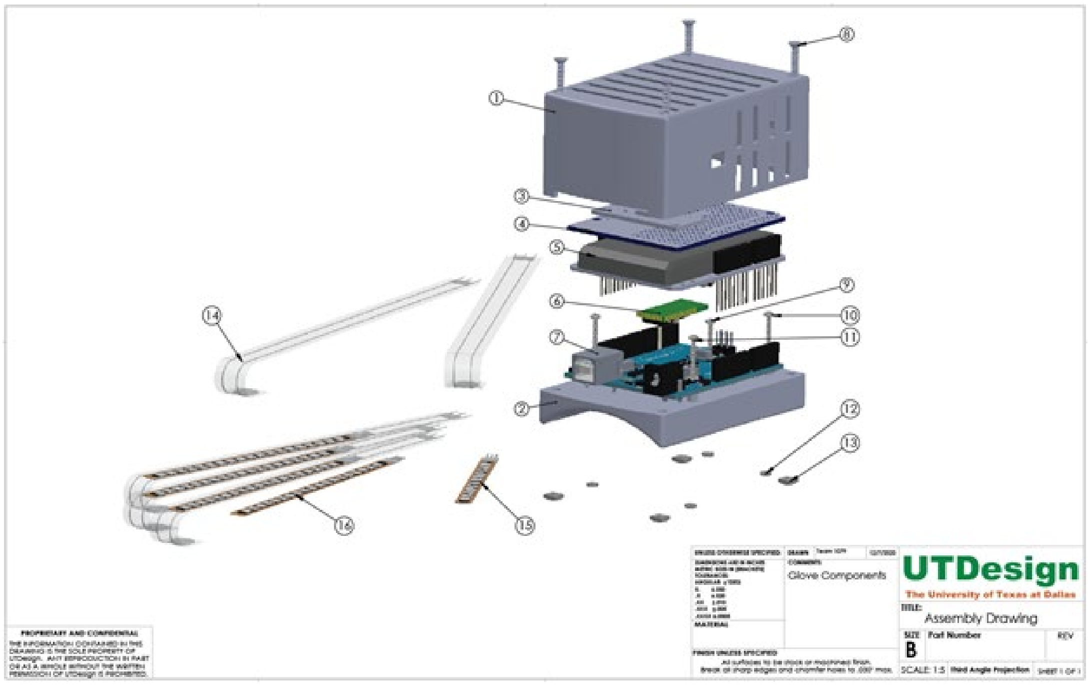


Figure 9. CAD Assembly of Glove Components

Table 1. Glove Components Labels, Descriptions, and Quantities

Item #	Component Description	Qty.
1	3D Printed Controller and Battery Housing (Top)	1
2	3D Printed Controller and Battery Housing (Base)	1
3	(SparkFun) Switch IC Development Tools A/D MUX Breakout CD74HC4067	1
4	(Adafruit) Daughter Cards & OEM Boards Arduino Proto Shield Kit - Stackable R3	1
5	(Nightshade Electronics) energyShield 2 Basic Arduino Battery Supply	1
6	(DSD TECH) HC-06 Wireless Bluetooth Module	1
7	(Arduino) WiFi Development Tools (802.11) UNO WIFI Rev2	1
8	Steel Phillips Flat Head Screw, M3 x 0.5 mm Thread, 50 mm Long	4
9	Steel Pan Head Phillips Screw, M2 x 0.4 mm Thread, 10 mm Long	1
10	Steel Pan Head Phillips Screw, M2 x 0.4 mm Thread, 14 mm Long	1
11	Steel Pan Head Phillips Screw, M2 x 0.4 mm Thread, 16 mm Long	2
12	18-8 Stainless Steel Thin Hex Nut, M2 x 0.4 mm Thread, DIN 439B, ISO 4035	4
13	18-8 Stainless Steel Thin Square Nut, M3 x 0.5 mm Thread	4
14	(Tekscan) FlexiForce Pressure Sensor – 25lbs.	5
15	(Adafruit) Short Flex Sensor	1
16	(Adafruit) Long Flex Sensor	4

The final sensor glove electronic configuration is simplified by the block diagram seen below in **Figure 10**. Looking at **Figure 10**, it shows that there are 13 distinct components (Items # 3,6,7,14-16 on **Table 1**) with 4 different operations integrated into the device.

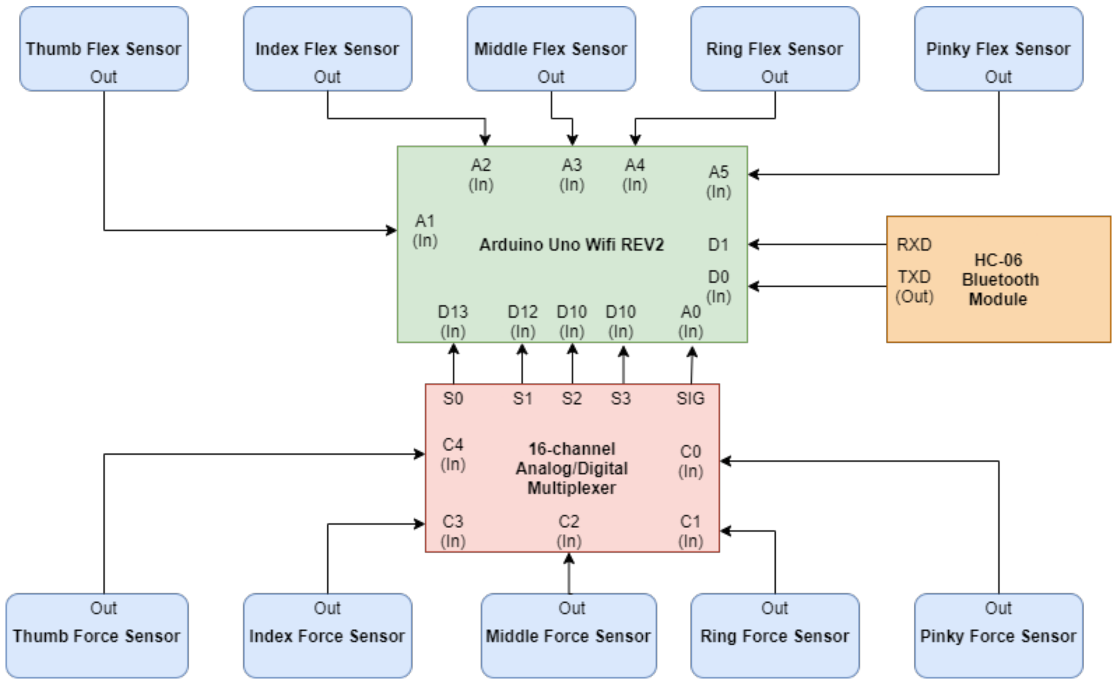


Figure 10. Simplified Electronics Schematic

Inside each Force Sensor (FSR), there are 2 conductive layers separated by a spacer. The two layers consist of active element dots and a semiconductor which when in contact, modulate the resistance of the FSR. The more force that is applied to an FSR causes more of the active element dots to touch the semiconductor which then lowers the resistance of the FSR. The change in resistance yields a measure of the amount of force applied. Because the Arduino Uno WiFi has only 6 analog inputs, a 16-channel multiplexer (MUX) is included in the glove design to incorporate all 10 analog sensors. The S0, S1, S2, and S3 pins on the MUX act as selectors and can connect each analog input to the A0 pin on the Arduino. The A0 analog pin on the Arduino only has one output so at any specific time, it only allows one channel to go out subject to the signal received in terms of binary combination from the microcontroller.

Inside each flex sensor, there is a conductive substrate attached to a flexible base. When the sensor is bent, the conductive layer is lengthened, resulting in an increased resistance which yields a measure of the bend radius.

To communicate glove data wirelessly, an HC-06 Bluetooth module was incorporated into the glove design. Data is received wirelessly from the HC-06 module in the same manner as a serial connection. The TX pin on the HC-06 acts as a transmitter and sends Bluetooth signals out from the module. The RX pin on the HC-06 receives signals.

Now that the sensor glove system overview is complete, each individual system can be further dissected and discussed. The resistance sensors themselves are very simple and require minimal signal conditioning. The following describes the configuration of both the FSR and flex sensors for the sensor glove. Each resistive sensor has 2 pins which are non-polarized. One pin is connected to a voltage divider circuit and the other pin is connected to the Arduino's 5V pin. The voltage divider circuit allows simple measurement of the resistive sensors. As depicted in the diagram in **Figure 11** below, for the voltage divider, one end of the resistive sensor (R1) is connected in series with a 10K Ohm pulldown resistor and V_{out}. In this configuration, the analog voltage reading ranges from 0V to about 5V depending on the sensors value. As the resistance of the sensor decreases, the total resistance across both the resistor and sensor decreases indicating an increase in the current flowing through both resistors and an increase in voltage across the pulldown resistor. The V_{out} for the 5 flex resistive sensors are wired directly to the A1-A5 analog pins on the microcontroller. The V_{out} for the FSR voltage dividers are connected to the C0-C4 pins on the MUX. The VCC pin on the MUX is supplied with 5V from the Arduino.

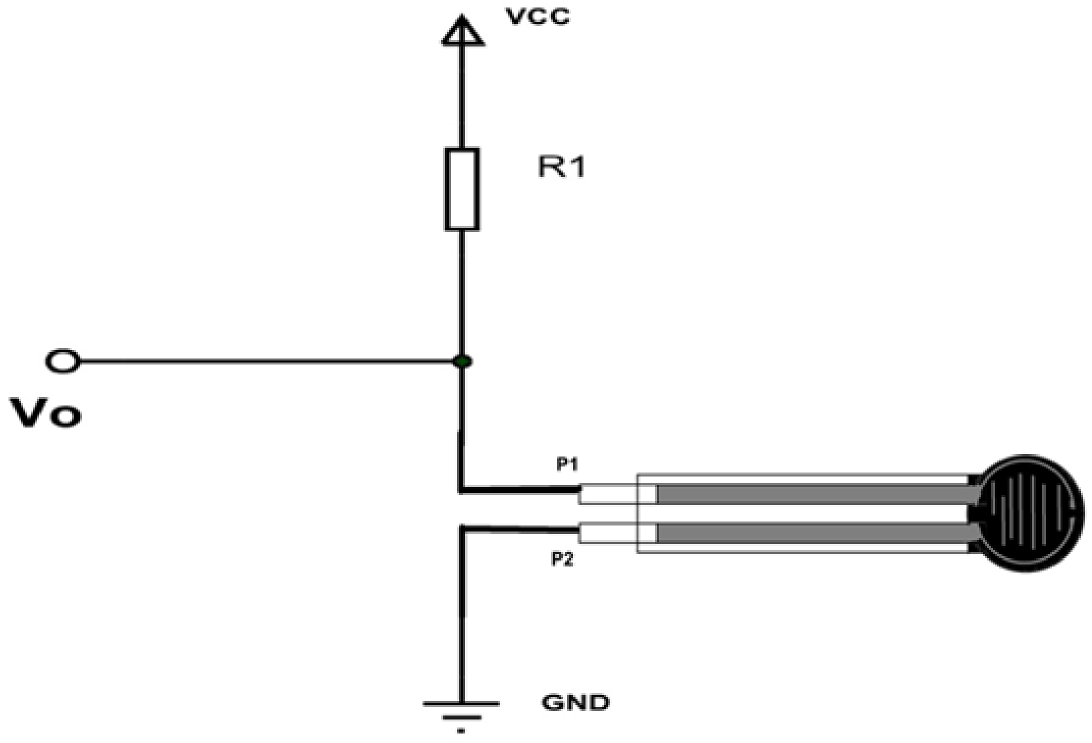


Figure 11. Resistive Voltage Divider Circuit [7]

The other system to be discussed is the HC-06 Bluetooth module. The module has a 3.3V regulator that allows an input voltage in the range of 3.6V to 6V. The HC-06 VCC pin is connected to the 5V pin on the Arduino. The HC-06 RX pin is limited to receiving 3.3V only. This limitation requires a voltage divider to be implemented to connect the RX pin to the 5V pin on the Arduino. A simple voltage divider was created (like that depicted in **Figure 11**) using a 1K Ohm resistor connected in series to a 2K Ohm resistor and the HC-06 RXD pin.

The glove is an open palm design made from five key fabric components, an inner layer, outer cover, assembly housing base cover, straps, and a sleeve. The inner layer runs from the fingertips to midway up the forearm. The fingertips are sewn as small sleeves that cover the top of the finger with a small layer of foam on the bottom. Along the length of the fingers, small channels are sewn onto the inner layer that act as pathways for the sensors and wires to run along. The outer cover slides over the inner layer to hide and protect the wires so that the design looks good and is safe to use. The assembly housing base cover slides over the housing base and is how the base gets sewn to the inner and outer portions of the glove. There are two straps that are secured with Velcro and run across the wrist and palm. They are sewn onto the inner layer along with two metal rings that the straps run through. The last key fabric component is the sleeve which is made from two long cuts of stretchy fabric that are sewn together. The sleeve is attached to the rest of the glove by running the eight assembly bolts through the sleeve, and tightening them, to hold the sleeve in place. The glove is equipped in three simple steps. The user must run their hand through the sleeve, put on the fingertips, and then secure the straps. After the glove is equipped and the VR headset is put on, the user is ready to start exercising.

VR Headset Overview

A high-level overview of the Denovus PT application is given in **Figure 12** below. The software was designed with stroke patients in mind, and thus it is easy to navigate through the application with limited motor functions in one afflicted side of the body. Any interactions the user can be expected to be had with the user interface can be carried out using his or her unimpaired arm and a simple point and pinch gesture with a snap-to-center property enabled on each element of the UI for convenience.

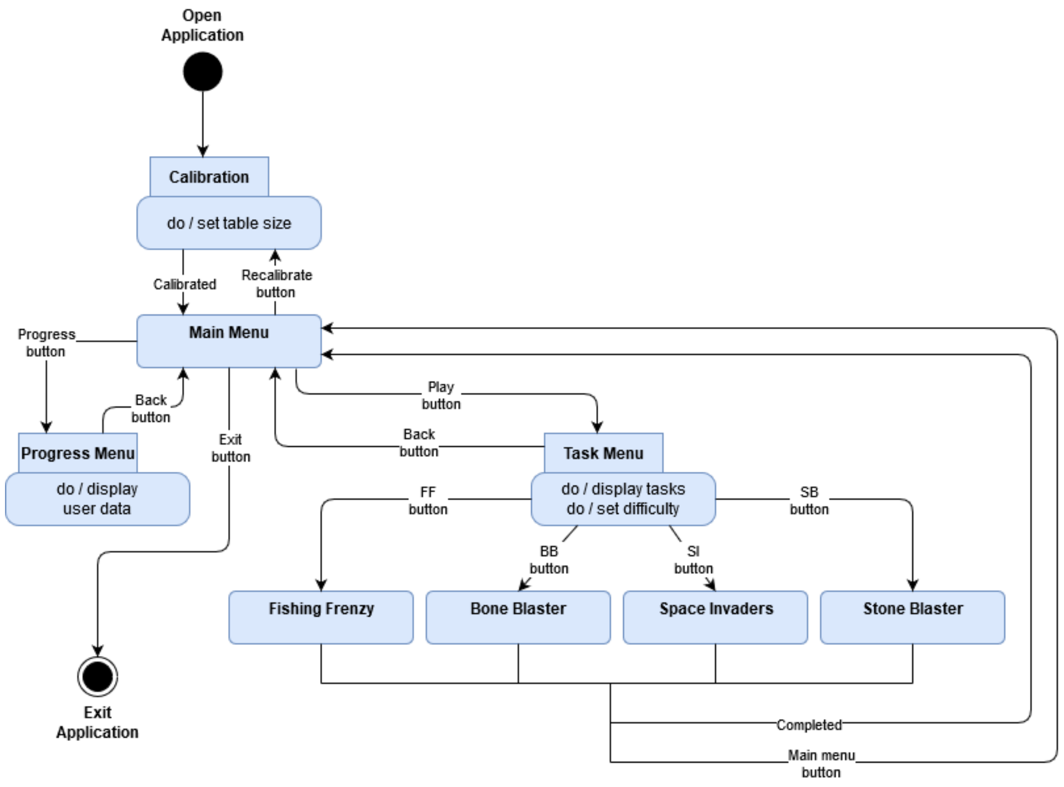


Figure 12. High-level UML State diagram for Denovus application

Upon opening the application, the user is taken to a table calibration phase, in which they follow a series of prompts to bring the table they are sitting at into the virtual world for safety reasons as well as to display various parameters in the exercises (see **Appendix C** for the code behind calibrating the table). Once the table has been calibrated, the main menu appears (**Figure 13**), where the user can access the progress menu and the task menu through their corresponding buttons. From the task menu, the user can manually set the desired difficulty for the exercises and load into each of the four exercises. Difficulty is left to the discretion of the user to grant them a degree of control and sense of independence regarding their own rehabilitation.

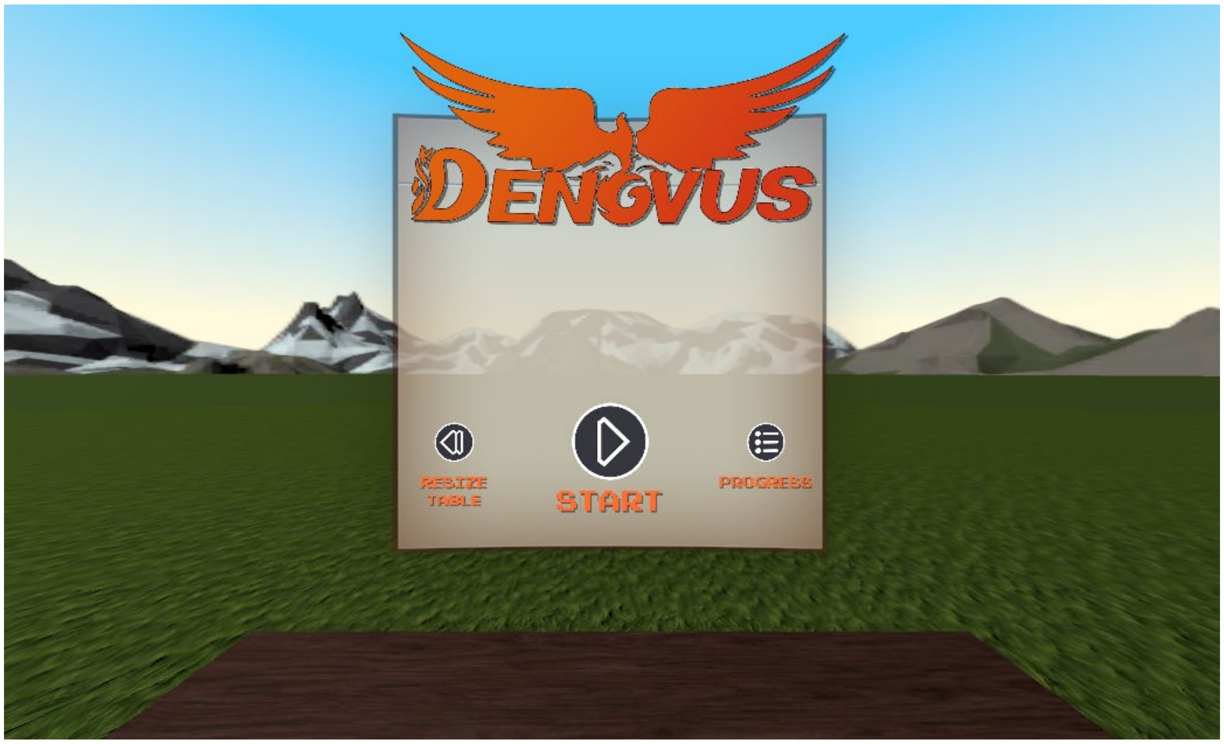


Figure 13. In-game screenshot of main menu UI

The minigames featured within Denovus PT were modeled after existing recommended hand exercises for subacute stroke patients. In particular, the ball grip exercise, where a user squeezes a resistive strength training ball as hard as possible, and finger pinch/opposition, where a user pinches each digit of their hand to the thumb, were utilized. These exercises appear in a multitude of early strength training tasks given to stroke patients to begin the rehabilitation process [6]. For the most part, these hand motions show the greatest increase in a stroke patients' ability to perform Activities of Daily Living (ADL) and serve as a foundation for future fine motor skill exercises. Since the *basics* are crucial to hand rehabilitation long term, these hand functions are the first to be implemented in the Denovus VR PT. Furthermore, the gamification of exercises in this *mini-game* fashion essentially transforms repetitive finger motions into tasks and goals a patient must perform to successfully accomplish a desired outcome. The presence of these tasks and goals has been shown to increase a stroke patient's functional ability which in turn increases their ability to perform activities of daily living. [15] An increase in a stroke patient's ability to perform ADL also increases their motivation to continue therapy, creating a positive feedback loop of successful rehabilitation. All Denovus mini-game exercises are intended to increase the patients hand strength and dexterity to improve their hand functionality. An increase in pinch strength and hand power in the affected hand is directly correlated with an increased independence of the patient, as they are better able to perform activities of daily living [17]. Furthermore, research shows that in all post stroke patient types, subacute, chronic, and post-chronic, an increase in grip and pinch strength resulted in an increase of Activities of Daily Living functionality [17]

Exercise 1 – Stone Blaster

Stone Blaster is the first of these types of games and has the user equip a stress ball for the entirety of the exercise. The user must squeeze the ball in their palm to destroy oncoming enemies, when prompted, in the allotted time (see **Figure 14** below). Extra time is granted upon the successful destruction of an enemy to give the user a sense of challenge and success upon completion of the exercise with a new high score of either "enemies killed" or "finger force readings". The presence of a challenging exercise as well as the mini-games repetition and duration have been shown to increase motivation and recovery rates among stroke patients [16]. See **Appendix D** for relevant source code and **Figure 15** for a more detailed breakdown of the exercise's UML structure.

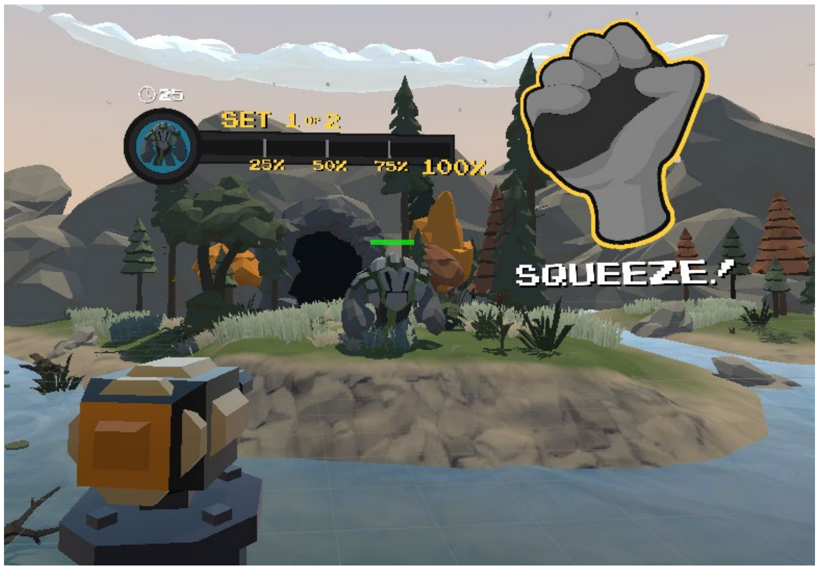


Figure 14. In-game screenshot of Stone Blaster exercise

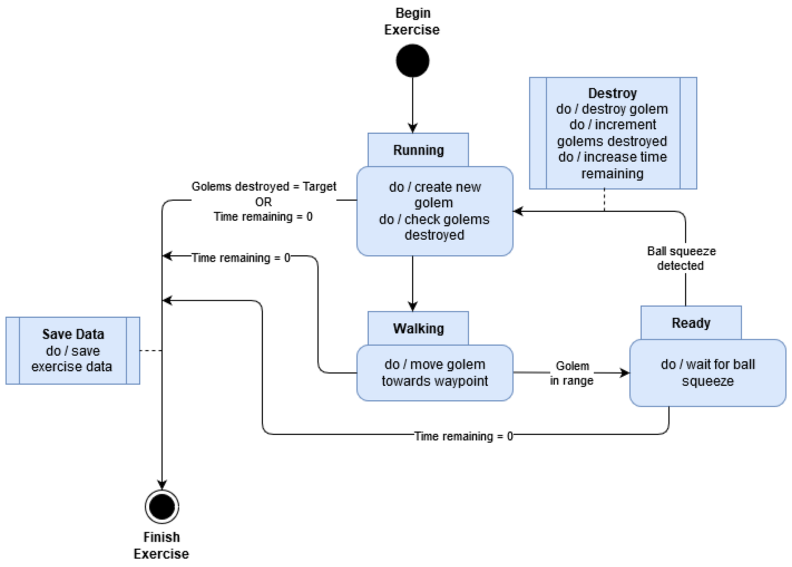


Figure 15. UML State diagram for Stone Blaster exercise

Exercise 2 – Bone Blaster

Bone Blaster follows a similar gameplay layout to Bone Blaster where time is allotted to the user to destroy certain amounts of enemies per wave by pinching when prompted (**Figure 16**). This exercise, however, increases difficulty and intensity in the sense that a Boss round appears randomly where the user must perform an action twice in a row to successfully fire a shot. The presence of intensity not only helps rehabilitation rates, but it also helps with a patient's motivation as they feel as though they are actively pushing towards rehabilitation. This exercise primarily makes use of the finger opposition exercise with random finger oppositions being shown to the user to create a sense of diversity and modulating difficulty. See **Appendix D** for relevant source code and **Figure 17** for a more detailed breakdown of the exercise's UML structure.

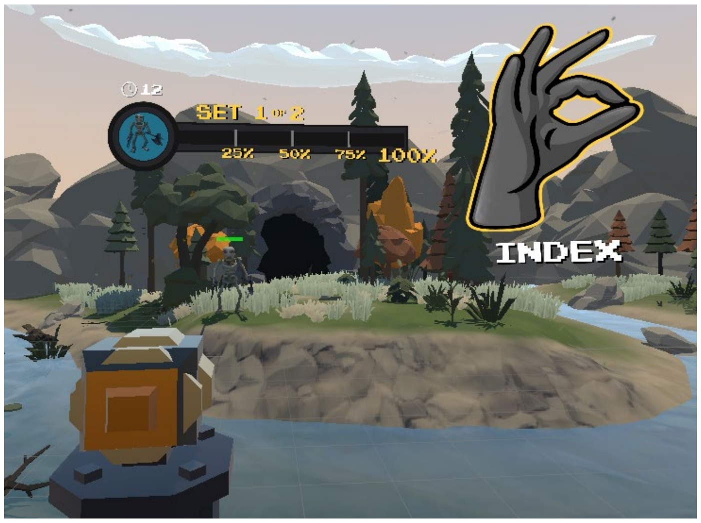


Figure 16. In-game screenshot of Bone Blaster exercise

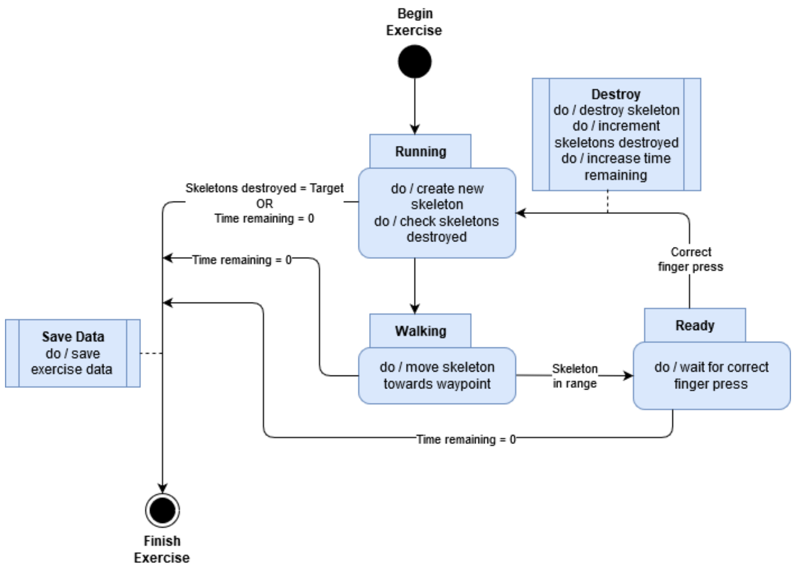

Begin Exercise
Running
do / create new skeleton
do / check skeletons destroyed
Destroy
do / destroy skeleton
do / increment skeletons destroyed
do / increase time remaining
Skeletons destroyed = Target OR Time remaining = 0
Time remaining = 0
Correct finger press
Walking
do / move skeleton towards waypoint
Skeleton in range
Ready
do / wait for correct finger press
Save Data
do / save exercise data
Time remaining = 0
Finish Exercise

Figure 17. UML State diagram for Bone Blaster exercise

Exercise 3 – Alien Annihilation

Alien Annihilation is an ode to the classic video game, *Space Invaders*. In this rendition, patients must aim at the invader they want to destroy and perform that invaders specific destroy pinch (**Figure 18**). This game mode is the most advanced of the four mini games included with the set as it combines repetitive strength training finger oppositions with lateral shoulder and wrist movement topped off with a cognitive matching game. The seamless integration of these game functions further helps to rehabilitate the stroke patient user as intensity and cognitive flexibility are a necessity for successful completion of this game. See **Appendix E** for relevant source code and **Figure 19** for a more detailed breakdown of the exercise's UML structure.

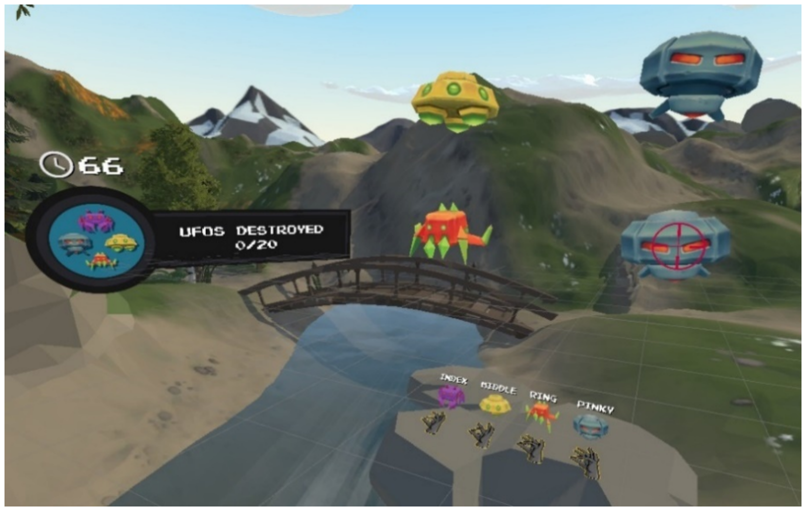


Figure 18. In-game screenshot of Alien Annihilation exercise

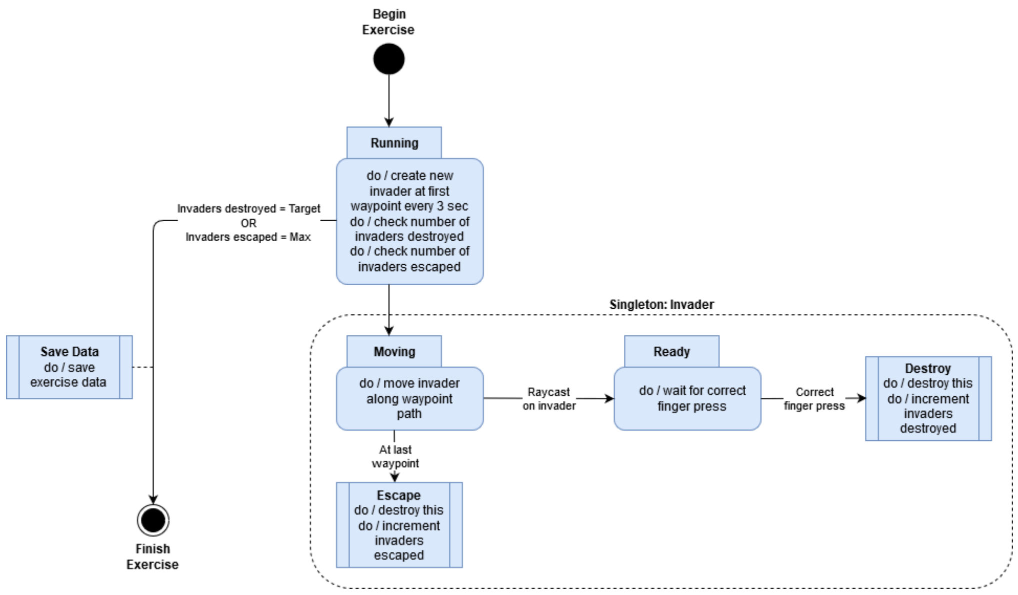

Begin Exercise
Running
do / create new invader at first waypoint every 3 sec
do / check number of invaders destroyed
do / check number of invaders escaped
Invaders destroyed = Target OR Invaders escaped = Max
Singleton: Invader
Save Data
do / save exercise data
Moving
do / move invader along waypoint path
Raycast on invader
Ready
do / wait for correct finger press
Correct finger press
Destroy
do / destroy this
do / increment invaders destroyed
At last waypoint
Escape
do / destroy this
do / increment invaders escaped
Finish Exercise

Figure 19. UML State diagram for Alien Annihilation exercise

Exercise 4 – Fishing Frenzy

Fishing Frenzy is the fourth and final mini game included with Denovus VR PT and switches main exercise functionality from finger/grip strength to dexterity and finger range of motion. In this game the user must match a specific finger flexion to reel the fish (**Figure 20**). The main progress tracker in this game is time to fish as users are encouraged to flex their fingers to reach the required flexion cap as fast as possible, increasing range of motion as well as speed of hand function. Finger Flexion is better than Finger Extension because it shows higher muscular activity in both flexor and extensor musculature of forearm [17]. See **Appendix F** for relevant source code and **Figure 21** for a more detailed breakdown of the exercise's UML structure.

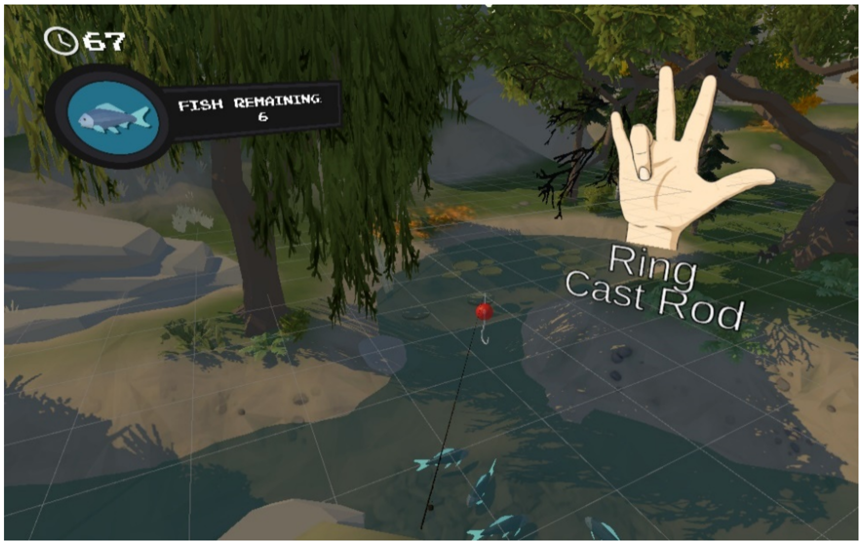


Figure 20. In-game screenshot of Fishing Frenzy exercise

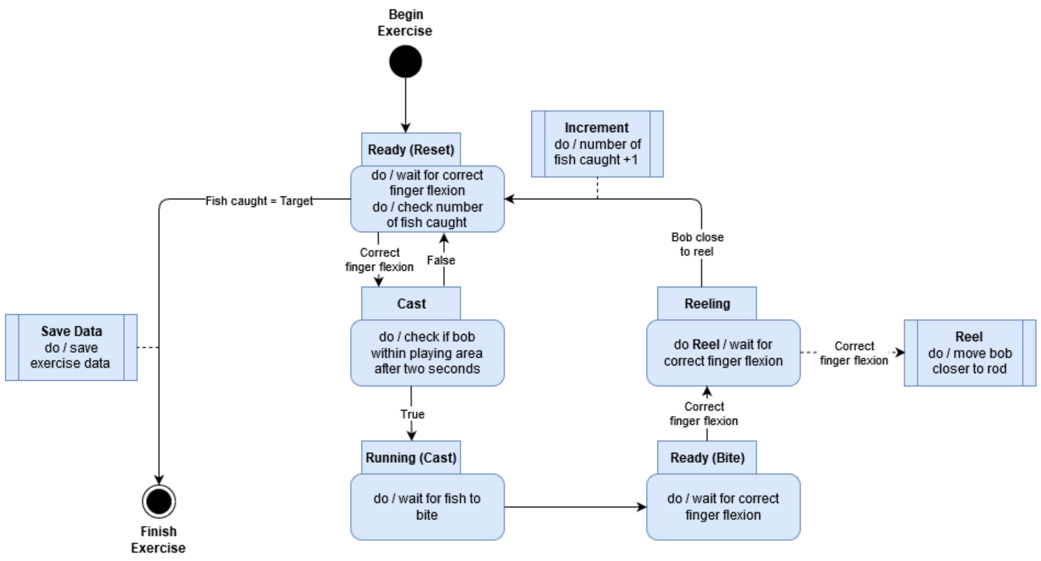
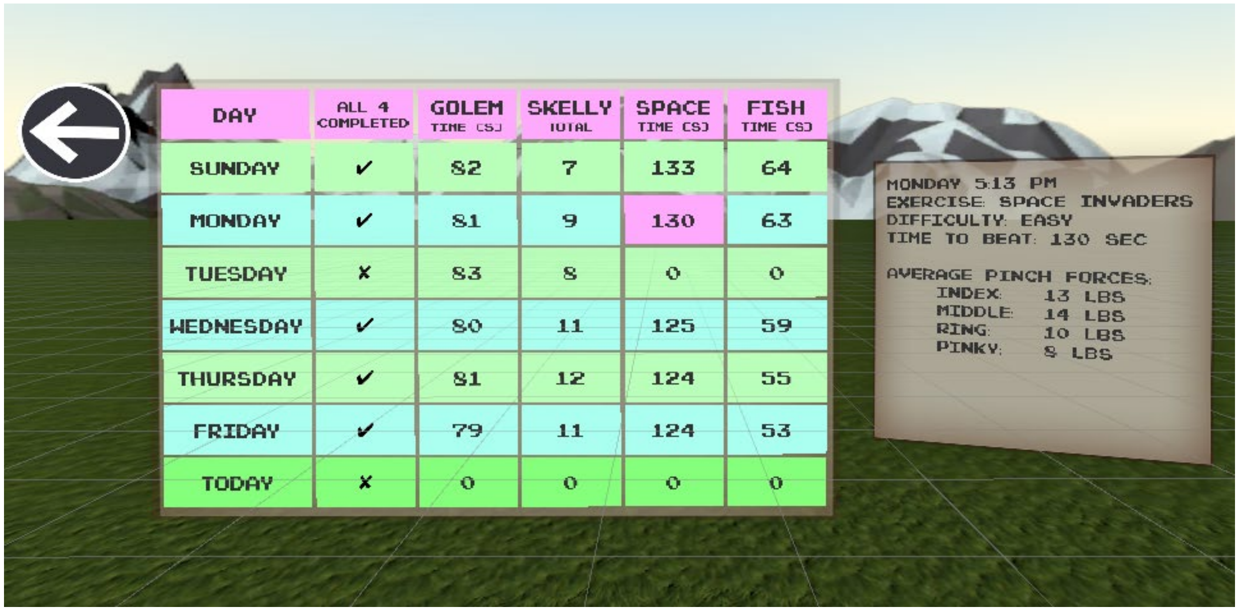

Begin Exercise
Ready (Reset)
do / wait for correct finger flexion
do / check number of fish caught
Increment
do / number of fish caught +1
Fish caught = Target
Bob close to reel
Correct finger flexion
False
Cast
do / check if bob within playing area after two seconds
True
Running (Cast)
do / wait for fish to bite
Ready (Bite)
do / wait for correct finger flexion
Correct finger flexion
Reeling
do Reel / wait for correct finger flexion
Correct finger flexion
Reel
do / move bob closer to rod
Save Data
do / save exercise data
Finish Exercise

Figure 21. UML State diagram for Fishing Frenzy exercise

Exercise Save Data

Upon selecting the "Progress" button from the Main Menu, the user will be presented with a previous Save Data menu (**Figure 22**). The sequence diagram depicted in **Figure 23** demonstrates how the interaction between several processes and layers within the Denovus application stores a user's exercise data from each workout session. A BluetoothManager wrapper class handles connection to the Arduino on app startup and sets the refresh rate at which sensor values are accessed and updated. Every x milliseconds, the BluetoothManager script grabs the last line from the Arduino sensor feed and converts it into a size 10 array of sensor values, where indices 0 to 4 denote force sensor readings from the thumb to the pinky, respectively. Similarly, indices 5 to 9 in the array denote flex sensor readings from the thumb to the pinky. If an exercise is completed, then a session-level SaveManager class will create a Serializable object with several parameters pertaining to the unique exercise, including date and time, time to complete, score, and any glove sensor values of interest. This object is serialized using the ByteStream encoder and stored in another Serializable array of user save data for each of the four exercises, which are stored permanently on the Oculus device. The progress menu accesses and decodes these arrays and displays the information in a viewable format for the user. See **Appendix G** for relevant source code.



DAY	ALL 4 COMPLETED	GOLEM TIME (S)	SKELLY TOTAL	SPACE TIME (S)	FISH TIME (S)
SUNDAY	✓	82	7	133	64
MONDAY	✓	81	9	130	63
TUESDAY	✗	83	8	0	0
WEDNESDAY	✓	80	11	125	59
THURSDAY	✓	81	12	124	55
FRIDAY	✓	79	11	124	53
TODAY	✗	0	0	0	0

MONDAY 5:13 PM
EXERCISE: SPACE INVADERS
DIFFICULTY: EASY
TIME TO BEAT: 130 SEC

AVERAGE PINCH FORCES:

INDEX:	13 LBS
MIDDLE:	14 LBS
RING:	10 LBS
PINKY:	8 LBS

Figure 22. In-game screenshot of progress menu UI

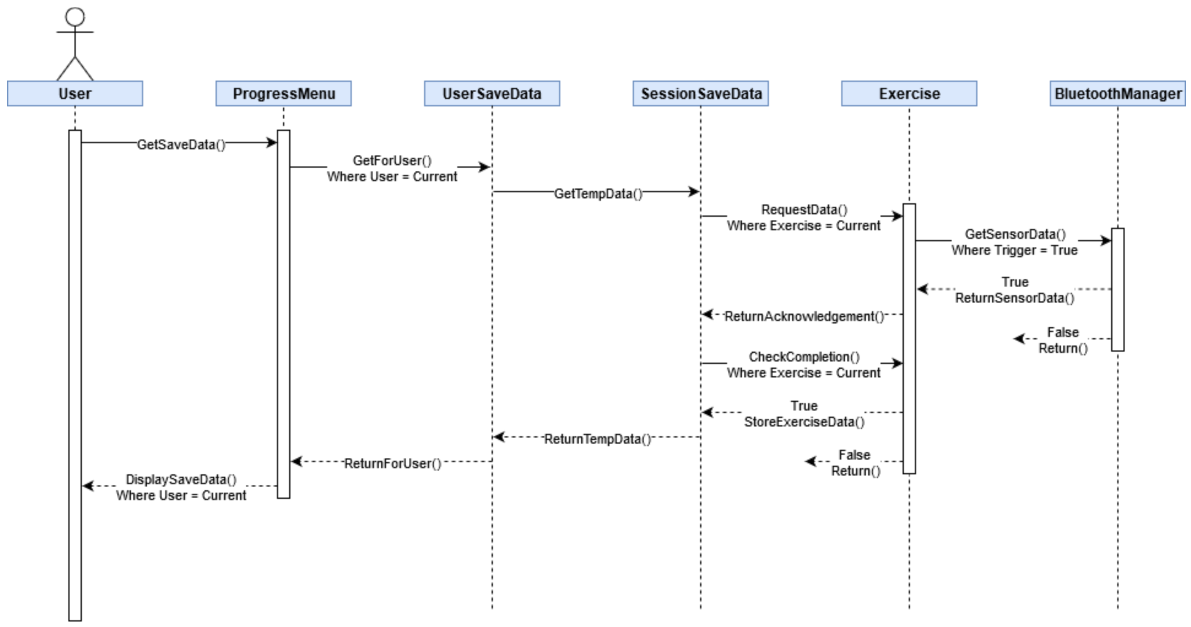


Figure 23. UML Sequence diagram for accessing user exercise data

Design Justification

User Data Capture

After reviewing the way patients would interact with sensor objects it became clear that a stroke patient would not want to continually turn on Augmented reality to find the sensor object after each exercise. An additional reason for wanting a sensor glove is because a patient would have to keep track of all these sensor objects to initiate the exercises, and stroke patients already feel agitated while doing exercises, so this would only add to their frustration. After calculating how many sensors would be needed for either the objects or the glove it was determined 15 sensors would be needed for the objects, increasing the overall size and complexity, while only 10 would be needed for the glove. Having an easy to equip and comfortable sensor glove was the obvious choice after looking at these individual aspects of design.

Object Tracking

The choice of XR/AR (augmented reality) tracking of objects was chosen after realizing the cost of buying objects that have Infrared Tracking on their exteriors. After researching the Oculus's camera settings, it was found out that the Oculus can switch to augmented reality by simply double tapping the side of the Oculus itself. It was also determined that other VR therapy competitors use overly complex, confusing, and expensive VR methods that often result in increased user frustration (**Table 6/Appendix H**).

Exercise Delivery

The decision to go with Mini games was based off the research done to determine how hard it is to create mini games or Role-playing Games using Unity. Mini Games were determined to be hard to design without having years of experience using Unity, but still achievable with the correct tutorials. After researching previous designers who created gaming code it was found that role playing games take several years to develop. And after spending time debating both types of games it was also decided that role playing games would also be overwhelming for an older demographic as larger portion of older gamers enjoy small, simple "mobile" style video games (**Figure 24**).

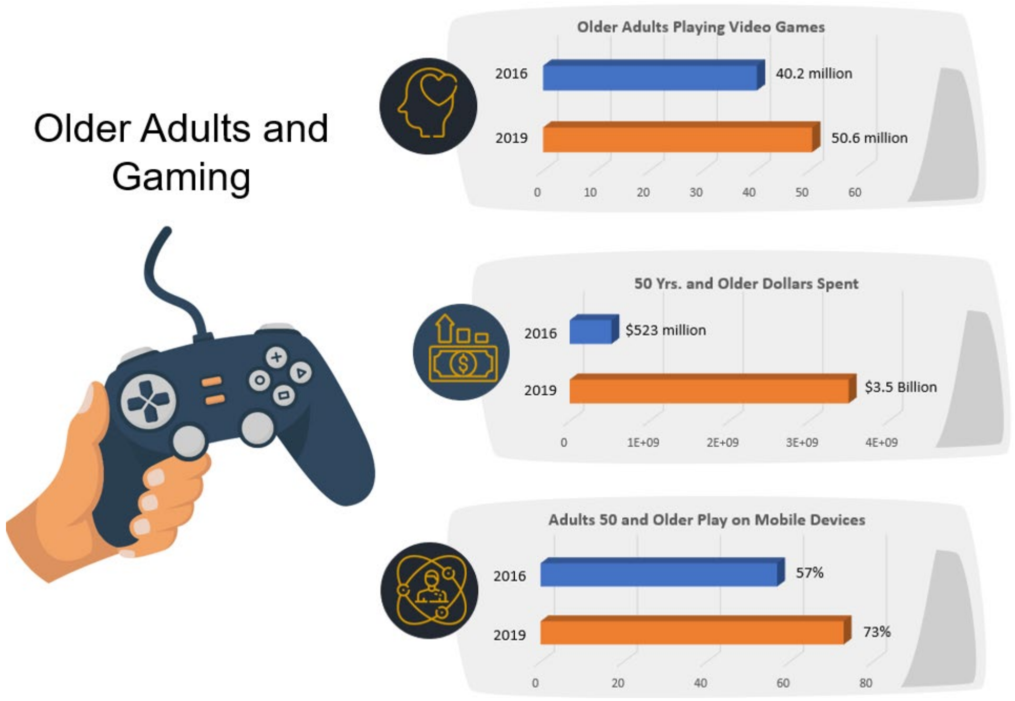


Figure 24. Figure demonstrating the number of adults over the age of 50 that play video games, how much money they spent during a six-month period, and what percentage of adult gamers play on mobile devices.[7]

Glove Connection to Headset

The decision to use Bluetooth for the glove and headset connectivity was determined after our components were found to be incompatible with Wi-Fi. After extensive research, it was found that connecting the glove to the headset with a wired connection would allow for proper data transfer, however, it was determined that this option might cause interference with hand tracking. Connectivity through USB wiring might also cause the headset to be abruptly thrust towards the glove if caught on the edge of the table. After experimenting with several different forms of Bluetooth code and fixing several errors in lines of code, a connection via Bluetooth was achieved and found to be sufficiently accurate.

Glove Designs

The housing unit used to collect all the electrical components was first placed on the back of the user's hand, outside of the glove. This pretotype housing unit design was taped to the hand at first and was discovered to be obstructing the cameras view of the hand which prevented the oculus from tracking the hand. A second pretotype of the housing unit was made to go just below the wrist, but this housing unit was wavering because the base of the design did not conform to the user's wrist structure. A third prototype was made with vents to allow heat expulsion and was also found to accurately fit the users upper arm area. Upon completion of the final prototype, it was also found that the finger holder portion of the glove was too loose and slipping off during exercises. It was decided to extend the length of the finger holders, but after discussing this further it would have made the design look incohesive. The team then decided making the strap on the wrist a sleeve could help with the slipping issue along with several other issues. The glove seemed to fit uncomfortably on the wrist and the housing unit was slightly shaking during the exercises and changing the strap to a sleeve would fix these issues. After implementation of the sleeve the glove was free of shaking or slipping. Upon completing the final prototype and sliding the housing unit down the forearm to fit the sleeve, the glove was found to be significantly more comfortable.

Sensor Selection

The first FSRs purchased (Adafruit FSR) were able to sufficiently read all ranges of values a user might input during the therapy. However, preliminary testing determined an inconsistency with repeated applied forces in that read forces would vary from press to press (**Figure 25**).

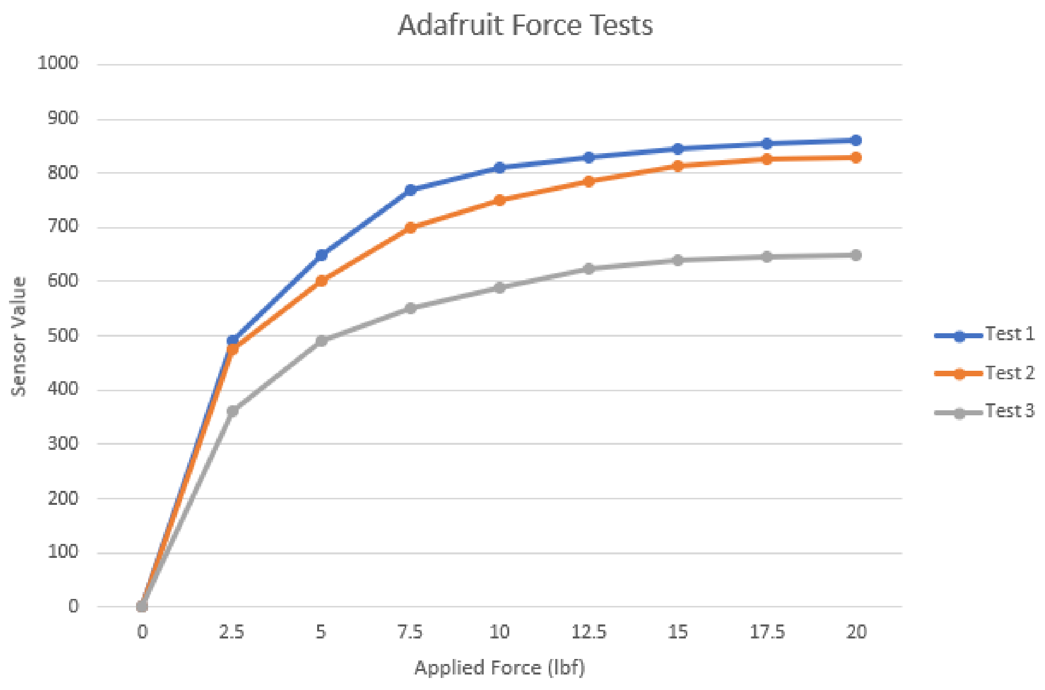


Figure 25. Preliminary test of Adafruit FSR

The accuracy of the sensors was also an important factor as these values would be displayed post-session. The force pucks used in **Figure 26** were used to test the accuracy of the sensors directly (per program specifications), however, another experiment was required to relate actual values in-glove to actual forces. As discussed in more detail later in ***Verification Methods and Results Discussion***, **Figure 28** shows an experiment performed to determine the consistency and accuracy of user finger pinch and grip forces. A simple weight-plate was designed then 3D printed to focus each weight's center of gravity on the FSR. Although this specific experiment yielded varied and relatively uncontrolled results, the intention was to accurately simulate a user's press while wearing the glove. And since a user's press will vary from press to press, this experiment was repeated ~20 times and the average results were used to create **Figure 27**'s plot.

Power Supply

The battery included within the controller assembly has a specified maximum battery duration of 14 hours (under specific conditions). Since the maximum session duration would end up at around 25-30 minutes, the group felt this battery life would be more than enough to power the glove. And due to this long power duration, the group also felt an increase in glove functions, at the cost of increased battery usage, would be beneficial. Seen below in **Table 2**, the battery life with and Arduino update at every 1000ms (1 sec) resulted in an average battery life of 8-10 hours, and that by increasing this update to every 10ms and receiving 100 times the data, the battery life only reduced to 4-6 hours.

Table 2. Table showing battery life of glove at increasing Arduino code update rates

Battery % Remaining per Arduino Update				
Update Rate	**2hr**	**4hr**	**6hr**	**8hr**
1000ms	85%	62%	43%	23%
500ms	81%	58%	39%	21%
100ms	69%	41%	22%	5%
10ms	61%	25%	2%	-

Standards

IEC-60601-1 – Medical Electrical Equipment:

- Due to the location of the electrical components/power supplies contained within the controller housing in the sensor glove that is to be equipped onto user's appendages (Type BF – body floating), certain design measures were considered to ensure proper isolation, insultation, and clearance. The listed requirements located within the standard were met through the following means:

- Isolation (3000 Vac): The 5V power supply supplied by Lithium-Ion Battery is self-contained within the energyShield 2 Power Supply device, which features an on/off switch and a durable, injection molded cover to prevent any damage to the lithium polymer cell.
- Insultation (Double): The electrical components within the housing are separated from the user's appendage by three layers: the PLA 3D printed base, a foam insert between the base and fabric, and a double-layer of neoprene fabric wrapping the previous two layers.
- Clearance (5 mm): The insulated housing base measures at 10.5 mm.

Each of these restrictions were considered when designing the location and size of the controller housing. Originally, the controller housing assembly and the user's appendage were only separated by fabric and a foam insert, failing the minimum 5mm clearance standard. The final pretotype was redesigned and the 3D printed base was added, thus passing all listed standards.

FDA:

- Sec. 890.1575 – Force-measuring platform (Class I, general controls)

  (a) *Identification.* A force-measuring platform is a device intended for medical purposes that converts pressure applied upon a planar surface into analog mechanical or electrical signals. This device is used to determine ground reaction force, centers of percussion, centers of torque, and their variations in both magnitude and direction with time.

  (b) *Classification.* Class I (general controls). The device is exempt from the premarket notification procedures in subpart E of part 807 of this chapter, subject to the limitations in § 890.9.

- Sec. 890.1925 – Isokinetic testing and evaluation system

  (a) *Identification.* An isokinetic testing and evaluation system is a rehabilitative exercise device intended for medical purposes, such as to measure, evaluate, and increase the strength of muscles and the range of motion of joints.

  (b) *Classification.* Class II (special controls). The device is exempt from the premarket notification procedures in subpart E of part 807 of this chapter subject to § 890.9.

Safety and Reliability

As discussed in the ***Standards*** sub-section, certain design precautions were taken with the location of the controller housing on the glove. By following the recognized guidelines listed in IEC-60601-1, no further safety precautions were required when designing the sensor glove. For questions on the safety of the VR headset, please refer to the Oculus Health & Safety Warnings document located within the supplied Final Project Documentation Package.

Design Validation

Acceptance Test Plan

Table 3 below outlines the project's specifications, how each was tested, and the result of the tests.

Table 3: Compliance Matrix for program specifications

Compliance Matrix						
Ref.	**Final Specification**	**Acceptable Outcome**	**Verification Method**	**Test Result**	**Pass/ Fail**	**Notes**
1.1	Physical therapy session takes less than 2 hours to complete	Session time less than 2 hours	Built-in timer that monitors session times	All 4 games were played in less than 15 mins on easy and less than 25 mins on hard	Pass	Built-in timer & external stopwatch both under 2 hours
2.1	Product costs less than $800	Final product manufacturing cost less than $800	Bill of Materials Breakdown	After only calculating the items used the final cost is $798.44	Pass	See BOM
3.1	Product documents and manages progress of Physical Therapy that can be accessed by the user	Application tracks and saves exercise data for user access	Save session data and allow user access through application	After playing each game the statistics do load automatically to the designated progress tracking table	Pass	Click Progress button on main menu
4.1	Product measures finger flexion and extension within range of 0 lbf and 100 lbf	Glove sensors read all forces < 100 lbf	Experiments using a range of validated weights	Each sensor accurately displayed the correct weight when using the force puck test	Pass	Test repeated for sensors in and out of glove
4.2	Product measures finger flexion and extension within sensitivity range of ± 5 lbf	Glove sensors read forces with a tolerance of ± 5 lbf	Experiments using a validated weight	Test results showed an error below 5% per sensor (± 5 lbf)	Pass	± 5 lbf of 100 lbf total is 5% per sensor

5.1	Product tracks hand and finger movement within tolerance ± 10mm	Hand and finger tracking error is less than 10mm	VR positional error test using printed grid and ruler	After measuring the hand position in and out of VR, the variance between each was under 10mm	Pass	N/A
5.2	Product tracks physical objects with refresh rate of 2 Hz and error of 3 cm in x, y, z directions	Minimal lag and good fidelity on physical object tracking	Unity Profiler and VR positional error test using ruler	Dependent on the results of 5.1	Pass	N/A
6.1	Increase difficulty by increasing number of exercise repetitions and resistance force	Difficulty of exercises scales based on in-game benchmarks or glove sensor readings	Verify increase in difficulty of exercise following user improvement	After changing difficulty before game, the game does increase repetitions or extend exercise time	Pass	Difficulty option is shown during choice of game
6.2	Device improves hand strength and dexterity	Users with issues with hand grip strength and flexion due to upper limb hemiparesis see any improvement	Periodic hand strength tests over the course of therapy	After several team members finished testing exercises, they felt sore and fatigued	Pass	Substitute test performed due to current climate
7.1	Headset weighs less than 2.5 lbs.	Headset weight < 2.5 lbs.	Weigh on two different scales	After weighing headset, the average weight is 1.26 lbs.	Pass	N/A
8.1	Includes a user's manual	An effective user's manual is provided	User's manual is provided with program	User manual was printed, stapled and then examined	Pass	N/A
8.2	VR application includes a user's tutorial	An effective VR tutorial is provided within the application	Tutorial plays upon application startup	Upon starting each exercise, a tutorial is shown automatically	Pass	Further instructions in User's Manual

Verification Methods and Results Discussion

<u>Session time limits and tracking</u>

1.1 (T) Physical therapy session takes less than 2 hours to complete:

Time restraint can be verified through testing. A timer will be displayed during each game session in the corner of the screen. Each individual game will last under seven minutes, so the maximum time spent for one session (three exercises) will be under 2 hours.

For testing, equip and turn on the sensor-glove, and equip the Oculus headset. After application launch and calibrations, start Exercise 1 and take the maximum allotted time allowed (seven minutes). After receiving the "time-exceeded" notification, return to the main menu and repeat the previous procedure for Exercises 2 and 3. Document the final on-screen total time. Final on-screen session time should be less than 2 hours.

<u>Discussion of Results:</u> Times were recorded externally (stopwatch) for an entire play session (all four exercises), each play session at a different difficulty. The final recorded time for a session on Easy difficulty was at 15 minutes, Medium at 20 minutes, and Hard at just under 25 minutes. Although the final times are well under the max time, this time was chosen as a limit to reduce user nausea and physical fatigue rather than an estimated time it should take to complete a full session.

<u>Product manufacturing costs</u>

2.1 (I) Product costs less than $800:

Visual examination/inspection of the final cost of materials will provide evidence that the product will not exceed $800 in manufacturing costs. Several items on the BOM are purchased in bulk quantities; certain costs will need to be divided down to get an accurate manufacturing estimate for one product.

<u>Discussion of Results:</u> See Bill of Materials, **Table 3**. As also discussed in Conclusions and Recommendations, this final value could have been reduced by another $100 by replacing the Oculus Quest headset with the newer Oculus Quest 2. However, to avoid complications mid-project it was decided to keep the original Quest.

<u>Session data capture, storage, and access by user</u>

3.1 (D) Document and manage progress of PT that can be accessed by the user:

Check the Progress data following each run of the exercises for multiple runs. The data should be easily accessible and written to through code. Following an instance of an exercise, the user should have the option to show their stats, which presents the data in a clean readable format.

For demonstration, equip and turn on the sensor-glove, and equip the Oculus headset. After application launch and calibrations, select the “Progress” button from the Main Menu and document the current data listed. Next, return to the Main Menu, select Exercise 1, and finish the exercise to completion. After completion, return to the “Progress” screen and document the change in data. Finally, close then re-open the application, return to the “Progress” screen, and document the data. Final data should track and store all current and previous exercise progress.

Discussion of Results: As shown previously in **Figure 22**, users can access all previous exercise data via an in-game menu option. **Appendix G** also covers Glove Save Data.

Glove data accuracy, ranges, and tolerances

Sensor glove data accuracy was verified by testing with validated weights (marked 2.5 and 20 lbs. weight-lifting plates). Data was obtained through static weight placement (not equipping glove) and checking Arduino Serial Monitor. Weights were evenly distributed across the sensor’s surface by using a 3D Printed Force Puck (**Figure 26**). The 3D Printed Force Puck will be 3D-printed by Nathan Barton and the marked 2.5 and 20 lbs. weight-lifting weights are owned by various team members. See **Appendix B** for full Test Plan Breakdown.

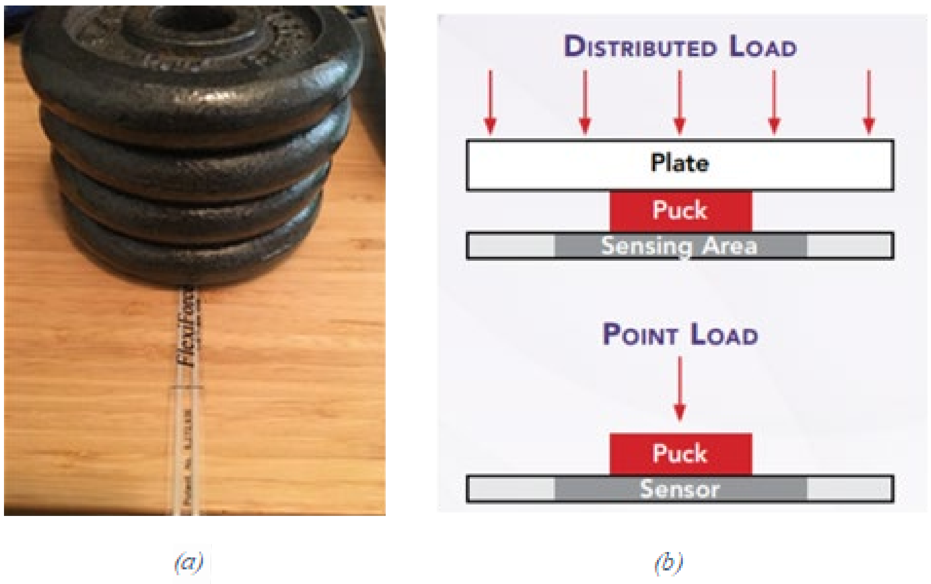


Figure 26: (a) 2.5lbs weight stacked on force sensor, (b) illustration showing the proper testing method using force pucks [1]

4.1 (T) Measure user finger flexion and extension within a range of 0-100 lbf:

The 0-100 lbf range is a combined range for all five force sensors, so the result of this single-sensor test should have a range of 0-20 lbf.

For testing, turn on and place the sensor glove flat on a table, force sensors facing up. Pair the glove with a computer via Bluetooth and open the Arduino Serial Monitor. Document the reading without any weights. Place the Force Puck flat on the face of any force sensor; make sure the Force Puck covers the entire sensor region of the force sensor as seen in Figure 1.b. Place one 2.5 lbs. weight on the force puck and document the Serial Monitor reading. Add an additional 2.5 lbs. weight and repeat the previous step; repeat until reaching 20 lbs. Final output weight reading should accurately read between 0-20 lbf.

Discussion of Results: The results from this test determined each force sensor was capable of reading >20 lbf (>100 lbf). Furthermore, the data captured during this test was also used for interpolating readings post-exercise and displaying actual force values to the user. **Figure 27** below shows the polynomial relation between the applied forces and sensor values; this same procedure was also followed to equate flex bent percentages from the flex sensors.

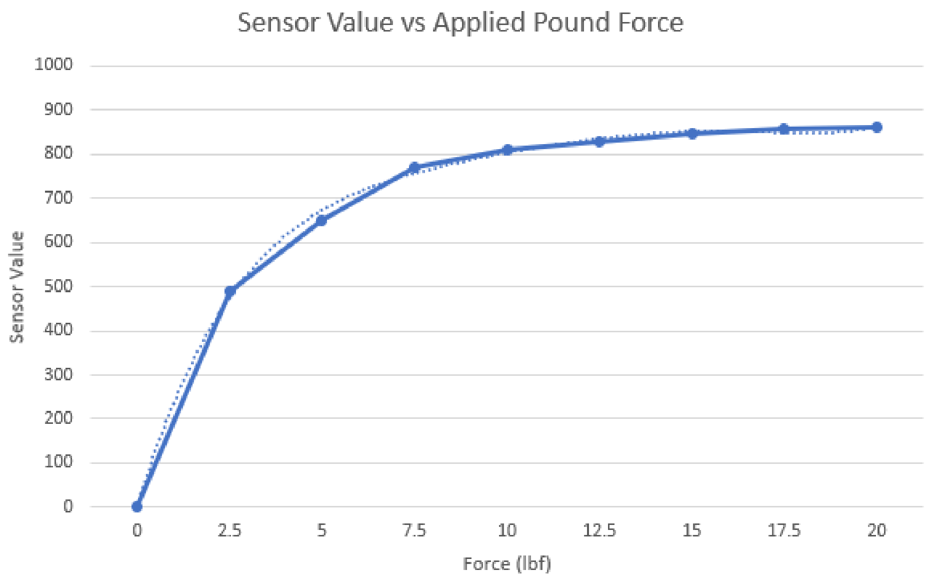


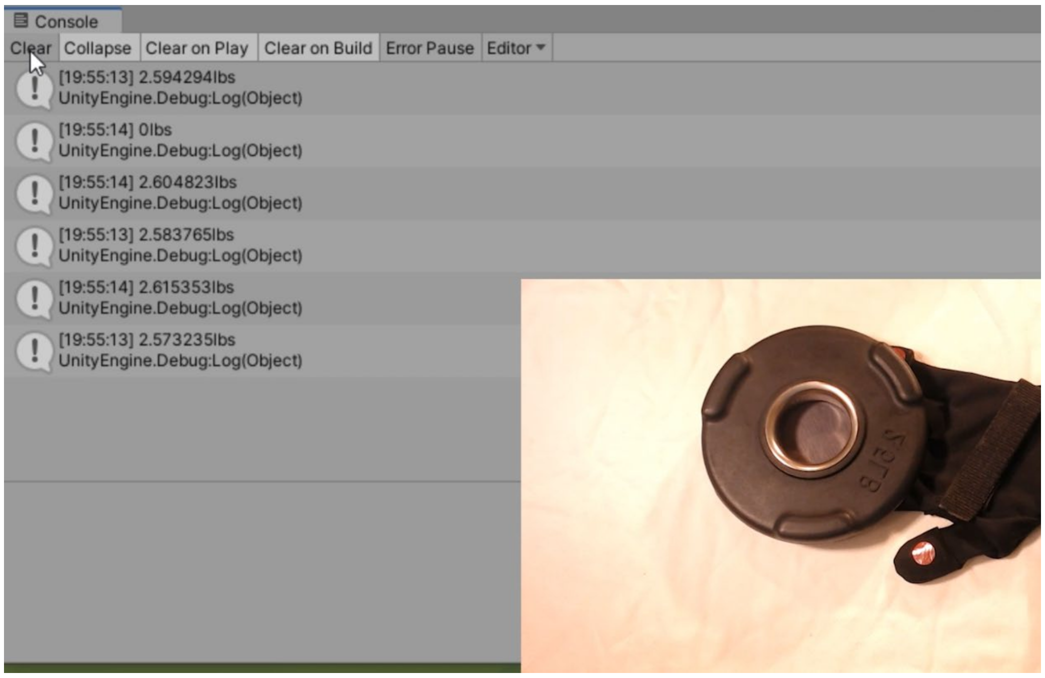
Figure 27. Plot showing relation between force sensor values and actual applied forces

4.2 (T) Measure user finger flexion and extension with a sensitivity range of ± 5 lbf:

The ±5 lbf tolerance is a combined tolerance for all five force sensors, so the result of this single-sensor test should have a tolerance of ±1 lbf.

For testing, turn on and place the sensor glove flat on a table, force sensors facing up. Pair the glove with a computer via Bluetooth and open the Arduino Serial Monitor. Place the Force Puck flat on the face of any force sensor; make sure the Force Puck covers the entire sensor region of the force sensor. Place one 20 lbs. weight on the force puck and document the Serial Monitor reading. Final output weight reading should be 20 ± 1 lbf.

Discussion of Results: The Force Puck tests resulted in force variances of under 5% per sensor, meeting the listed specification. However, a further test was conducted to ensure the accuracy of the sensors after they were installed in the final glove. Since the sensors would only be used while in the glove, both the interpolation in **Figure 27** and the accuracy tests here were repeated and the results of these tests used in the final calculations. **Figure 28** below shows the weight test performed on the "gloved" force sensors.



```
[19:55:13] 2.594294lbs
UnityEngine.Debug:Log(Object)
[19:55:14] 0lbs
UnityEngine.Debug:Log(Object)
[19:55:14] 2.604823lbs
UnityEngine.Debug:Log(Object)
[19:55:13] 2.583765lbs
UnityEngine.Debug:Log(Object)
[19:55:14] 2.615353lbs
UnityEngine.Debug:Log(Object)
[19:55:13] 2.573235lbs
UnityEngine.Debug:Log(Object)
```

Figure 28. Gloved sensor weight test

Real-time hand tracking accuracy and positioning on-screen

An AR-type positional tracking test will be employed to evaluate hand-tracking accuracy and positioning. First, a coordinate grid will be printed out with millimeter-spaced lines. A 3D-coordinate grid with millimeter-spaced lines will also be created in the virtual world and the two grids will be lined up on the XY-plane.

5.1 (T) Product tracks hand and finger movement within tolerance ± 10mm:

For testing, the subject will place his/her hand in a certain orientation on the grid and note the coordinates of the palm and fingertips in the VR space. Keeping the hand in the same position, the coordinates of the palm and fingertips will be measured in the real world using a vertical ruler for the z-axis and the grid for the x and y values. Finally, the error will be determined by calculating the distance between the two sets of corresponding points for the palm and fingertips. The final position should be accurate with a tolerance of ± 10 mm.

<u>Discussion of Results:</u> A table the exact dimension of the physical table being tested on was first calibrated into the program using the initial Table Calibration tool. A grid plane with 10x15mm boxes was then placed on the main table within Unity. Finally, a matching grid was placed on the physical table, as seen in **Figure 29**.

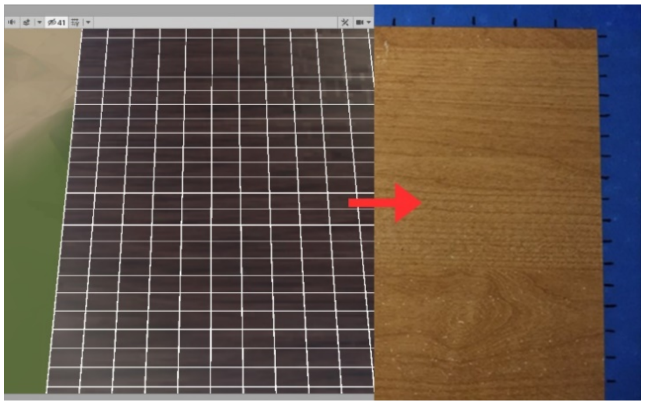
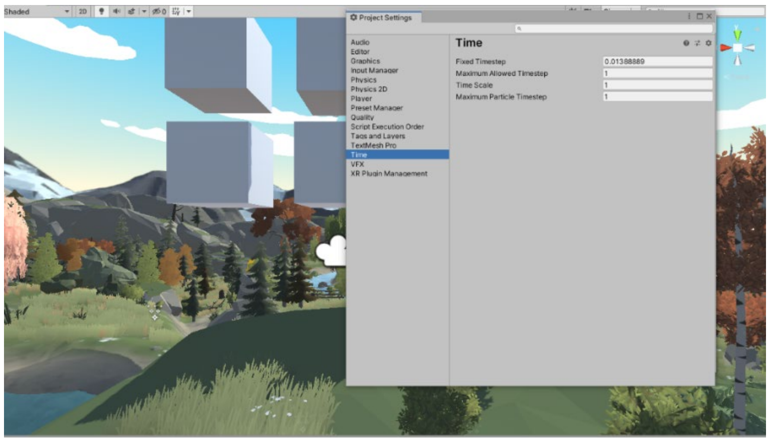


Figure 29. Hand tracking accuracy test

5.2 (I) Track physical objects with a minimum refresh rate of 2 Hz to a maximum error of 3 cm in x, y, and z directions:

Object tracking will be performed through real-time tracking (pass-through camera). Since objects will not be tracked in real-time, tracking will simply be its position within the application: application refresh rate is determined by the Oculus Quest headset specifications, 72 Hz. For verification, inspect the Oculus Quest specifications listed in the owner's manual. The Project Settings reflect a 1/72 second refresh rate, see **Figure 30** under *Fixed Timestamp.*



Time

Fixed Timestep	0.01388889
Maximum Allowed Timestep	1
Time Scale	1
Maximum Particle Timestep	1

Figure 30: Time-Stamp Refresh Rate Verification

The same applies for object positioning; an object's position will be determined by its location in the application, relative to the user's simulated/tracked hand. Thus, the maximum error of 3 cm in all directions is determined by the accuracy of **5.1** – "Product tracks hand and finger movement within tolerance ± 10mm". For verification, inspect the test results from **5.1**.

Discussion of Results: As discussed in **5.2**, the passing of **5.1** results in a pass for this test.

Scalable program difficulty and improvements in user's affected limbs

The program's difficulty system scales with user progression. A baseline is obtained on the first session and is used a referenced starting point. See **Figure 36** in **Appendix B** for full Test Plan Breakdown.

6.1 (T) Increase difficulty by increasing number of repetitions and resistance force:

Demonstration will be conducted to verify that the game is adding in repetitions. If a certain number of targets are hit the game will increase the number of targets or increase the speed at which they appear during the next session.

For testing, equip then turn on the sensor-glove and the Oculus headset. After application launch and calibrations, select Exercise 1 on the Main Menu. Document the target number of repetitions displayed for the exercise. Complete the exercise and produce the target number of repetitions. After completion, return to the Main Menu. Select Exercise 1, document the change in the target number of repetitions during gameplay. Upon successful completion of the target number of repetitions for an exercise, the game should increase the number of targets and/or increase the speed at which they appear during the next session of that exercise.

Discussion of Results: After increasing the difficulty slider from the exercise selection screen (**Figure 8**), the results can be seen by either starting the game in multiple difficulties, or by simply inspecting the code in **Appendix D**.

6.2 (T) Device improves hand strength and dexterity:

The team will run through the exercise's multiple times in a week to test and verify that a person's hand will feel fatigue and muscle development from executing the VR exercises.

Due to present circumstances, appropriate testing for determining the ability of the device to improve the hand strength and dexterity of stoke patients will not be able to be conducted. Although research articles point to exercises that have quantity, duration, and intensity can greatly improve hand functionality [17], testing on actual stroke patients is necessary to test the true rehabilitation functionality of Denovus PT. In lieu of testing the device on a stroke patient, a Denovus team member will test the improvement of grip strength functionality by completing exercises on his non-dominant hand that was affected by wrist fracture. The non-dominant hand wrist fracture will simulate a minor loss of grip functionality that can be improved through use of Denovus Physical Therapy. Two other Denovus members from the VR team will act as the control group

For testing, the team member will power up and put on the headset then complete the entire program of exercises for the day. The team member and control group will repeat this procedure for ten days straight, then use an analog grip strengthener to compare strength growth before and after the ten days of P.T. On average, the affected team member with the wrist fracture gained approximately 1.3lbf of grip strength and 0.70lbf in pinch force while the unaffected team members gained approximately 1.55lbf of grip strength and 0.625lbf in pinch force. Though minimal gains were acquired, this small unofficial test proves that hand strength gain is possible through Denovus PT.

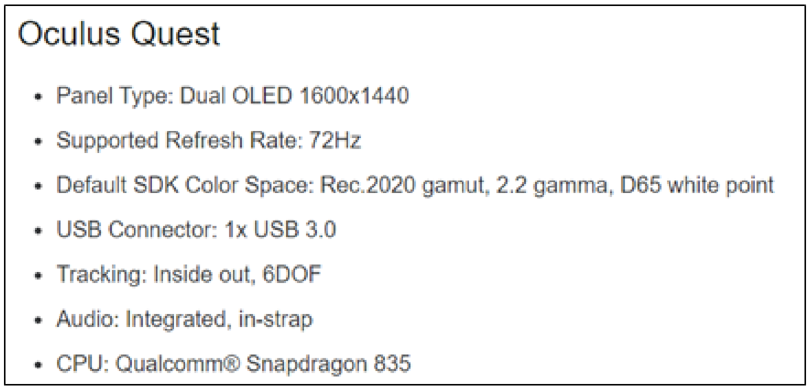
<u>Discussion of Results:</u> Since the current socially distanced climate prevents the testing of the device on stroke patients, the results of team member testing are a sufficient substitute.

<u>Hardware weight specifications</u>

7.1 (I) Headset weighs less than 2.5 lbs.:

An inspection will be conducted by placing the headset on two different weight scales that the team owns, to verify the headset is under 2.5 lbs. This process will then be repeated on the second scale to ensure accuracy.

<u>Discussion of Results:</u> Refer to Oculus Quest specifications, **Figure 31.**


Oculus Quest

- Panel Type: Dual OLED 1600x1440
- Supported Refresh Rate: 72Hz
- Default SDK Color Space: Rec.2020 gamut, 2.2 gamma, D65 white point
- USB Connector: 1x USB 3.0
- Tracking: Inside out, 6DOF
- Audio: Integrated, in-strap
- CPU: Qualcomm® Snapdragon 835

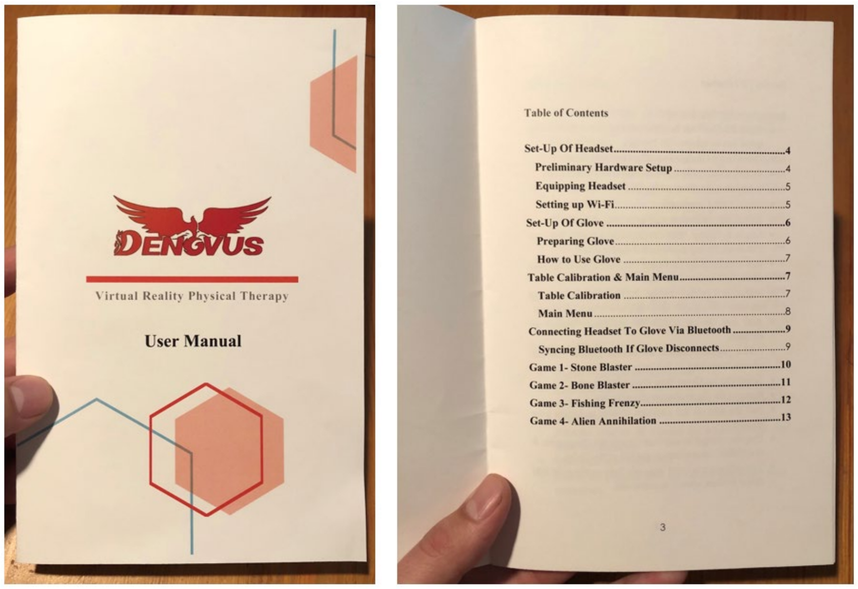
Figure 31: Oculus Quest Hardware specifications [15]

Program documentation and guidance

8.1 (I) Includes a user's manual:

An inspection of a user's manual and the appropriate system descriptions will be conducted. The user manual will be given to a random sample of ten people who will then answer a short survey regarding readability, brevity, and comprehension of the Denovus User Manual. The survey answers will be formatted on a scale of extremely unlikely to extremely likely with a total of 5 answer choices. An Overall Average Rating of Somewhat Likely denotes a successful User Manual. See **Appendix B** for full Test Plan Breakdown.

Discussion of Results: A digital version of the program User's Manual can be found in the Final Project Documentation Package, and physical version was supplied with the program (**Figure 32** below).



DENOVUS

Virtual Reality Physical Therapy

User Manual

Table of Contents

3

Figure 32. Physical User's Manual

8.2 (I) VR application includes a user's tutorial:

A demonstration of a user's tutorial will be conducted and recorded. After putting on the headset and starting the application, an examination of the user tutorial will be conducted to ensure the video plays without interruption and successfully functions as a proper tutorial.

Discussion of Results: As seen in **Figure 33**, each exercise begins with a simple tutorial.

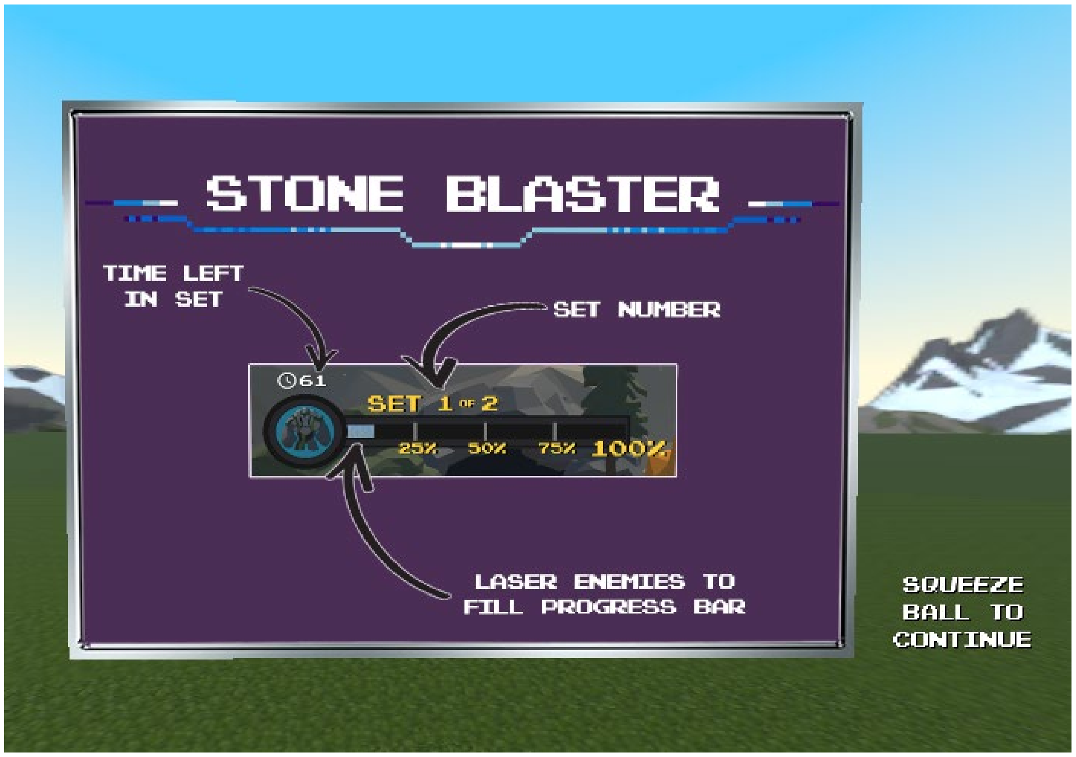


Figure 33. Exercise tutorial introduction screen

Prototype Construction

Bill of Materials

Table 4 below represents the final cost required to manufacture one complete therapy prototype.

Table 4: Bill of Materials

Item Description	Vendor Information	Model/Catalog Number	Qty	Cost Per	Total Cost
(Oculus) Quest All-in-one VR Gaming Headset - 64GB	Amazon	B07HNW68ZC	1	\$399.00	\$399.00
(Arduino) WiFi Development Tools (802.11) UNO WIFI Rev2	Mouser	782-ABX00021	1	\$44.90	\$44.90
(SparkFun) Switch IC Development Tools A/D MUX Breakout CD74HC4067	Mouser	474-BOB-09056	1	\$5.50	\$5.50
(Nightshade Electronics) energyShield 2 Basic Arduino Battery Supply	Amazon	B06VVBRR7H	1	\$29.95	\$29.95
(CUI Inc.) Wall Mount AC Adapters 6W 12V 0.5A US blade micro USB	Mouser	490-SWI5-12-N-MUB	1	\$6.55	\$6.55
(Adafruit) Daughter Cards & OEM Boards Arduino Proto Shield Kit - Stackable R3	Mouser	485-2077	1	\$9.95	\$9.95
(Adafruit) Short Flex Sensor	Adafruit	1070	1	\$7.95	\$7.95
(Adafruit) Long Flex Sensor	Adafruit	182	4	\$12.95	\$51.80
(Tekscan) FlexiForce Pressure Sensor - 25lbs.	Sparkfun	SEN-08712	5	\$21.95	\$109.75
(SparkFun) SparkFun Accessories Amphenol FCI Clinchr Conn. (2 POS,M)	Mouser	474-COM-14195	10	\$1.95	\$19.50
3D Printed Controller and Battery Housing (Base)	UTDesign	G-CBH-1	2.04 in^3	\$15 + \$8/in^3	\$31.32
3D Printed Controller and Battery Housing (Top)	UTDesign	G-CBH-2	2.13 in^3	\$15 + \$8/in^3	\$32.04
Steel Phillips Flat Head Screw, M3 x 0.5 mm Thread, 50 mm Long (Pack of 50)	McMaster-Carr	91420A140	1	\$9.00	\$9.00
Steel Pan Head Phillips Screw, M2 x 0.4 mm Thread, 10 mm Long (Pack of 100)	McMaster-Carr	92005A033	1	\$4.14	\$4.14
Steel Pan Head Phillips Screw, M2 x 0.4 mm Thread, 14 mm Long (Pack of 100)	McMaster-Carr	92005A036	1	\$8.00	\$8.00
Steel Pan Head Phillips Screw, M2 x 0.4 mm Thread, 16 mm Long (Pack of 100)	McMaster-Carr	92005A037	1	\$4.66	\$4.66
18-8 Stainless Steel Thin Hex Nut, M2 x 0.4 mm Thread, DIN 439B, ISO 4035 (Pack of 50)	McMaster-Carr	90710A020	1	\$3.24	\$3.24

18-8 Stainless Steel Thin Square Nut, M3 x 0.5 mm Thread (Pack of 50)	McMaster-Carr	97258A101	1	$10.00	$10.00
(Yageo) Carbon Film Resistors - Through Hole - 10kOhms 500mW	Mouser	603-CFR50SJT-52-10K	10	$0.07	$0.69
Solder Leaded - 100-gram Spool	Sparkfun	TOL-09161	1	$5.95	$5.95
Neoprene Fabric Black (Width: 54")	Joann	17518622	1	$13.99	$13.99
(Coats & Clark) Dual Duty XP Heavy Thread 125yds	Joann	xprd840932	1	$2.99	$2.99
(VELCRO) 1"x 6' Home Decor Tape	Joann	prd29495	1	$14.99	$14.99
(Dritz) 3/4" Ribbed Nonroll Elastic Hank Black	Joann	11384583	1	$3.49	$3.49
(voidbiov) Hand Squeeze Stress Balls Set with Carry Bag 3 Resistance, Finger Wrist Arthritis Therapy Rehab Exerciser, Carpal Tunnel, Stroke Rehabilitation Equipment	Amazon	ASIN: B07BFZ4BV4	1	$13.99	$13.99
(DSD TECH) HC-06 Wireless Bluetooth Module	Amazon	ASIN: B01FCQZ8VW	1	$8.49	$8.49
Total Cost					**$798.44**

Fabrication Processes

Glove Fabric Shell Fabrication

The process of sewing the glove consists of cutting out and sewing five key components and connecting them all together to make the final prototype. The five key components are the inner layer, outer cover, assembly base cover, straps, and sleeve. To construct the inner layer, begin by tracing a hand on a piece of paper that runs from the top of the middle finger to midway down the forearm and cut it out to create the fabric print. Make a vertical cut starting at the base of the index finger to cut off the thumb on the fabric print. Use the fabric print to trace out the hand, thumb, and the fingertips on a piece of Neoprene fabric. Cut out the pieces and sew the fingertips to the tops of the fingers on the large cutouts. Sew the thumb to the large hand cutout so that it is facing inward on the middle of the palm. Doing this ensures the thumb will be easy to bend and use during therapy. Cut out small rectangular strips that are the width of the fingers and sew them to each finger running from the top joint to midway down the back of the hand. The last step in creating the inner layer is to cut out the straps and sew them to the outer left side located at the first index finger joint and at the wrist. The metal rings are attached by using a small rectangular piece of fabric and sewing each end of the piece, with the ring in the middle, on to the inner layer located adjacent to the straps.

The next step in creating the glove is to cut out and sew the outer cover. This is accomplished by adding ¼ of an inch to the perimeter of the paper cut out and using that for the outer cover since it must be wider than the inner layer. Follow the same process for attaching the fingertips and thumb to the inner layer for the top part of the cover, while the bottom part does not have fingertips attached. Cut out holes for the straps and sew a perimeter around the holes to ensure they do not tear. Sew the top and bottom part together and invert them to not show the sewn seems.

To create the assembly base cover, three pieces of fabric are needed. Two pieces of fabric to cover the top and bottom and one long strip to cover the perimeter of the base. Cut holes in the top piece where the bolt holes are located on the assembly housing base. Sew the long strip to the perimeter of the bottom piece of fabric, and then sew the conjoined bottom piece to the perimeter of the top piece. Be sure to leave a small flap of fabric on the bottom to connect it to the rest of the glove. Invert the assembly base cover and slide the base inside the cover.

The last portion that was created before joining all the parts together was the sleeve. Two make the sleeve, two long pieces of fabric need to be cut out that are slightly wider than the forearm. The top piece needs to have hole cut out to match the bolt holes on the base and the bottom piece needs to have a slight chamfer to account for the decreasing size of the forearm from the elbow to the wrist. Sew the top and bottom pieces together and invert the sleeve to hide the seams.

The final step is to connect all the pieces together. Start by covering the inner layer with the outer cover. Then sew the bottom portion of the outer cover to the flap on the assembly base cover. Finally slide the sleeve over the outside of the assembly base cover and outside cover. Run the bolts through all the holes in the sleeve and assembly base cover. Tighten the bolts to secure the base and sleeve in place. Following this process will result with the complete and final version of the prototype glove.

<u>Glove Controller Assembly</u>

Concerning the sensor glove electrical components, the wireless connectivity system was the system with the most design overhaul. The original design intended to utilize the built-in Wi-Fi capabilities of the ECC608 crypto chip accelerator of Arduino Uno WIFI REV2 however, certain complications arose when attempting to read and write files from a webserver onto the Oculus Quest. Standard encryption and privacy notices within the Quest's computer architecture did not allow for this type of connection. The failure to show the patients progress within the VR application itself proved detrimental to a couple of our specifications and as such, another approach to wireless connectivity was required. As an alternative, the second design intended to utilize the onboard Bluetooth Low Energy (BLE) capabilities of the Arduino Uno WIFI Rev2 however the read and write architecture was incompatible with the Oculus Quest's Bluetooth signature. A Bluetooth module capable of outputting signals with signatures of HC-04, HC-05, or HC-06 were required for seamless connectivity to the Oculus Quest. As such, the current HC-06 Bluetooth module design was utilized as it was the most powerful, in terms of input/output speeds, of the available Bluetooth modules compatible with the Oculus Quest.

Cost Summary

The costs shown here in **Table 5** reflect the actual costs of materials and services, while labor is based on hypothetical fully-loaded rates for the various personnel involved in the project.

Table 5: Actual Costs of Materials and Services

Category				**Cost**
		Materials		$1762
		Services		$0
Labor Classifications	*Loaded Rate*	*Hours*	*Cost*	
Engineer	$100/hr.	952	$95,200	
Technical Manager	$175/hr.	60	$10,500	
Engineering Director	$250/hr.	15	$3,750	
Staff	$60/hr.	30	$1,800	
		Labor Subtotal		$111,250
			Total	**$113,012**

Materials

Most of the materials purchased for the project were used directly in the final prototype, aside from the initial Adafruit FSRs and the extra Oculus Quest headset used for testing. The main goal for this project was to maintain an affordable price point so costs were kept low, as demonstrated by the BOM.

Services

No external services or rentals were required. 3D printed parts listed on BOM were printed in-house on a team member's

Labor

The Denovus Team built, designed, and coded everything in house. All sewing, soldering, and testing of the Sensor Glove was performed by the Glove Sub-Team. Multiple design iterations were considered as no member was experienced in the field of glove design and manufacture. Similarly, all coding, debugging, and testing was executed by the VR Team, which were also not experts in Game Design and Code Implementation. Regardless of Denovus' lack of expertise in these fields during the start of development, a fully functional VR assisted Physical Therapy program prototype was created; one that satisfies the entire scope, deliverables, and constraints set up at the onset of the project.

Conclusions and Recommendations

The Denovus team has successfully designed a virtual reality therapy that can help rehabilitate upper extremities after experiencing a stroke and increase a stroke user's ability to perform activities of daily living. By developing and modeling several VR games after proven PT hand movement exercises and creating a soothing, discordant, and immersive VR environment, Denovus has combined a multitude of successful rehabilitation techniques into one, easy-to-use, package. Lamentably, there were complications with recruiting active stroke patients to test our PT mainly due to the current global pandemic. A severely limited rehabilitation test was performed using active team members' grip and pinch strength. The three VR Team members that had access to the Physical Therapy a day's worth of exercises and measured their grip and pinch strengths using analogue grip force meters. Two weeks of almost daily use of the Denovus program, the users experienced soreness and fatigue due to the physical therapy regimen provided with the program and experienced an average flat 1.5lbf increase in grip strength and 0.625lbf increase in pinch strength. The affected team member saw similar gains in grip strength and pinch strength.

To improve the design of our physical therapy program, extensive testing on older adults would need to be conducted with specific regard to ease of use, increase in upper limb functionality, and self-reported increase in activities of daily living. There are a few adjustments and additions that our team would recommend instituting. Adding additional games to the program would give the users more variety which would lead to less boredom. Furthermore, these additional games would also give our team the opportunity to add a wider variety of exercise movements that would allow us to develop exercises for the lower arm and shoulders as well as inner hand dexterity (finger abductions and resistive flexion/extension). Denovus would also recommend developing games further with harder tasks so our VR therapy could be used in later stages of stroke recovery, games that require the user to keep their pinch and flexion within a specific range to promote fine motor movements. Conversely, Denovus would also like to add very simple flex and pinch exercises to help those with severe flaccidity or spasticity also benefit from Denovus VR. Denovus also recommends adding an auto difficulty setting to progress the user to more difficult mini-game exercises and tasks to ensure continued intensity and challenge present within the exercises. Denovus would also recommend using smaller electrical components so the housing unit could be smaller and lighter, and the hand tracking of the Oculus Quest would not *flicker* on occasion as it does with the bulkier model. Finally, Denovus would recommend upgrading the Oculus Quest to the Oculus Quest 2 as it is cheaper in cost as well as includes enhanced processing and hand tracking capabilities.

References

[1] S. Han, "Stroke Recovery: What to Expect," *healthline*, 07-Jun-2018. [Online]. Available: https://www.healthline.com/health/stroke/recovery#complications. [Accessed: 01-Dec-2020].

[2] E. S. Lawrence, C. Coshall, R. Dundas, J. Stewart, A. G. Rudd, R. Howard, and C. D. A. Wolfe, "Estimates of the Prevalence of Acute Stroke Impairments and Disability in a Multiethnic Population," *Stroke*, vol. 32, no. 6, pp. 1279–1284, June 2001.

[3] "Types of Stroke," *Centers for Disease Control and Prevention*, 31-Jan-2020. [Online]. Available:https://www.cdc.gov/stroke/types_of_stroke.htm. [Accessed: 15-Jun-2020].

[4] Delgado and S. Han, "Is rehabilitation always successful?," *Stroke Recovery: What to Expect*,07-Jun-2018.[Online]. Available: https://www.healthline.com/health/stroke/recovery#outlook. [Accessed: 11-Jun-2020].

[5] K. M. Godwin, J. Wasserman, S. K. Ostwald, "Cost Associated with Stroke: Outpatient Rehabilitative Services and Medication," *pubmed.gov,* Oct. 18, 2011. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/22120036/. [Accessed July 2, 2020].

[6] H. Hoffman, "How to Stay Motivated During Stroke Recovery," *saebo.com,* April 6, 2017. [Online]. Available: https://www.saebo.com/blog/stay-motivated-stroke-recovery/. [Accessed July 2, 2020].

[7] J. S. Park, G. Lee, J. B. Choi, N. K. Hwang, Y. J. Jung, "Game-based hand resistance exercise versus traditional manual hand exercises for improving hand strength, motor function, and compliance in stroke patients: A multi-center randomized controlled study," *NeuroRehabilitation,* p. 221, 2019.

[8] "FSR400 - Force Sensor," *Components101*. [Online]. Available: https://components101.com/sensors/fsr400-force-sensor. [Accessed: 07-Dec-2020].

[9] S. M. Hatem, G. Saussez, M. D. Faille, V. Prist, X. Zhang, D. Dispa, and Y. Bleyenheuft, "Rehabilitation of Motor Function after Stroke: A Multiple Systematic Review Focused on Techniques to Stimulate Upper Extremity Recovery," *Front Hum Neuroscience,* vol. 10, 2016.

[10] D. Lloyd-Jones, R. Adams, M. Carnethon, G. D. Simone, T. B. Ferguson, K. Flegal, E. Ford, K. Furie, A. Go, K. Greenlund, N. Haase, S. Hailpern, M. Ho, V. Howard, B. Kissela, S. Kittner, D. Lackland, L. Lisabeth, A. Marelli, M. Mcdermott, J. Meigs, D. Mozaffarian, G. Nichol, C. Odonnell, V. Roger, W. Rosamond, R. Sacco, P. Sorlie, R. Stafford, J. Steinberger, T. Thom, S. Wasserthiel-Smoller, N. Wong, J. Wylie-Rosett, and Y. Hong, "Heart Disease and

Stroke Statistics—2019 Update," *A Report From the American Heart Association*, vol. 119, no. 3, Jan. 2019.

[11] Lee, Kyoung Bo et al. "Six-month functional recovery of stroke patients: a multi-time-point study." *International journal of rehabilitation research. Internationale Zeitschrift fur Rehabilitationsforschung. Revue internationale de recherches de readaptation* vol. 38,2 (2015): 173-80. doi:10.1097/MRR.0000000000000108

[12] "5 REASONS TECHNOLOGY IS THE FUTURE OF THE REHABILITATION SYSTEM," *INDUSTRY INSIGHT*. [Online]. Available: https://assets.cdnma.com/16114/assets/Industry Insight - TOFU 01 11mar.pdf. [Accessed: 01-Jul-2020].

[13] Soares, Antonio Vinicius, Woellner, Simone Suzuki, Andrade, Camile dos Santos, Mesadri, Thiago Julian, Bruckheimer, Alessandro Diogo, & Hounsell, Marcelo da Silva. (2014). The use of Virtual Reality for upper limb rehabilitation of hemiparetic Stroke patients. *Fisioterapia em Movimento*, *27*(3), 309-317. https://doi.org/10.1590/0103-5150.027.003.AO01

[14] Nair, Krishnan & Taly, Arun. (2003). Stroke rehabilitation: Traditional and modern approaches. Neurology India. 50 Suppl. S85-93.

[15] Bolognini, Nadia, et al. "The Sensory Side of Post-Stroke Motor Rehabilitation." *Restorative Neurology and Neuroscience*, U.S. National Library of Medicine, 11 Apr. 2016, www.ncbi.nlm.nih.gov/pmc/articles/PMC5605470/.

[16] Eschmann, H., Héroux, M., Cheetham, J., Potts, S. and Diong, J., 2019. *Thumb And Finger Movement Is Reduced After Stroke: An Observational Study*. [online] US National Library of Medicine. Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6561636/> [Accessed 1 July 2020].

[17] "Oculus Quest: All-in-One VR Headset," *Oculus*. [Online]. Available: https://www.oculus.com/quest/?locale=en_US. [Accessed: 11-Jun-2020].

[18] Carmeli, Eli & Peleg, Sara & Bartur, Gadi & Elbo, Enbal & Vatine, Jean-Jacques. (2011). HandTutorTM enhanced hand rehabilitation after stroke - a pilot study. Physiotherapy research international : the journal for researchers and clinicians in physical therapy. 16. 191-200. 10.1002/pri.485.

[19] Bae, Jung Hyun & Kang, Si Hyun & Seo, Kyung & Kim, Don Kyu & Shin, Hyun Iee & Shin, Hye. (2015). Relationship Between Grip and Pinch Strength and Activities of Daily Living in Stroke Patients. Annals of Rehabilitation Medicine. 39. 752. 10.5535/arm.2015.39.5.752.

Appendix A – 3D CAD Model of Controller Housing, Top & Bottom

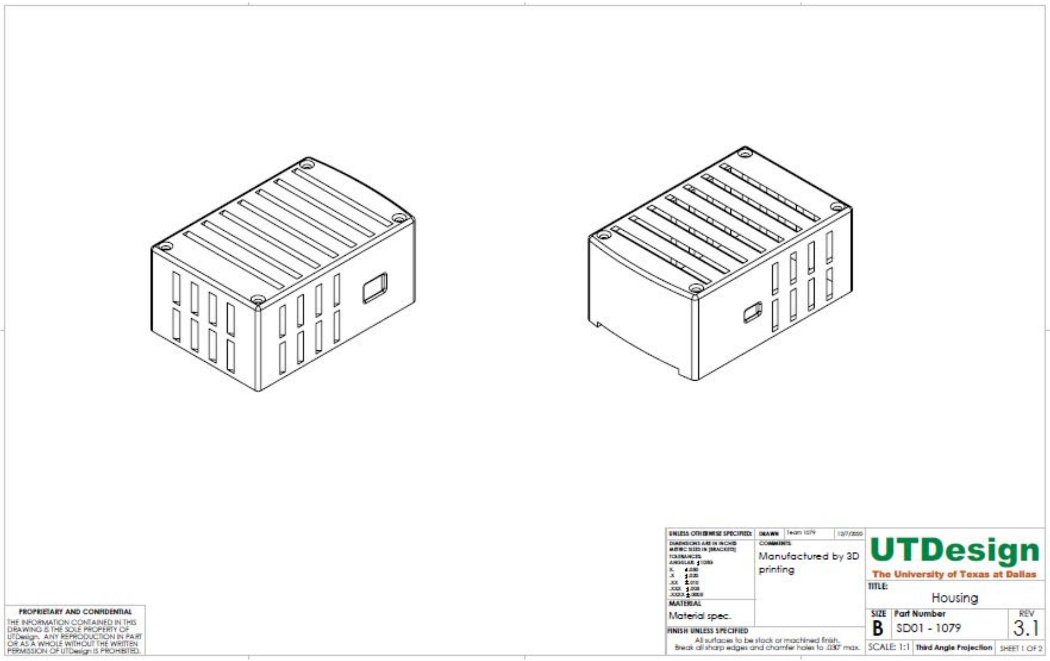

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UTDesign. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UTDesign IS PROHIBITED.
UNLESS OTHERWISE SPECIFIED:
DRAWN
Team 1079
12/7/2020
COMMENTS:
Manufactured by 3D printing
MATERIAL
Material spec.
FINISH UNLESS SPECIFIED
All surfaces to be stock or machined finish. Break all sharp edges and chamfer holes to .030" max.
UTDesign
The University of Texas at Dallas
TITLE:
Housing
SIZE
B
Part Number
SD01 - 1079
REV
3.1
SCALE: 1:1
Third Angle Projection
SHEET 1 OF 2

Figure 34. Controller Housing CAD Model, Top

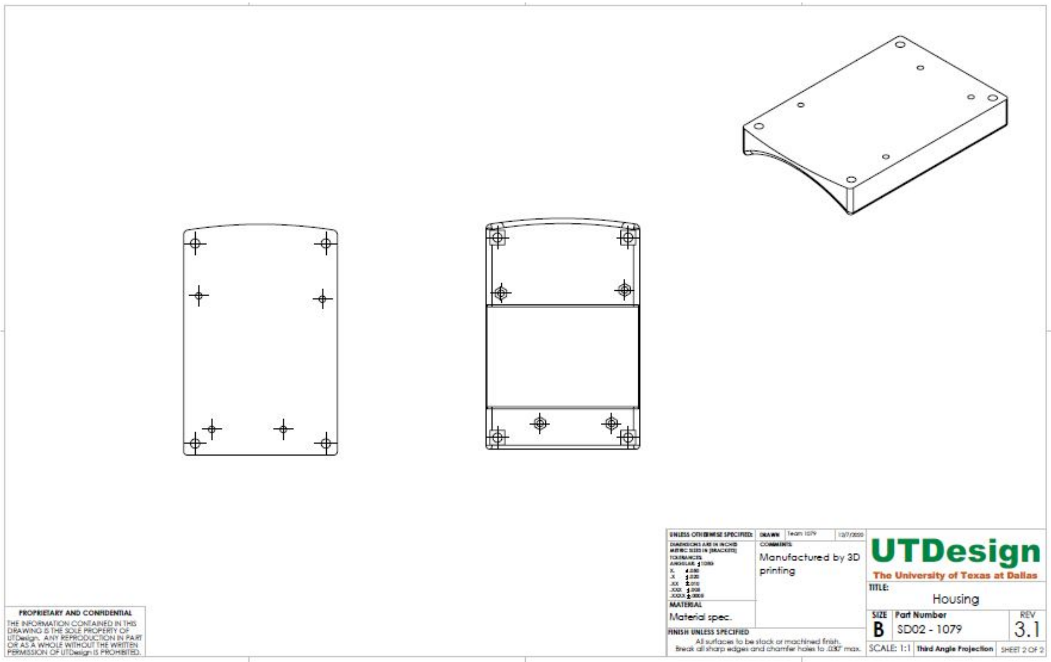

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UTDesign. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UTDesign IS PROHIBITED.
UNLESS OTHERWISE SPECIFIED:
DRAWN
Team 1079
12/7/2020
COMMENTS:
Manufactured by 3D printing
MATERIAL
Material spec.
FINISH UNLESS SPECIFIED
All surfaces to be stock or machined finish. Break all sharp edges and chamfer holes to .030" max.
UTDesign
The University of Texas at Dallas
TITLE:
Housing
SIZE
B
Part Number
SD02 - 1079
REV
3.1
SCALE: 1:1
Third Angle Projection
SHEET 2 OF 2

Figure 35. Controller Housing CAD Model, Bottom

Appendix B - Test Plan Breakdown of Major Components

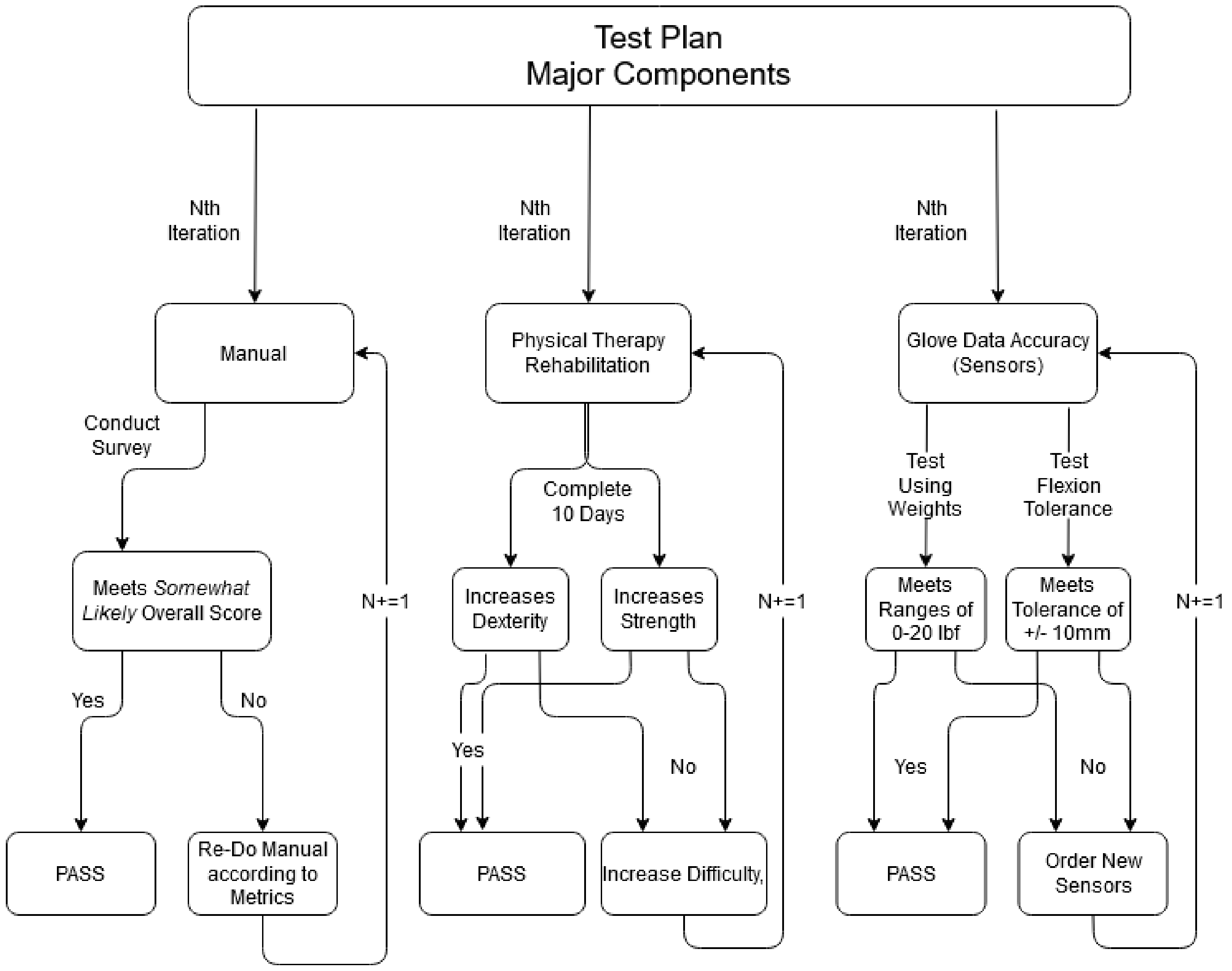


Figure 36. Test Plan Component Breakdown

Appendix C – Table Calibration Script (ScaleTable.cs)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ScaleTable : MonoBehaviour
{
    private GameObject table;
    public Renderer tableRenderer;
    private Mesh tableMesh;
    private float tableLength;
    private float tableHeight;
    private float tableWidth;

    void Start()
    {
        table = GameObject.Find("Table");
        MeshFilter tableMeshFilter = (MeshFilter)table.GetComponent("MeshFilter");
        tableMesh = tableMeshFilter.mesh;
        tableRenderer = table.GetComponent<Renderer>();
        tableLength = tableMesh.bounds.size.x * table.transform.localScale.x;
        tableHeight = tableMesh.bounds.size.y * table.transform.localScale.y;
        tableWidth = tableMesh.bounds.size.z * table.transform.localScale.z;
        tableRenderer.enabled = false;
    }

    void Scale(Vector3 leftCorner, Vector3 rightCorner)
    {
        float distance = Vector3.Distance(leftCorner, rightCorner);
        float height = (leftCorner.y + rightCorner.y) / 2;
        Vector3 midpoint = new Vector3((leftCorner.x + rightCorner.x) / 2, 0f, (leftCorner.z +
rightCorner.z) / 2);
        table.transform.localScale = new Vector3(distance / tableMesh.bounds.size.x, height /
tableMesh.bounds.size.y, distance / tableMesh.bounds.size.x);
        PlayerPrefs.SetFloat("TableXScale", table.transform.localScale.x);
        PlayerPrefs.SetFloat("TableYScale", table.transform.localScale.y);
        PlayerPrefs.SetFloat("TableZScale", table.transform.localScale.z);
        float newTableLength = tableMesh.bounds.size.x * table.transform.localScale.x;
        float newTableWidth = tableMesh.bounds.size.z * table.transform.localScale.z;
        Vector3 leftCornerProjection = new Vector3(leftCorner.x, 0, leftCorner.z);
        Plane plane = new Plane(leftCorner, rightCorner, leftCornerProjection);
        Vector3 normalVec = plane.normal;
        Vector3 unitNormalVec = plane.normal / plane.normal.magnitude;
        table.transform.position = new Vector3(midpoint.x - (newTableLength / 2) *
unitNormalVec.x, 0, midpoint.z - (newTableWidth / 2) * unitNormalVec.z);
        table.transform.LookAt(normalVec);
        tableRenderer.enabled = true;
    }

    public void ScalerPressed(Vector3 leftCornerPosition, Vector3 rightCornerPosition)
    {
        Scale(leftCornerPosition, rightCornerPosition);
        StartCoroutine(DelayScalerPressed());
    }

    IEnumerator DelayScalerPressed()
    {
        yield return new WaitForSecondsRealtime(3);
        GameObject.Find("SceneManager").GetComponent<SceneSwitcher>().SwitchCallibrationToMain();
    }
}
```

Appendix D – Main Turret Exercise Manager (TurretBehaviour.cs)

```
using UnityEngine;
using UnityEngine.UI;

public class TurretBehaviour : MonoBehaviour
{
    [Header("General")]
    private float killGoal;
    private float killTotal = 0f;
    private int previousDeathCount = 1;
    private int deathCount = 0;
    private int bossCount = 0;
    public float clock;
    [HideInInspector]
    public int setNumber = 1;
    [HideInInspector]
    public float nextRoundCountdown = 10f;
    private bool isSqueezed;
    private bool retryShow;
    private float retrySqueezeDelay = 2f;
    private float finalCountdown = 6f;
    private bool finalCheck;
    private bool squeeze;
    private int squeezeValue;
    private readonly float addedTimeAmount = 15f;
    [HideInInspector]
    public float totalTime = 0f;
    private bool bossTextShown;
    private int totalScore = 0;
    private bool showValues;
    [HideInInspector]
    public bool exerciseComplete;

    [Header("Unity Setup")]
    public Image progressBar;
    public Text countdownText;
    public Text setNumberText;
    public GameObject AddedTimeTextMesh;
    public Text nextRoundCountdownText;
    public Text finalRoundText;
    public Text outOfTimeText;
    public Text outOfTimePinchText;
    public Text goodJobText;
    public Text goodJobTextFinal1;
    public Text goodJobTextFinal2;
    public Text bossText;
    public Image golemIcon;
    public Image skellyIcon;
    public Text setTextOf2;
    public Text scoreText;
    public Image skellyScore;

    void Awake()
    {
        bossText.transform.Translate(0f, -500f, 0f);
        //used for testing
        //PlayerPrefs.SetInt("Squeeze", 1);
        //PlayerPrefs.SetInt("Difficulty", 1);
        //PlayerPrefs.SetString("Scene", "Turret");
    }
    void Start()
    {
        squeezeValue = PlayerPrefs.GetInt("Squeeze", 1);
        if (squeezeValue == 1)
            squeeze = true;
        else
            squeeze = false;

        setNumberText.text = setNumber.ToString();
        nextRoundCountdownText.text = Mathf.Floor(nextRoundCountdown).ToString();
        progressBar.fillAmount = 0;

        if (squeeze)
        {
            skellyIcon.transform.Translate(0f, -500f, 0f);
            skellyScore.transform.Translate(0f, -500f, 0f);
            clock = 60f;
            killGoal = 10f;

        }
        else
        {
            golemIcon.transform.Translate(0f, -500f, 0f);
            killGoal = 4f;
            setTextOf2.transform.Translate(0f, -500f, 0f);

            // Difficulty scalar
            int difficulty = PlayerPrefs.GetInt("Difficulty", 1);
            if (difficulty == 1)
                clock = 150f;
            else if (difficulty == 2)
                clock = 180f;
            else
                clock = 210f;

        }
    }
    void Update()
    {
        //// Used for testing runtime sensor lb readings
        //GameObject bluetoothManager = GameObject.Find("BluetoothManager");
        //BTManager manager = bluetoothManager.GetComponent<BTManager>();
        //float poundValue;

        //if (manager.sensorValue[2] >= 0f && manager.sensorValue[2] <= 300f)
        //    poundValue = (.0083f * manager.sensorValue[2])/1.7f;
        //else if (manager.sensorValue[2] >= 300f && manager.sensorValue[2] <= 440f)
        //    poundValue = ((.0179f * manager.sensorValue[2]) - 2.8571f)/1.7f;
        //else if (manager.sensorValue[2] >= 440f && manager.sensorValue[2] <= 550f)
        //    poundValue = ((.0227f * manager.sensorValue[2]) - 5f)/1.7f;
        //else if (manager.sensorValue[2] >= 550f && manager.sensorValue[2] <= 730f)
        //    poundValue = ((.0139f * manager.sensorValue[2]) - .1389f)/1.7f;
        //else
        //    poundValue = 0f;

        //Debug.Log(poundValue + "lbs");

        // If user is out of time before filling progress bar
        if (clock <= 0.1f)
        {
            // If golem, stop all laser audio and effects
            GameObject enemy = GameObject.FindGameObjectWithTag("Enemy");
            if (squeeze)
            {
                EnemyGolem e = enemy.GetComponent<EnemyGolem>();

                GameObject turret = GameObject.Find("LaserGun");
                Turret t = turret.GetComponent<Turret>();

                if (t.lineRenderer.enabled)
                {
                    t.lineRenderer.enabled = false;
                    t.beamImpactEffect.Stop();
                    t.impactLight.enabled = false;
                }

                t.beamAudio.Stop();
                e.hitAudio.Stop();

                if (!retryShow)
                {
                    outOfTimeText.transform.Translate(0f, 5f, 0f);
                    retryShow = true;
                    FindObjectOfType<AudioManager>().Play("Fail");
                }
                BTManager manager =
GameObject.Find("BluetoothManager").GetComponent<BTManager>();

                if (retrySqueezeDelay >= 0f)
                {
                    retrySqueezeDelay -= Time.deltaTime;
                }

                else if (manager.sensorValue[0] > TriggerValues.forceTrigger ||
                         manager.sensorValue[1] > TriggerValues.forceTrigger ||
                         manager.sensorValue[2] > TriggerValues.forceTrigger ||
                         manager.sensorValue[3] > TriggerValues.forceTrigger ||
                         manager.sensorValue[4] > TriggerValues.forceTrigger)
                {
                    isSqueezed = true;
                }

                else if (isSqueezed)
                {
                    if (nextRoundCountdown >= 0.1f)
                    {
                        if (nextRoundCountdown == 10f)
                        {
                            outOfTimeText.transform.Translate(0f, -5f, 0f);
                            finalRoundText.transform.Translate(0f, 5f, 0f);
                        }

                        nextRoundCountdown -= Time.deltaTime;
                        nextRoundCountdownText.text = Mathf.Floor(nextRoundCountdown).ToString();
                    }
                    else
                    {
                        GameObject turret2 = GameObject.Find("LaserGun");
                        Turret t2 = turret2.GetComponent<Turret>();
                        t2.ResetHealth();

                        finalRoundText.transform.Translate(0f, -5f, 0f);
                        killTotal = 0f;
                        progressBar.fillAmount = killTotal / killGoal;
                        nextRoundCountdown = 10f;
                        clock = 60f;
                        retryShow = false;
                        isSqueezed = false;
                        retrySqueezeDelay = 2f;
                    }
                }
            }
            // Else skelly, stop all audio and effects
            else
            {
                exerciseComplete = true;

                EnemySkeleton e = enemy.GetComponent<EnemySkeleton>();

                GameObject turret = GameObject.Find("LaserGun");
                Turret t = turret.GetComponent<Turret>();

                if (t.lineRenderer.enabled)
                {
                    t.lineRenderer.enabled = false;
                    t.beamImpactEffect.Stop();
                    t.impactLight.enabled = false;
                }

                if (!finalCheck)
                {
```

```
GameObject.Find("FinalGloveValuesUI").GetComponent<FinalGloveValues>().UpdateValues();
                    finalCheck = true;
                }

                t.beamAudio.Stop();
                e.hitAudio.Stop();

                if (!retryShow)
                {
                    outOfTimePinchText.transform.Translate(0f, 5f, 0f);
                    retryShow = true;
                    FindObjectOfType<AudioManager>().Play("Victory");
                }

                BTManager manager =
GameObject.Find("BluetoothManager").GetComponent<BTManager>();

                if (retrySqueezeDelay >= 0f)
                {
                    retrySqueezeDelay -= Time.deltaTime;
                }

                else if (manager.sensorValue[0] > TriggerValues.forceTrigger ||
                         manager.sensorValue[1] > TriggerValues.forceTrigger ||
                         manager.sensorValue[2] > TriggerValues.forceTrigger ||
                         manager.sensorValue[3] > TriggerValues.forceTrigger ||
                         manager.sensorValue[4] > TriggerValues.forceTrigger)
                {
                    golemIcon.transform.Translate(0f, 500f, 0f);
                    setTextOf2.transform.Translate(0f, 500f, 0f);
                    PlayerPrefs.SetInt("SkellyScore", totalScore);

GameObject.Find("SceneManager").GetComponent<SceneSwitcher>().SwitchExerciseToMain();
                }
            }

            // Show appropriate fail message

            clock += Time.deltaTime;
        }

        // Successful set loop
        else if (progressBar.fillAmount == 1)
        {
            // If user completed set 1
            if (squeeze)
            {
                if (setNumber == 1)
                {
                    // Reset timer if below 60
                    if (clock < 60f)
                    {
                        clock = 60f;
                    }
                    if (nextRoundCountdown >= 0.1f)
                    {
                        if (nextRoundCountdown == 10f)
                        {
                            FindObjectOfType<AudioManager>().Play("Victory");
                            goodJobText.transform.Translate(0f, 5f, 0f);
                            finalRoundText.transform.Translate(0f, 5f, 0f);
                        }

                        clock += Time.deltaTime;
                        nextRoundCountdown -= Time.deltaTime;
                        nextRoundCountdownText.text = Mathf.Floor(nextRoundCountdown).ToString();
                    }
                    else
                    {
                        goodJobText.transform.Translate(0f, -5f, 0f);
                        finalRoundText.transform.Translate(0f, -5f, 0f);
                        setNumber = 2;
                        setNumberText.text = setNumber.ToString();
                        killTotal = 0f;
                        progressBar.fillAmount = killTotal / killGoal;
                        nextRoundCountdown = 10f;
                    }

                }
                // Else user has completed both exercise sets
                else
                {
                    exerciseComplete = true;

                    if (!showValues)
                    {
GameObject.Find("FinalGloveValuesUI").GetComponent<FinalGloveValues>().UpdateValues();
                        showValues = true;
                    }

                    clock += Time.deltaTime;

                    if (finalCountdown >= 0f)
                    {
                        if (finalCountdown == 6f)
                        {
                            FindObjectOfType<AudioManager>().Play("Victory");
                            goodJobTextFinal1.transform.Translate(0f, 5f, 0f);
                        }

                        finalCountdown -= Time.deltaTime;

                    }
                    else
                    {
                        if (!finalCheck)
                        {
                            goodJobTextFinal1.transform.Translate(0f, -5f, 0f);
                            goodJobTextFinal2.transform.Translate(0f, 5f, 0f);
                            finalCheck = true;
                        }

                        BTManager manager =
GameObject.Find("BluetoothManager").GetComponent<BTManager>();

                        if (retrySqueezeDelay >= 0f)
                        {
                            retrySqueezeDelay -= Time.deltaTime;
                        }
                        else if (manager.sensorValue[0] > TriggerValues.forceTrigger ||
                                 manager.sensorValue[1] > TriggerValues.forceTrigger ||
                                 manager.sensorValue[2] > TriggerValues.forceTrigger ||
                                 manager.sensorValue[3] > TriggerValues.forceTrigger ||
                                 manager.sensorValue[4] > TriggerValues.forceTrigger)
                        {
                            //GetComponent<SaveStone>().SaveStoneValues(totalTime);
                            skellyIcon.transform.Translate(0f, 500f, 0f);
                            skellyScore.transform.Translate(0f, 500f, 0f);


GameObject.Find("SceneManager").GetComponent<SceneSwitcher>().SwitchExerciseToMain();
                        }
                    }
                }
            }

            // Else progress bar full and skelly
            else
            {
                if (nextRoundCountdown >= 0.1f)
                {
                    if (nextRoundCountdown == 10f)
                    {
                        FindObjectOfType<AudioManager>().Play("Victory");
                        goodJobText.transform.Translate(0f, 5f, 0f);
                        finalRoundText.transform.Translate(0f, 5f, 0f);
                    }

                    clock += Time.deltaTime;
                    nextRoundCountdown -= Time.deltaTime;
                    nextRoundCountdownText.text = Mathf.Floor(nextRoundCountdown).ToString();
                }
                else
                {
                    goodJobText.transform.Translate(0f, -5f, 0f);
                    finalRoundText.transform.Translate(0f, -5f, 0f);
                    setNumber++;
                    setNumberText.text = setNumber.ToString();
                    killTotal = 0f;
                    progressBar.fillAmount = killTotal / killGoal;
                    nextRoundCountdown = 10f;
                }
            }
        }
        clock -= Time.deltaTime;
        countdownText.text = Mathf.Floor(clock).ToString();

        GameObject GM = GameObject.Find("GameMaster");
        WaveSpawner spawner = GM.GetComponent<WaveSpawner>();
        deathCount = spawner.deathCount;
        bossCount = spawner.bossCount;
        totalScore = (deathCount - 1) + (bossCount * 2);
        if (totalScore < 0)
            scoreText.text = 0.ToString();
        else
            scoreText.text = totalScore.ToString();

        totalTime += Time.deltaTime;
        // Display added time when enemy killed
        if (deathCount > previousDeathCount)
        {
            if (squeeze)
            {
                Destroy(Instantiate(AddedTimeTextMesh, new Vector3(-1.56f, 6.49f, -10.25f),
transform.rotation), 1.45f);
                clock += addedTimeAmount;
            }

            killTotal++;
            progressBar.fillAmount = killTotal / killGoal;
            previousDeathCount = deathCount;
        }

        if (spawner.enemyBoss)
        {
            if (!bossTextShown)
            {
                bossText.transform.Translate(0f, 500f, 0f);
                bossTextShown = true;
            }
        }
        else
        {
            if (bossTextShown)
            {
                bossText.transform.Translate(0f, -500f, 0f);
                bossTextShown = false;
            }
        }
    }
}
```

Appendix E – Main Space Invaders Exercise Manager (UFO.cs)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Linq;

public class UFO : MonoBehaviour
{
    private GameObject[] wayPoints;
    private int nextWayPointIndex;
    private float speed = 2;
    private Transform RayTool;
    private SphereCollider collider;
    private BTManager bTManager;
    private int forceThreshold;
    private waves Waves;
    private int lastWayPointIndex;
    private Vector3 lastWayPoint;
    private Vector3 nextWayPoint;
    private Vector3 flyAwayPosition = new Vector3(-25f, 50f, -50f);
    private bool flyingAway = false;
    private GameObject target;
    public Material aliensMaterial;
    public ParticleSystem alienDeathEffect;
    public Color _orange;

    public enum Finger
    {
        Index, Middle, Ring, Pinky
    };

    public Finger finger;

    void Start()
    {
        RayTool =
GameObject.Find("RightHandAnchor").transform.Find("OVRHandPrefab").gameObject.GetComponent<OVRHan
d>().PointerPose;
        var wayPointsSorted = GameObject.FindGameObjectsWithTag("Waypoints").OrderBy( go =>
go.name ).ToList();
        wayPoints = wayPointsSorted.ToArray();
        lastWayPointIndex = wayPoints.Length - 1;
        lastWayPoint = wayPoints[lastWayPointIndex].transform.position;
        nextWayPointIndex = 0;
        nextWayPoint = wayPoints[0].transform.position;
        collider = GetComponent<SphereCollider>();
        target = gameObject.transform.Find("Target").gameObject;
        target.SetActive(false);
        bTManager = FindObjectOfType<BTManager>();
        _orange = new Color(1.0f, 0.5f, 0.0f);

        Waves = GameObject.Find("GameMaster").GetComponent<waves>();
        switch(Waves.globalDifficulty)
        {
            case 1:
                forceThreshold = TriggerValues.forceTrigger;
                break;
            case 2:
                forceThreshold = TriggerValues.forceTriggerMedium;
                break;
            case 3:
                forceThreshold = TriggerValues.forceTriggerHard;
                break;
            default:
                break;
        }
    }

    void Update()
    {
        MoveUFO();
        RaycastHit hit;
        if (Physics.Raycast(RayTool.position, RayTool.TransformDirection(Vector3.forward), out
hit, Mathf.Infinity) )
        {
            if (GameObject.ReferenceEquals(hit.collider.gameObject, this.gameObject))
            {
                target.SetActive(true);
                switch (finger)
                {
                    case Finger.Index:
                        if (bTManager.sensorValue[1] > forceThreshold)
                        {
                            GameObject.Find("SaveManager").GetComponent<SaveSensors>().toggle();
                            DestroyUFO();
                        }
                        break;
                    case Finger.Middle:
                        if (bTManager.sensorValue[2] > forceThreshold)
                        {
                            GameObject.Find("SaveManager").GetComponent<SaveSensors>().toggle();
                            DestroyUFO();
                        }
                        break;
                    case Finger.Ring:
                        if (bTManager.sensorValue[3] > forceThreshold)
                        {
                            GameObject.Find("SaveManager").GetComponent<SaveSensors>().toggle();
                            DestroyUFO();
                        }
                        break;
                    case Finger.Pinky:
                        if (bTManager.sensorValue[4] > forceThreshold)
                        {
                            GameObject.Find("SaveManager").GetComponent<SaveSensors>().toggle();
                            DestroyUFO();
                        }
                        break;
                    default:
                        break;
                }
            }
            else
            {
                target.SetActive(false);
            }
        }
        /*
        //For testing
        switch (finger)
        {
            case Finger.Index:
                if (Waves.currentIndexValue > forceThreshold)
                {
                    Debug.Log("Destroying UFO");
                    DestroyUFO();
                }
                break;
            case Finger.Middle:
                if (Waves.currentMiddleValue > forceThreshold)
                    DestroyUFO();
                break;
            case Finger.Ring:
                if (Waves.currentRingValue > forceThreshold)
                    DestroyUFO();
                break;
            case Finger.Pinky:
                if (Waves.currentPinkyValue > forceThreshold)
                    DestroyUFO();
                break;
            default:
                break;
        }
        */
    }
    private void MoveUFO()
    {

        if (Vector3.Distance(transform.position, lastWayPoint) > 0.1f)
        {
            nextWayPoint = wayPoints[nextWayPointIndex].transform.position;
            Vector3 direction = nextWayPoint - transform.position;
            transform.Translate(direction.normalized * speed * Time.deltaTime, Space.World);
        }
        if (Vector3.Distance(transform.position, nextWayPoint) < 0.5f && nextWayPointIndex <
lastWayPointIndex)
        {
            nextWayPointIndex++;
        }
        if (nextWayPointIndex == lastWayPointIndex && Vector3.Distance(transform.position,
lastWayPoint) < 0.5f)
        {
            flyingAway = true;
            collider.enabled = false;
            transform.rotation = new Quaternion(0f, -90f, 0f, 0f);
        }
        if (flyingAway)
        {
            Vector3 direction = flyAwayPosition - transform.position;
            transform.Translate(direction.normalized * 10 * speed * Time.deltaTime, Space.World);
            if (Vector3.Distance(transform.position, flyAwayPosition) < 0.5f)
                DestroyUFOWithoutIncrement();
        }
    }
    public void DestroyUFO()
    {
        Waves.UFOsDestroyed++;

        switch (finger)
        {
            case Finger.Index:
                aliensMaterial.SetColor("_Color", Color.magenta);
                Destroy((Instantiate(alienDeathEffect, transform.position,
transform.rotation)).gameObject, 1f);
                break;
            case Finger.Middle:
                aliensMaterial.SetColor("_Color", Color.yellow);
                Destroy((Instantiate(alienDeathEffect, transform.position,
transform.rotation)).gameObject, 1f);
                break;
            case Finger.Ring:
                aliensMaterial.SetColor("_Color", _orange);
                Destroy((Instantiate(alienDeathEffect, transform.position,
transform.rotation)).gameObject, 1f);
                break;
            case Finger.Pinky:
                aliensMaterial.SetColor("_Color", Color.grey);
                Destroy((Instantiate(alienDeathEffect, transform.position,
transform.rotation)).gameObject, 1f);
                break;
            default:
                break;
        }
        GameObject.Find("SaveManager").GetComponent<SaveSensors>().toggle();
        Destroy(gameObject);
    }
    public void DestroyUFOWithoutIncrement()
    {
        Destroy(gameObject);
    }
}
```

Appendix F – Main Fishing Exercise Manager (BobBehaviour.cs)

```
using System.Collections;
using UnityEngine;

public class BobBehaviour : MonoBehaviour
{
    public bool castable = true;
    public bool reeling = false;
    public bool onReel = false;
    public FishSpawner fishSpawner;
    private Transform fishingRod;
    private float speed = 5f;
    private Vector3 offset = new Vector3(0f, -0.4f, 0);
    private Vector3 localOffset = new Vector3(0f, -0.4f, 0);
    public Vector3 direction;
    public Renderer castArea;
    public Score scoreText;
    private bool CR_running = false;
    public AudioSource reelingAudio;

    private BTManager bTManager;
    public PosesToRender poses;
    public int waitingPose;
    public bool waitingPoseReel;
    private Rigidbody rb;
    private float thrust = 20f;
    private bool toggled;

    void Start()
    {
        fishingRod = GameObject.Find("AnchorPoint").transform;
        bTManager = FindObjectOfType<BTManager>();
        rb = GetComponent<Rigidbody>();
        rb.useGravity = false;
        GetComponent<Floater>().enabled = false;

        transform.parent = fishingRod;
        transform.localPosition = localOffset;
        createNewInstruction();
    }

    void Update()
    {
        if(castable) //If bob is castable, wait for user to trigger then cast bob and make it
uncastable
        {
            transform.parent = fishingRod;
            transform.localPosition = localOffset;
            switch (waitingPose)
            {
                case 5:
                    if (bTManager.sensorValue[5] < fishSpawner.thumbTrigger &&
bTManager.sensorValue[5] != -999) //bTManager.sensorValue[5] < fishSpawner.thumbTrigger
                    {
                        transform.parent = null;
                        GetComponent<Rigidbody>().useGravity = true;
                        GetComponent<Floater>().enabled = true;
                        Vector3 direction = transform.forward;
                        rb.AddForce(-0.5f*direction.x * thrust, 0, 0.866f*direction.z,
ForceMode.Impulse);
                        StartCoroutine(waitCheckBobLocation());
                        castable = false;
                        hideInstruction();
                        poses.showWaitText();
                        fishSpawner.caught = true;
                    }
                    break;
                case 6:
                    if (bTManager.sensorValue[6] < fishSpawner.indexTrigger &&
bTManager.sensorValue[6] != -999) //bTManager.sensorValue[6] < fishSpawner.indexTrigger
```

```
                    {
                        transform.parent = null;
                        GetComponent<Rigidbody>().useGravity = true;
                        GetComponent<Floater>().enabled = true;
                        Vector3 direction = transform.forward;
                        rb.AddForce(-0.5f * direction.x * thrust, 0, 0.866f * direction.z,
ForceMode.Impulse);
                        StartCoroutine(waitCheckBobLocation());
                        castable = false;
                        hideInstruction();
                        poses.showWaitText();
                        fishSpawner.caught = true;
                    }
                    break;
                case 7:
                    if (bTManager.sensorValue[7] < fishSpawner.middleTrigger &&
bTManager.sensorValue[7] != -999) //bTManager.sensorValue[7] < fishSpawner.middleTrigger
                    {
                        transform.parent = null;
                        GetComponent<Rigidbody>().useGravity = true;
                        GetComponent<Floater>().enabled = true;
                        Vector3 direction = transform.forward;
                        rb.AddForce(-0.5f * direction.x * thrust, 0, 0.866f * direction.z,
ForceMode.Impulse);
                        StartCoroutine(waitCheckBobLocation());
                        castable = false;
                        hideInstruction();
                        poses.showWaitText();
                        fishSpawner.caught = true;
                    }
                    break;
                case 8:
                    if (bTManager.sensorValue[8] < fishSpawner.ringTrigger &&
bTManager.sensorValue[8] != -999) //bTManager.sensorValue[8] < fishSpawner.ringTrigger
                    {
                        transform.parent = null;
                        GetComponent<Rigidbody>().useGravity = true;
                        GetComponent<Floater>().enabled = true;
                        Vector3 direction = transform.forward;
                        rb.AddForce(-0.5f * direction.x * thrust, 0, 0.866f * direction.z,
ForceMode.Impulse);
                        StartCoroutine(waitCheckBobLocation());
                        castable = false;
                        hideInstruction();
                        poses.showWaitText();
                        fishSpawner.caught = true;
                    }
                    break;
                case 9:
                    if (bTManager.sensorValue[9] < fishSpawner.pinkyTrigger &&
bTManager.sensorValue[9] != -999) //bTManager.sensorValue[9] < fishSpawner.pinkyTrigger
                    {
                        transform.parent = null;
                        GetComponent<Rigidbody>().useGravity = true;
                        GetComponent<Floater>().enabled = true;
                        Vector3 direction = transform.forward;
                        rb.AddForce(-0.5f * direction.x * thrust, 0, 0.866f * direction.z,
ForceMode.Impulse);
                        StartCoroutine(waitCheckBobLocation());
                        castable = false;
                        hideInstruction();
                        poses.showWaitText();
                        fishSpawner.caught = true;
                    }
                    break;
                default:
                    break;
            }
        }

        if(onReel)
        {
            switch(waitingPose)
            {
                case 5:
                    if (bTManager.sensorValue[5] < fishSpawner.thumbTrigger &&
bTManager.sensorValue[5] != -999) //bTManager.sensorValue[5] < fishSpawner.thumbTrigger
                        reeling = true;
                    else
                        reeling = false;
                    break;
                case 6:
                    if (bTManager.sensorValue[6] < fishSpawner.indexTrigger &&
bTManager.sensorValue[6] != -999) //bTManager.sensorValue[6] < fishSpawner.indexTrigger
                        reeling = true;
                    else
                        reeling = false;
                    break;
                case 7:
                    if (bTManager.sensorValue[7] < fishSpawner.middleTrigger &&
bTManager.sensorValue[7] != -999) //bTManager.sensorValue[7] < fishSpawner.middleTrigger
                        reeling = true;
                    else
                        reeling = false;
                    break;
                case 8:
                    if (bTManager.sensorValue[8] < fishSpawner.ringTrigger &&
bTManager.sensorValue[8] != -999) //bTManager.sensorValue[8] < fishSpawner.ringTrigger
                        reeling = true;
                    else
                        reeling = false;
                    break;
                case 9:
                    if (bTManager.sensorValue[9] < fishSpawner.pinkyTrigger &&
bTManager.sensorValue[9] != -999) //bTManager.sensorValue[9] < fishSpawner.pinkyTrigger
                        reeling = true;
                    else
                        reeling = false;
                    break;
                default:
                    reeling = false;
                    break;
            }
        }
        if(reeling)
        {
            GetComponent<Rigidbody>().useGravity = false;
            GetComponent<Floater>().enabled = false;
            direction = fishingRod.position + offset - transform.position;
            transform.Translate(direction.normalized * speed * Time.deltaTime, Space.World);
            float dist = Vector3.Distance(transform.position, fishingRod.position + offset);
            poses.changeInstructionText("Keep going!");
            if (!toggled)
            {
                GameObject.Find("SaveManager").GetComponent<SaveSensors>().toggle();
                toggled = true;
            }
            if (!reelingAudio.isPlaying)
                reelingAudio.Play();

            if (dist < 0.1f)
            {
                foreach(Transform child in transform)
                {
                    if(child.tag == "Fish")
                        Destroy(child.gameObject);
                }
                reeling = false;
                onReel = false;
                castable = true;
                waitingPoseReel = false;
                GetComponent<Rigidbody>().useGravity = false;
                GetComponent<Floater>().enabled = false;
                transform.SetParent(fishingRod);
                transform.localPosition = localOffset;
                hideInstruction();
                createNewInstruction();
                fishSpawner.numFish--;
                scoreText.setScore(fishSpawner.numFish);
                reelingAudio.Stop();
            }
        }
        else
        {
            if (toggled)
            {
                GameObject.Find("SaveManager").GetComponent<SaveSensors>().toggle();
                toggled = false;
            }
        }

    }
    private IEnumerator waitCheckBobLocation()
    {
        CR_running = true;
        yield return new WaitForSecondsRealtime(2);
        if(castArea.bounds.Contains(transform.position))
        {
            fishSpawner.caught = false;
            fishSpawner.addBobAsWaypoint();
            //Debug.Log("Bob in play area");
        }
        else
        {
            reeling = false;
            onReel = false;
            castable = true;
            waitingPoseReel = false;
            GetComponent<Rigidbody>().useGravity = false;
            GetComponent<Floater>().enabled = false;
            transform.SetParent(fishingRod);
            transform.localPosition = localOffset;
            hideInstruction();
            createNewInstruction();
            //Debug.Log("Bob not in play area");
        }
        CR_running = true;
    }
    public void createNewInstruction()
    {
        if(waitingPoseReel)
        {
            poses.changeInstructionText("Reel In");
        }
        else
        {
            poses.changeInstructionText("Cast Rod");
        }
        waitingPose = poses.getCurrentPose(waitingPose);
    }
    public void hideInstruction()
    {
        poses.deleteAllPoses();
    }
    public void endBob()
    {
        transform.SetParent(fishingRod);
        transform.localPosition = localOffset;
        hideInstruction();
        enabled = false;
    }
}
```

Appendix G – Sensor Glove Save Data Script (SaveSensors.cs)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Runtime.Serialization.Formatters.Binary;
using System.Runtime.Serialization;
using System.IO;
using System;


public class SaveSensors : MonoBehaviour
{
    [HideInInspector]
    public List<int> thumbForceSensorValues, indexForceSensorValues, middleForceSensorValues,
ringForceSensorValues, pinkyForceSensorValues = new List<int>();
    public List<int> thumbFlexSensorValues, indexFlexSensorValues, middleFlexSensorValues,
ringFlexSensorValues, pinkyFlexSensorValues = new List<int>();
    private bool savingValues = true;
    private int forceThreshold, thumbFlexThreshold, indexFlexThreshold, middleFlexThreshold,
ringFlexThreshold, pinkyFlexThreshold;
    private int thumbFlexMax, indexFlexMax, middleFlexMax, ringFlexMax, pinkyFlexMax;

    public enum Exercise
    {
        Space, Stone, Fish, Bone
    };

    public Exercise exercise;
    private BTManager manager;
    [HideInInspector]
    public int avgThumbForce, avgIndexForce, avgMiddleForce, avgRingForce, avgPinkyForce;
    [HideInInspector]
    public int maxThumbForce, maxIndexForce, maxMiddleForce, maxRingForce, maxPinkyForce;

    void Start()
    {
        manager = FindObjectOfType<BTManager>();
        forceThreshold = TriggerValues.forceTrigger;
        thumbFlexThreshold = TriggerValues.thumbFlexTrigger;
        indexFlexThreshold = TriggerValues.indexFlexTrigger;
        middleFlexThreshold = TriggerValues.middleFlexTrigger;
        ringFlexThreshold = TriggerValues.ringFlexTrigger;
        pinkyFlexThreshold = TriggerValues.pinkyFlexTrigger;
        thumbFlexMax = TriggerValues.thumbFlexMax;
        indexFlexMax = TriggerValues.indexFlexMax;
        middleFlexMax = TriggerValues.middleFlexMax;
        ringFlexMax = TriggerValues.ringFlexMax;
        pinkyFlexMax = TriggerValues.pinkyFlexMax;
        if (exercise == Exercise.Stone)
        {
            if(PlayerPrefs.GetInt("Squeeze",1) == 2)
            {
                exercise = Exercise.Bone;
            }
        }
        InvokeRepeating("SaveSensorValues", 0f, 0.05f);
    }

    public void toggle()
    {
        savingValues = !savingValues;
    }

    void SaveSensorValues()
    {
        if (savingValues)
        {
            switch (exercise)
            {
                case Exercise.Space: //Force
                    if (manager.sensorValue[0] > forceThreshold)
                        thumbForceSensorValues.Add(manager.sensorValue[0]);
                    if (manager.sensorValue[1] > forceThreshold)
                        indexForceSensorValues.Add(manager.sensorValue[1]);
                    if (manager.sensorValue[2] > forceThreshold)
                        middleForceSensorValues.Add(manager.sensorValue[2]);
                    if (manager.sensorValue[3] > forceThreshold)
                        ringForceSensorValues.Add(manager.sensorValue[3]);
                    if (manager.sensorValue[4] > forceThreshold)
                        pinkyForceSensorValues.Add(manager.sensorValue[4]);
                    break;
                case Exercise.Stone: //Force
                    if (manager.sensorValue[2] > forceThreshold)
                        middleForceSensorValues.Add(manager.sensorValue[2]);
                    break;
                case Exercise.Fish: //Flex
                    if (manager.sensorValue[5] <= thumbFlexThreshold && manager.sensorValue[5] >=
thumbFlexMax)
                        thumbFlexSensorValues.Add(manager.sensorValue[5]);
                    if (manager.sensorValue[6] <= indexFlexThreshold && manager.sensorValue[6] >=
indexFlexMax)
                        indexFlexSensorValues.Add(manager.sensorValue[6]);
                    if (manager.sensorValue[7] <= middleFlexThreshold && manager.sensorValue[7]
>= middleFlexMax)
                        middleFlexSensorValues.Add(manager.sensorValue[7]);
                    if (manager.sensorValue[8] <= ringFlexThreshold && manager.sensorValue[8] >=
ringFlexMax)
                        ringFlexSensorValues.Add(manager.sensorValue[8]);
                    if (manager.sensorValue[9] <= pinkyFlexThreshold && manager.sensorValue[9] >=
pinkyFlexMax)
                        pinkyFlexSensorValues.Add(manager.sensorValue[9]);
                    break;
                case Exercise.Bone: //Force
                    if (manager.sensorValue[0] > forceThreshold)
                        thumbForceSensorValues.Add(manager.sensorValue[0]);
                    if (manager.sensorValue[1] > forceThreshold)
                        indexForceSensorValues.Add(manager.sensorValue[1]);
                    if (manager.sensorValue[2] > forceThreshold)
                        middleForceSensorValues.Add(manager.sensorValue[2]);
                    if (manager.sensorValue[3] > forceThreshold)
                        ringForceSensorValues.Add(manager.sensorValue[3]);
                    if (manager.sensorValue[4] > forceThreshold)
                        pinkyForceSensorValues.Add(manager.sensorValue[4]);
                    break;
                default:
                    break;
            }
        }
    }


    public float[] GetSensorValue()
    {
        switch(exercise)
        {
            case Exercise.Space: //Force
                float[] values = new float[5];
                values[0] = returnAvg(thumbForceSensorValues);
                values[1] = returnAvg(indexForceSensorValues);
                values[2] = returnAvg(middleForceSensorValues);
                values[3] = returnAvg(ringForceSensorValues);
                values[4] = returnAvg(pinkyForceSensorValues);
                return values;
            case Exercise.Stone: //Force
                float[] values1 = new float[1];
                values1[0] = returnAvg(middleForceSensorValues);
                return values1;
            case Exercise.Fish: //Flex
```

```
                float[] values2 = new float[5];
                values2[0] = returnAvg(thumbFlexSensorValues);
                values2[1] = returnAvg(indexFlexSensorValues);
                values2[2] = returnAvg(middleFlexSensorValues);
                values2[3] = returnAvg(ringFlexSensorValues);
                values2[4] = returnAvg(pinkyFlexSensorValues);
                return values2;
            case Exercise.Bone: //Force
                float[] values3 = new float[5];
                values3[0] = returnAvg(thumbForceSensorValues);
                values3[1] = returnAvg(indexForceSensorValues);
                values3[2] = returnAvg(middleForceSensorValues);
                values3[3] = returnAvg(ringForceSensorValues);
                values3[4] = returnAvg(pinkyForceSensorValues);
                return values3;
            default:
                return new float[] { 0f };
        }
    }

    private float returnAvg(List<int> arr)
    {
        float total = 0;
        foreach(int i in arr)
            total += i;
        return total / arr.Count;
    }
}
```

Appendix H – Pugh Matrix Comparison

Table 6. Pugh matrix comparing Denovus to traditional physical therapy and current VR physical therapy solutions

Selection Criteria	**Traditional PT (Reference)**	**Denovus (Proposed)**	**NeuroRehab**	**MagicMoovr**
Recovery of hand functions	0	0	-	-
All-inclusive (Speech, upper, lower body)	0	-	0	-
Ease of use	0	+	-	+
Cost	0	+	+	+
Convenience of locale	0	+	-	-
Appeal	0	+	+	+
Motivational	0	+	+	0
Adaptability	0	-	0	0
Funding	0	-	-	-
Extent of testing	0	-	-	0
Sum +’s	0	5	3	3
Sum 0’s	10	1	2	3
Sum -’s	0	4	6	4
Net Score	0	1	-2	-1
Rank	2	1	4	3

Appendix I – Glove Controller Assembly Electrical Schematic

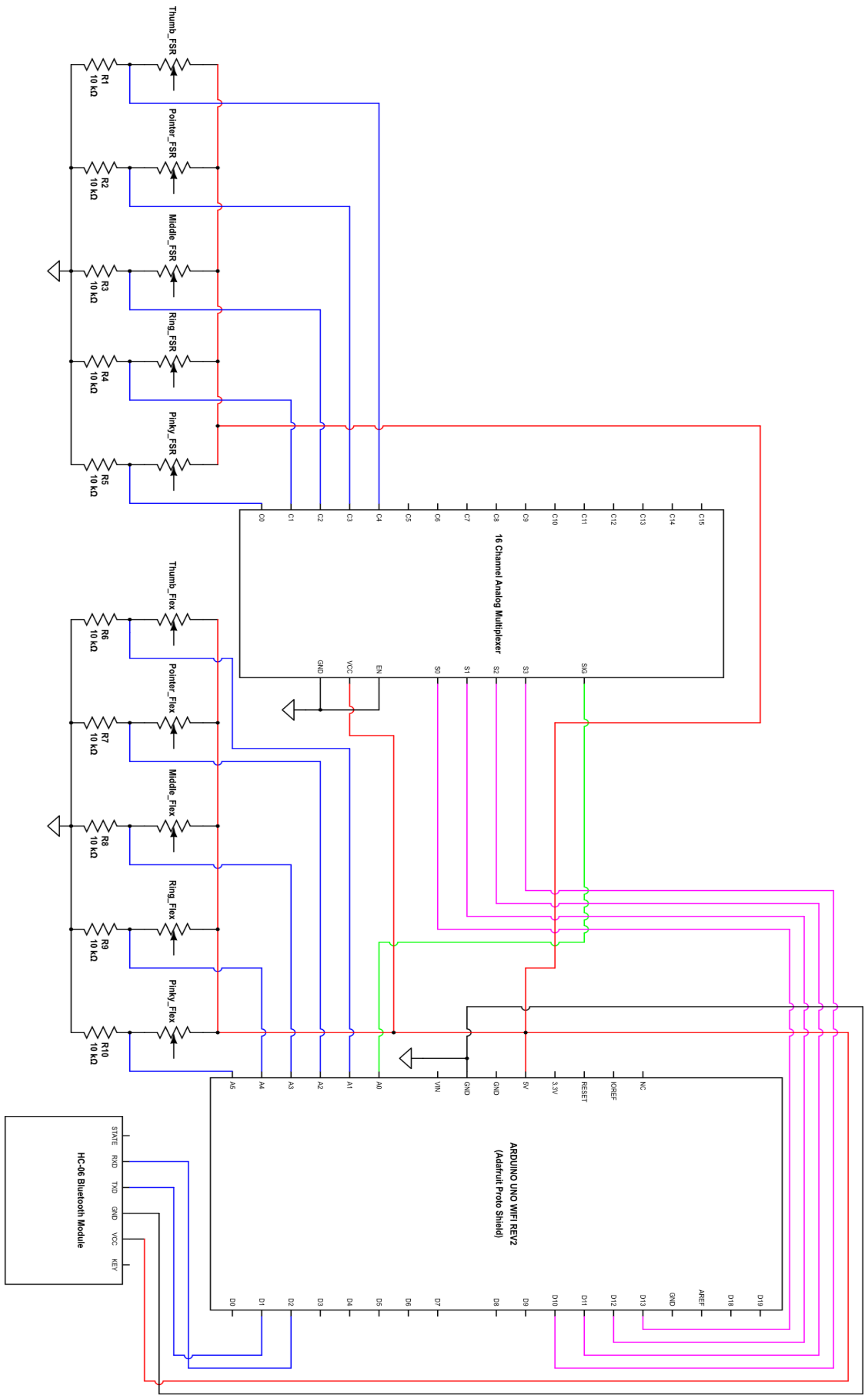

Thumb_FSR
Pointer_FSR
Middle_FSR
Ring_FSR
Pinky_FSR
R1 10 kΩ
R2 10 kΩ
R3 10 kΩ
R4 10 kΩ
R5 10 kΩ
Thumb_Flex
Pointer_Flex
Middle_Flex
Ring_Flex
Pinky_Flex
R6 10 kΩ
R7 10 kΩ
R8 10 kΩ
R9 10 kΩ
R10 10 kΩ
16 Channel Analog Multiplexer
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15
GND VCC EN S0 S1 S2 S3 SIG
ARDUINO UNO WIFI REV2
(Adafruit Proto Shield)
A5 A4 A3 A2 A1 A0 VIN GND GND 5V 3.3V RESET IOREF NC
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 GND AREF D18 D19
HC-06 Bluetooth Module
STATE RXD TXD GND VCC KEY

Figure 37. Electrical Schematic for Controller Housing Components

Appendix J – Electrical Diagram for Sensor Location on Board

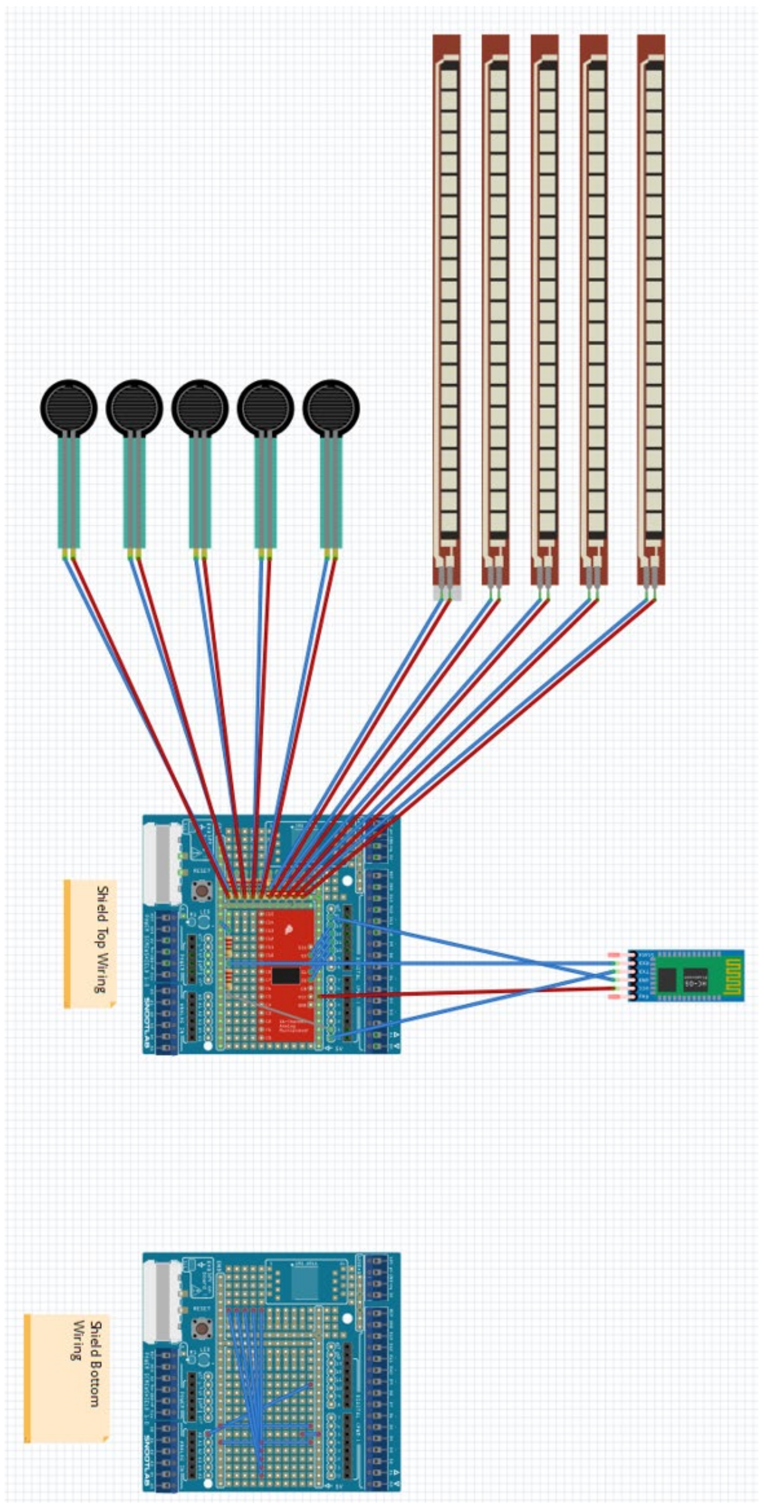


Figure 38. Electrical Diagram for attaching sensors to board

Appendix K – Electrical Diagram for Direct Connections to Arduino

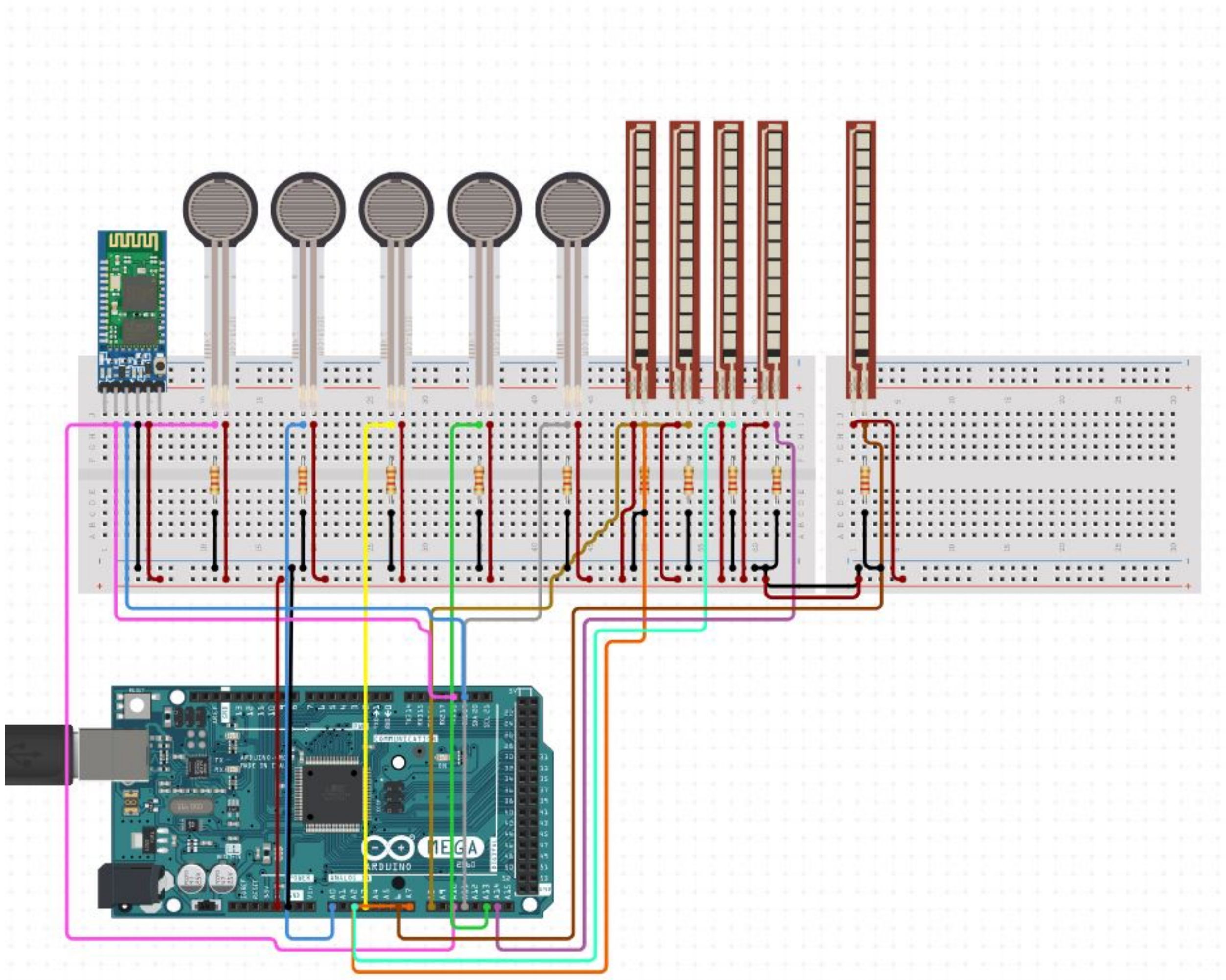


Figure 39. Schematic for direct component connections to Arduino