Next-Step: Software Design Document

Jett Koele
Benjamin Huntoon
Naima Ontiveros
Kendall Callison

Sponsored by:
Jack R Williams & Zachary Lerner

Mentored by:
Scott Larocca

Version: 2.0

Table of Contents

Introduction

Over 7.5 million individuals in the United States face mobility impairments that hinder their ability to walk and engage in physical activity. Physical therapy (PT) is a widely used treatment to help restore movement and strength, but a significant challenge remains which is sustaining patient engagement throughout the rehabilitation process. Many patients struggle to stay motivated, leading to incomplete therapy sessions and suboptimal recovery outcomes.

To address this challenge, the Biomechatronics Lab at Northern Arizona University (NAU), under the direction of Dr. Zachary Lerner and Dr. Jack Williams, developed OpenExo, a fully open-source exoskeleton system. OpenExo provides real-time biofeedback through an open-source Python API, allowing researchers and therapists to monitor and adjust therapy sessions. However, OpenExo lacks an interactive component to maintain user engagement, which is crucial for maximizing therapy benefits.h

Our capstone project focuses on addressing this engagement challenge. We aim to develop a gamified rehabilitation training tool that interfaces with the OpenExo system through its already-built Python API. We will add this component to the Python API to communicate with Bluetooth sensors and built-in sensors to collect real-time data and provide interactive, game-like feedback to patients as they use the exoskeleton for different exercises. By transforming rehabilitation into an engaging experience, we hope to improve patient motivation and enhance the overall effectiveness of therapy. To make the tool more accessible and encourage collaboration, we are making it open-source. This will let developers, therapists, and researchers customize and improve it to fit different rehabilitation needs. With an open-source approach, we hope to build a community that keeps the tool innovative, flexible, and available to everyone.

A clear understanding of the key requirements is needed for the program to meet its intended purpose. The following section summarizes the main requirements of the project that will guide the design and development process.

- User-level Requirements
    - Users interact with the system through the exoskeleton sensors that convert biofeedback into game inputs for Unity and Pygame-based rehabilitation games.
    - Game controls are made for specific body parts and movements to avoid overwhelming the user.
    - Users should receive real-time audio and visual feedback during gameplay to keep track of their progress and to maintain engagement and user motivation.
    - The game should have intuitive and low-impact movements to accommodate users who are not familiar with games.
- Functional Requirements

    - Input/controls: The program should be able to collect data from the exoskeleton sensors and translate it into game controls.
    - Real-time feedback: Audio and visual cues for the users to track progress.
    - Session statistics: Success rates should be displayed to the user and sensor thresholds should be recorded.
    - Documentation: The code should be well-documented and easy to understand and modify.
- Performance Requirements
    - Response time: The system should be able to provide feedback at a rate of at least 500 Hz to ensure effectiveness
    - Game launch: The games should be able to start within a minute of opening the application.
    - Ease of use: GUI should be intuitive for both the users and the researchers.
    - Sensor usage: Focus on two active sensors with real-time data collection.
- Environmental Constraints
    - Testing environments: Sessions take place in a controlled environment, which can be on a treadmill, with safety protocols to avoid falls.
    - User limitations: The system accommodates users of different sizes and experience levels by using simple game actions.
    - System compatibility: The system supports multiple operating systems such as Windows and MacOS.
    - Hardware capability: The exoskeleton has a limited battery life of 30 minutes and has motor torque limitations.
    - System compatibility: Bluetooth communication is used for real-time data transmission.

Implementation Overview

The system uses an exoskeleton brace with interactive rehabilitation games. The Bleak Python library allows Bluetooth communication so the device can send movement and pressure data to the researcher's computer. The gamepad converts this data to controller inputs, enabling seamless interaction with the games. On the game development side, Pygame is used for smaller games while Unity is used for more complex features that use C# to manage game logic. The system follows an Observer design pattern for games to respond dynamically to the researcher inputs and user movements. The existing framework will be used, but modifications will focus on integrating game interactions while maintaining the core data collection and processing functions. This allows the researchers to monitor the session data while enabling real-time user engagement through an interactive gameplay experience.

The following provides a more detailed overview of each component and its integration within the system:

- Exoskeleton Brace:
  - Users will wear an exoskeleton brace to monitor movement and pressure exerted by the user. The brace's core function is to boost the user's mobility while wearing the brace, collecting and transmitting the user's movement data through a mounted Bluetooth transmitter. The data is then collected by the researcher for display and processing.

- Bleak Python Library
  - The Bleak library is used to connect the researcher's computer to the user's brace. Bleak allows for efficient and rapid connection between devices and handles the transmission and reception of user data from the brace. Through this transfer of data, researchers can collect and process the wearer's session statistics to optimize the user's rehabilitation session in real time.
- vGamepad
  - The vGamepad library is a Python library which allows non-standard game inputs to be reinterpreted as a gamepad input. In the context of this project vGamepad serves as the interpreter between the user's input and the games developed for them to play during their session. Specifically, user pressure data, collected in the foot of the brace, is interpreted as Xbox controller trigger inputs. The cGamepad interpretation serves two key purposes, the first being that as a controller input, the user's movement can be used to interact with the game displayed in real time, and the second being that controller triggers allow for a range of input. This range of input, (specifically the range 0-255) can be used to simulate difficulty within the game as it can be set higher or lower as a pressure threshold for the user to meet before being able to move the game's avatar.
- Pygame
  - Pygame is the first of two tools chosen for game development. Development in Pygame was chosen for its simplicity, ease of use, and accessibility in a python development environment. The Pygame library generally only permits development of two dimensional games and has the capacity for collecting controller input. These factors allow us to make real time responsive games that are easily understood and enjoyed by the user.
- Unity

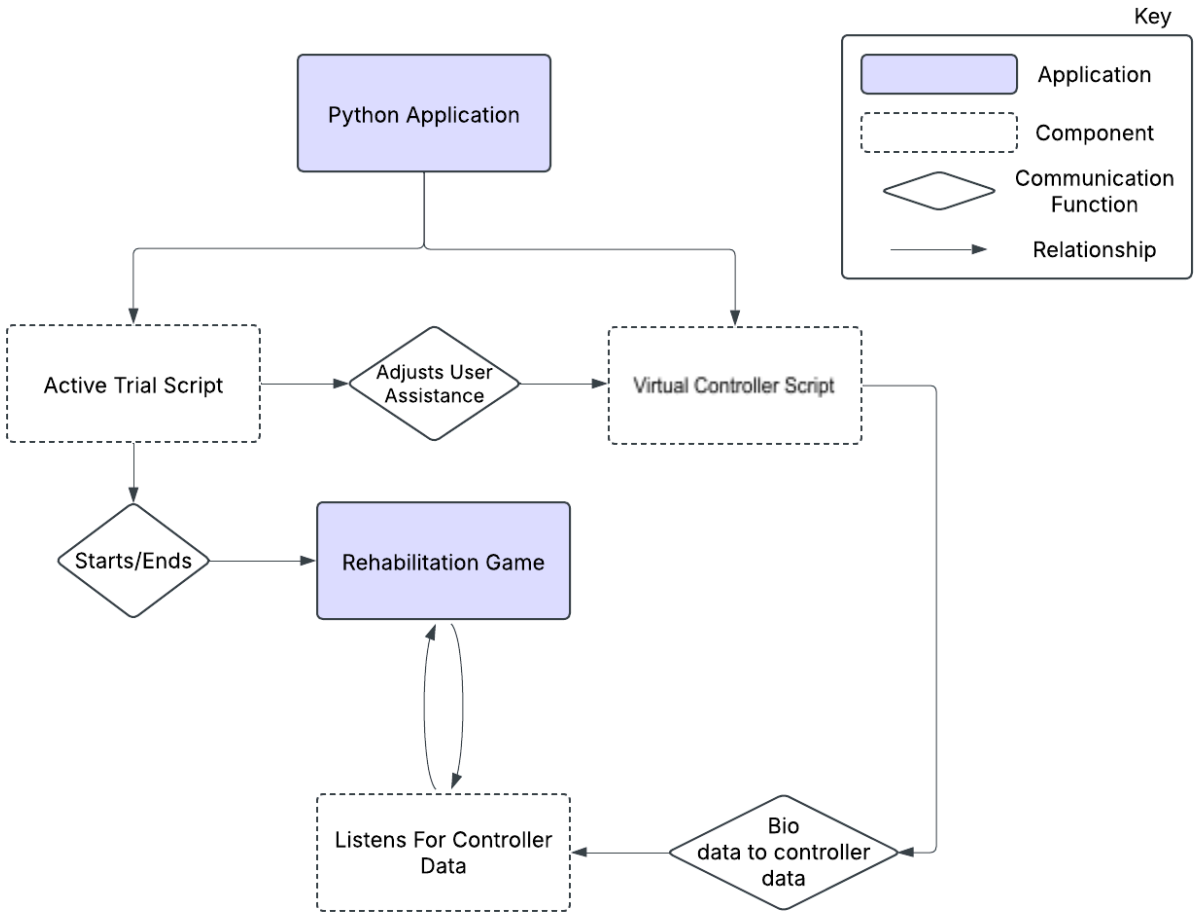
        - Unity is the second of the two game development tools selected for designing games for the brace. Unity is a robust game development environment which allows for a wide variety of games and has the capacity for controller input. The wide variety of games Unity can be used to develop combined with its ease of use and controller support allow for the development of responsive games the user will enjoy.
    - Programming Languages
        - Python is the primary programming language used throughout the project. The entirety of the existing code base handling data transmission and reception was created with Python. Python is easy to use and has a wide variety of existing open-source libraries available for use to aid in development. These libraries include, but are not limited to Pygame and Bleak.
        - C# is the language Unity mostly uses for game development. This allows for fast processing speeds and in the Unity environment, includes a solid selection of development tools for designing games. The Unity environment also allows for seamless integration with the Python portion of the project.
    - Framework
        - The project framework largely already exists. This framework consists of data collection, processing, transmission and reception surrounding user input observed by the leg brace. The games developed for the existing system will only modify the existing front-end of the data processing portion of the code base. This means researcher's will be able to view the original statistics, while also granting the option of selecting and displaying a game for the user. This will allow for a dynamic display that researchers can choose on a case by case basis. The backend of the existing code base will be modified to allow the interception of user data to the vGamepad. This modification will allow for the user input to interact with the games without any major modification to any other backend component.
- Design pattern
    - The games developed will largely follow an observer design pattern. This means the games are directly controlled by the researcher's specifications and the user's movements. Without interaction from the researcher the games should start in default mode and without interaction from the user the game will remain still. Only with observed interaction will the games demonstrate responsiveness.
- Main Components Overview
    - Overall the only added component to the existing project structure is the gamepad and the games themselves. The existing code base remains largely the same, while

user input is collected both for the researcher to view and as an interactable game for the user. The front-end component will be dynamic allowing for researchers to select and run games to certain specifications while the backend largely focuses on interpreting the brace's data and using that to interact with the game in real time.

Overall the rehabilitation game system is designed to run concurrently with the existing program meant to monitor user movement. Data collected from pressure sensors within the brace is taken and used within the game in real time to help engage the user with their rehabilitation session. The games implemented are designed to be intuitive and easy to play to enable the user to focus on their movement and the directions given by the researchers. The architecture of this system is meant to reflect each of these aspects, emphasizing a clear flow of user movement data to ensure timing within the game is not detrimental to the overall session.

Architectural Overview

The overall architecture of the implemented game system is intended to only extend the existing brace monitoring system without majorly interfering with the existing structure. As shown in the diagram below, the game system only interacts with the transmission of data being collected from the brace, allowing full operation of the previous existing system to occur in the background of a rehabilitation session. Once a session is started the researchers will be able to view the session progress and statistics while a game is selected and then displayed for the user. As the user moves throughout their session, movement data is sent to the researcher data module as well as the game module via the virtual gamepad. That data is then used to play the chosen game, encouraging the user to continue their session. Once a session is complete the game and the existing research module will present session statistics.



Python Application Architecture

The Pre-Existing Application

The application, as it was given to the team, already has some development and features. Its main role for existence is to display and save data that is created from a user wearing an exo-skeleton suit. This pre-existing python application can be divided up into a few different categories of components to help illustrate its inner workings. These components are separated into separate folders with the respective names below.

Views

The Views folder contains all of the user interface windows. Each file within here is a separate window that uses the Tkinter.py library to create buttons, labels, and graphs. These different windows give the researcher access to buttons that control every aspect of the application while also giving real time visual aid of the data being created by the user. All in all, the scripts in Views are the front end of the application, acting as a control panel for the exo-skeleton.

Device

The next important folder is Device. This folder contains the backend processing of the entire application. The contents of this folder are designed to connect to the exo-skeleton via bluetooth, specifically using the Bleak python library, from within the “exoDeviceManager.py” script. Other functionality includes processing all the data coming from the bluetooth signal. For this, “realTimeProcessor.py” is used to bring together all of the other scripts from within the Device folder. It is a centralized point where all the data from the exo-skeleton meets with all the other backend scripts. In other words RealTimeProcessor is a “mother” class that is used to house all other backend components/classes.

The RealTimeProcessor class is also the class that is in direct communication with the frontend user interface. For a button to call a function, it must do so via the current RealTimeProcessor object.

Data

The Data folder is the most simple of the three described, but provides an essential feature. After a trial is ended, exo-skeleton data is saved and then stored in a newly created .csv file. The contents of this file are divided into columns. Each column represents a different variable from within the application. A new row of data is saved after each process cycle so that data can be seen changing overtime. New variables that are important to view after a trial has ended are to be added here, in the Data folder.

The Application Additions

Alongside the overall structure of the pre-existing application, our team must create the functionality for games. This feature will require new user interface, communication of data to a said game, and the games itself. Our team's solution builds upon the current application with some modifications and the creation of a few files.

Active Trial

The Active Trial window, found in the Views folder, had already existed beforehand. This class not only updates a window to display the exo-skeleton data in real time, but also gives a researcher access to key controls to the exo-skeleton. From this point, buttons will be added to the user interface that give access to different games. Buttons such as select game, start game, and end game are all necessary. They will look into the Games folder of the application and start up a new game window that can then be ended at any time. Aside from selecting, starting, and ending a game, a researcher must also have access to changing a user’s difficulty level. In doing so, communication to the Virtual Controller class, discussed below, is essential.

Virtual Controller

The Virtual Controllerclass is the first new script needed for our solution. This class will be described, along with all the others, in more detail below. In terms of architecture, the Virtual Controllerclass acts as a middle man between the python application and the running game.

More specifically, it takes real time data from the current RealTimeProcessor object and converts it to virtual gamepad data, that is then understood by the game. Essentially mapping the exo-skeleton movements to a virtual gamepad. In doing so, the Virtual Controller class will also take into account the assistance/difficulty level set from within the Active Trial UI. This will be a one way communication from the application to the current running game.

With this entire application being open-source, our client wants development to be as easy and accessible to everyone as possible. By creating the virtual gamepad, one can develop on any existing game platform that provides gamepad support. Furthermore, a developer can then test games using a physical gamepad, rather than needing an entire exo-skeleton. All in all, making development more accessible to everyone.

Game Application Architecture

As stated previously, the games will be developed in both Unity and pygame. Each game is designed to do the following: Listen for gamepad inputs that are being simulated by the python application, convert these inputs into some sort of live feedback based on how the game is interpreting this information and lastly give the user some sort of task to complete or a goal set by the researcher.

Firstly, the main reason to have each game simulate a gamepad is to allow for the future development of this open-source software and allow other developers and researchers to add more controller inputs to the device based on other movements. It also allows the game developers to be a bit more creative and they will not need to understand the behind the scene work of the python application, but rather only consider what inputs the game will receive from the device. In our case, we designed the games around the triggers on a controller as it will give us a range from 0 to 100% trigger pull. This design is purposeful as the user will want to see how close they are to the set goal on a scale of 0 to 100%.

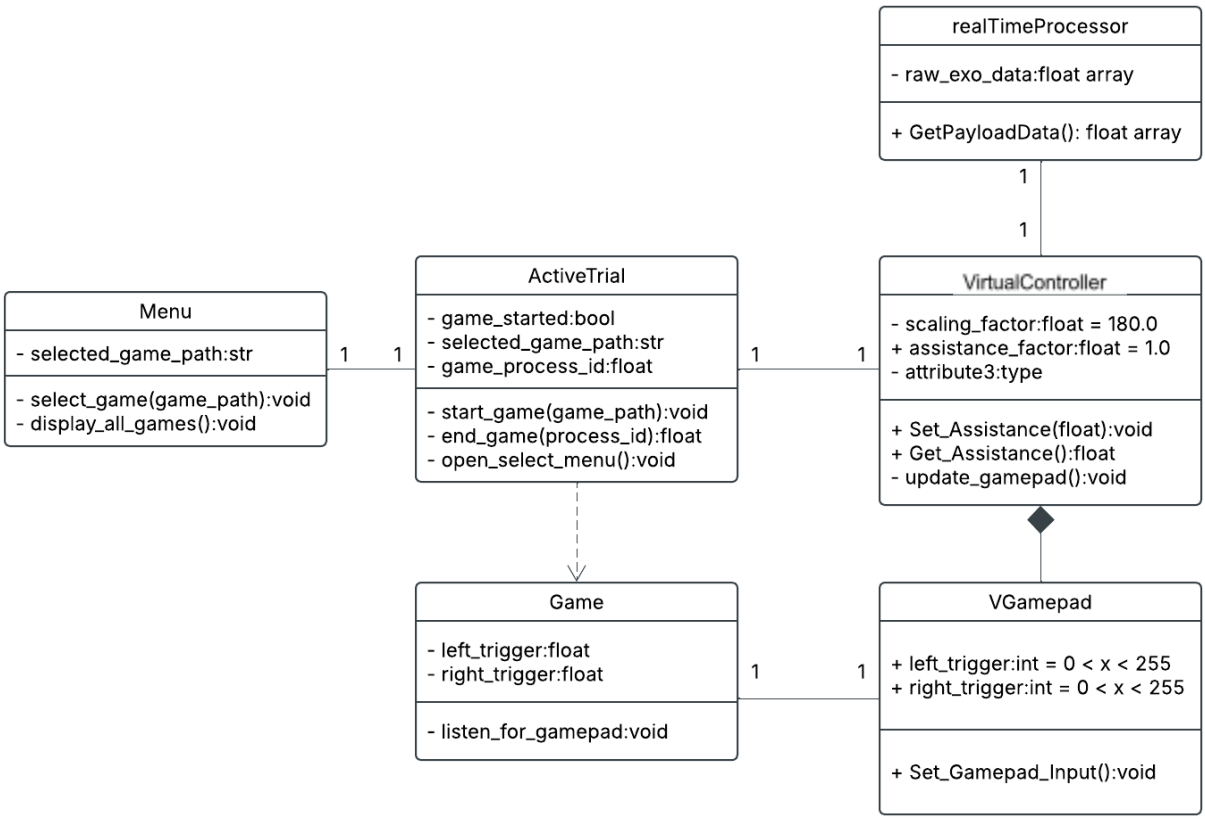
Secondly, the purpose of having some sort of live feedback is to encourage the user to push themselves harder and not discourage them when they fail. Having a bar that fills up as the user takes a step until they meet their goal is definitely a must in every game design. As stated, the goal is to encourage user engagement and promote a rewarding experience as they proceed through their physical therapy experience.

Lastly, having the user able to have an overall goal to meet besides the individual steps that will be counted, plays a bigger role in the game design process. This goal will generally be set by the researcher and will be something that should be achievable for the user to complete. These goals should be something that can be tracked by either a scoring system or some sort of counter that will increase as each individual step goal is met. Having an overall goal is good for the user as they are focusing on each individual step, they will be able to see how far their overall progress is becoming more accurately.

Module and Interface

The following will be a list of more detailed sections of each part of the project. As the development progresses continues we will stick to using these base guidelines to fully utilizing the python application and creating a fully functional game that properly utilizes the python application. As seen in the UML diagram below, this is the overall structure of the project.

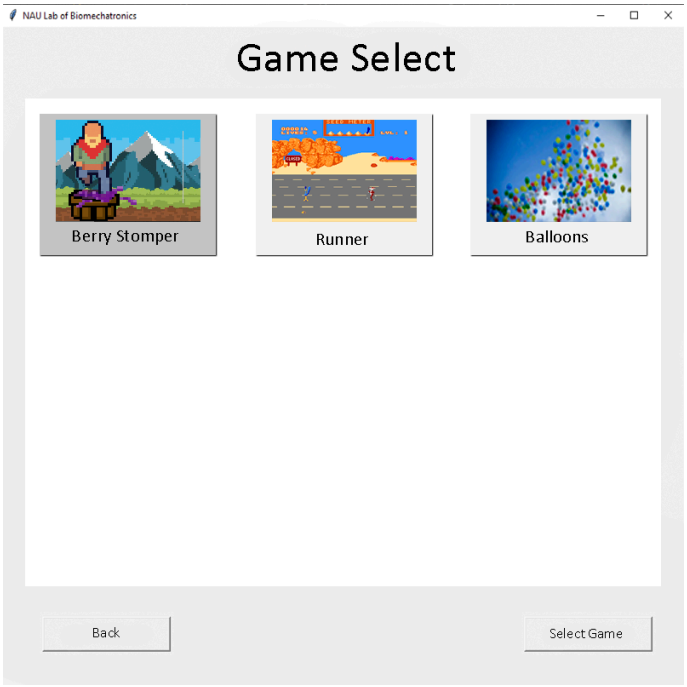
Complete UML Diagram



Active Trial Class

As it stands, before the development of the Next-Step addition, the Python Application already has some functionality set up. Though not entirely in the scope of this document, some key components are being utilized for the functionality to play games. To start, one must look at the Active Trial class. While conducting a trial for the user, the ActiveTrial class gives graphical visual feedback and a UI to the researcher. This class is to be added upon in a few key ways. New functionality such as a menu to select, launch, and end any downloaded game is required. This will give the researcher full control over what games are played, and for how long.

Starting with the Games Menu button. After being clicked will create a new Menu object. This object will open a new window for the researcher to view. Inside, all the games a researcher has downloaded will be on display. Any of the games will be clickable causing the game to become highlighted one at a time. When finished, the researcher can either x-out of the window, leading to nothing happening, or pressing a select button that will save the file path of the highlighted game.



Menu
- selected_game_path:str
- select_game(game_path):void - display_all_games():void

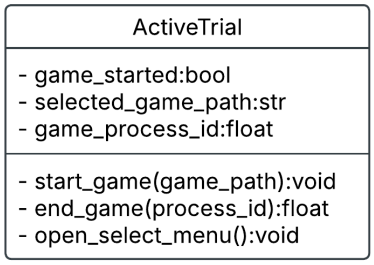
Using the Start button, only after a game has been selected will the researcher be able to click the button. Once a game is selected, the start button will be clickable. In doing so will use the file path found by the Menu object to boot up the game. The game's process id will be saved as well.

Lastly is the End button. This button will be greyed out until the Start button is clicked. This ensures that the button is only useful when there is a game running. When active and clicked, the button will simply grab the already saved process id of the running game, and end it. After doing so, the button will become unclickable again until the next Start button press.

Adding these buttons/controls within the Active Trial class is important. This keeps the program consistent, while also keeping the functionality of past development efforts from the Biomechatronics Lab. Researchers are given the ability to access and run games, while also seeing the original, more data oriented, user interface.

In addition to giving the researcher game-start and end functionality, one must also be able to adjust a user's difficulty level for reaching goals. This requires a one way communication

with the Virtual Controller class. The Active Trial class will prompt the researcher and tell the Virtual Controller class to scale values accordingly. To keep the system simple for future development, the communication between classes will be kept to one way. The Active Trial class must execute functions within other objects, but it must not directly modify anything outside of its own class. This ensures abstraction within the program's classes. Leading to an easier future development process.



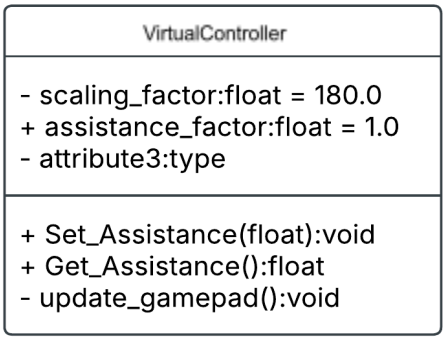
Virtual Controller Class

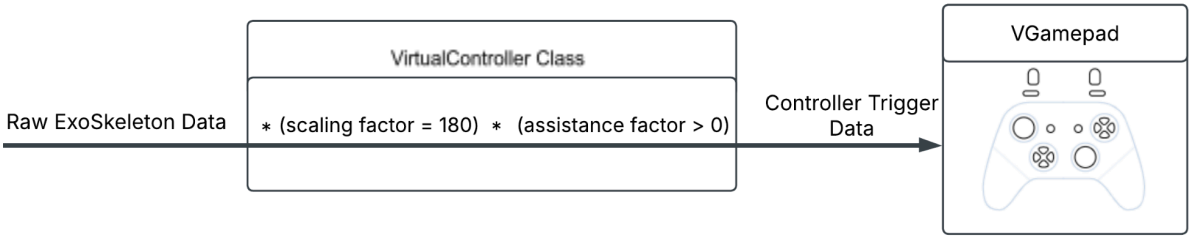
This leads into the next important component/class of the program, the Virtual Controller class. The Virtual Controller class works by taking in biometric data from the user's exoskeleton suit, and converting it into gamepad inputs for a virtual controller/gamepad. The Virtual Controller class is the only communicator with the running game at any given time. The game simply sees a controller/gamepad and begins reading its inputs.

Within the Virtual Controller class a few key elements are required for proper functionality. To start, the class must create a virtual gamepad. Using a python library such as "vgamepad" new virtual gamepads can be created. With the gamepad/controller created, all that is left is to constantly update the virtual gamepad with inputs. The information generated by the exoskeleton is automatically scaled to have the user's average movement be a float of "1.0". This means that a user's normal step would equate to "1.0", where a heavier step might be "1.7" and

so on. On the other hand, a gamepad's triggers operate on a different scale, specifically, "0 to 255". Because of this, the raw data from the suit must be scaled. To do so, a few variables are needed. The "scaling factor" and the "assistance factor". Both of which will be multiplied to the raw data to achieve a new gamepad trigger pull. The scaling factor will be set to "180". This allows for games to visually show if the user is either under or over achieving. The next variable is set by the researcher. As explained above, this variable will be augmented via the Active Trial user interface. As a default, this value will be "1.0", meaning that there is no assistance given. Although the value can be set anywhere above "0.0".

This middleman step is necessary for the sake of open-source development. Because the exoskeletons are currently in a research and development phase, developers will not always have access to the equipment. Via the Virtual Controllerclass, the equipment is not necessary. A developer only needs a game controller to be able to develop and test functional games for the program. This opens the door to anyone in game development to have the ability to create different games and experiences for users. Because the games only need to listen for controller inputs, developers are also granted the ability to develop on any game engine they please. This allows a wider demographic of developers to help contribute games to the program.

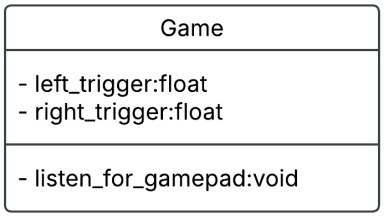




Games

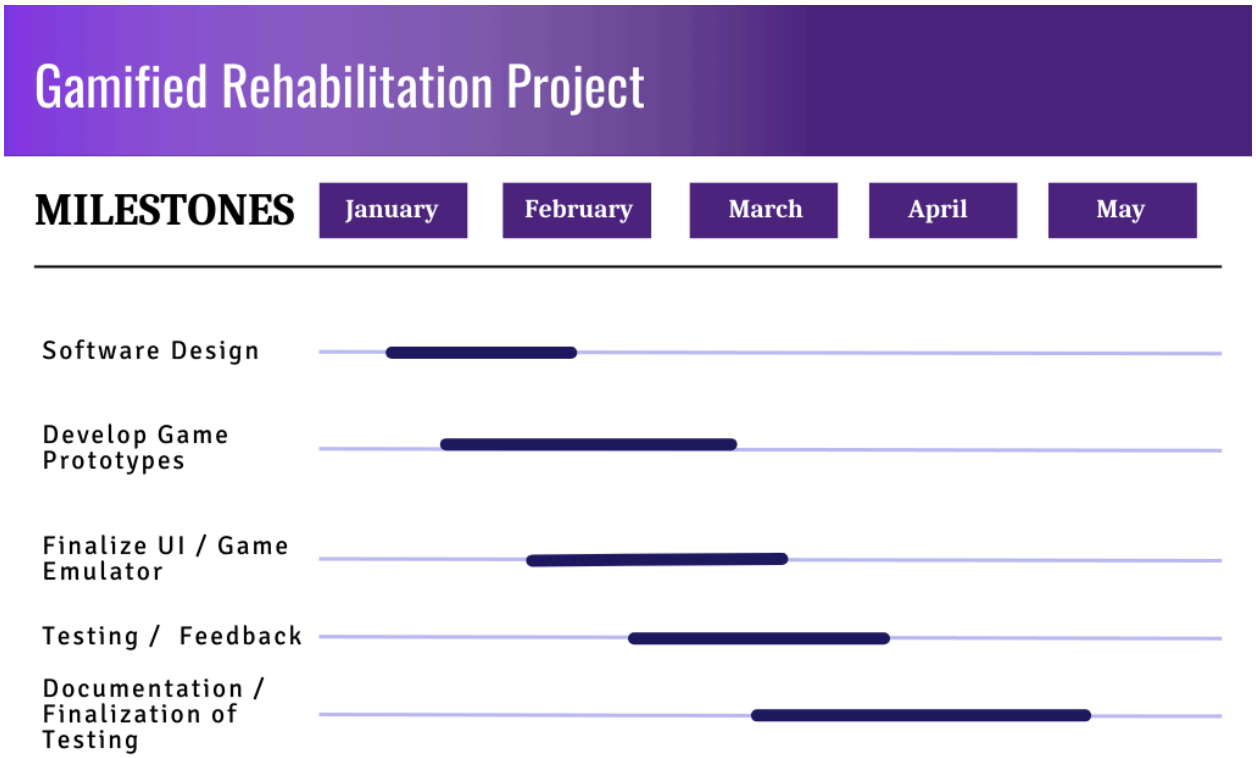
When it comes to the different components of the games, there are only a few that are technically necessary. Most of all is controller/gamepad support. More specifically, a game must

be able to read controller/gamepad trigger pulls. The process in which to read the input will change depending on the chosen game engine of the developer, but is a universal feature amongst different engines. After having the trigger data, a threshold should be set to a trigger pull value of 180. Anything past 180 should display a “reward”, for this is the semi-arbitrary value set to be the “scaling factor” from above. The value of 180 represents a user’s goal being reached. When approaching the goal value, games should display how much further one must go in order to reach a necessary goal. This will be displayed differently from game to game, but is key to keep user engagement and prevent discouragement.



The next big piece to any game being developed is that each game should not have an end. Games should loop over and over until the researcher ends the game from the Active Trial user interface as described above. With these key technical features being implemented, the rest of the game is up to the creativity of the developer.

Implementation Plan



The graph above shows the overall plan for the development process of the games and front end UI of the python project. We have 5 months to develop a fully functional front end and 3 fully developed games. This can be achieved but we need to set some parameters and expectations before getting started with the development. Below shows the expectation of each phase of development for the project.

- Starting Development
  - During this phase, the team will be focused on creating a fully functional game that will be developed while getting weekly feedback from our clients. The games will include a minimum of a start screen, settings, menu, and gameplay by the end of this phase.
- First Implementation of Games and UI
  - This phase will consist of combining the UI and ensuring games can be interacted with properly while the Python application is running and properly combining the game features with the Python application.
- Testing Phase
  - The team will start to test the project in multiple different environments as well as start to find and fix bugs and errors occurring in specific situations. The team will also be expected to work with the client to start doing testing with the suit and with a real user.

- Finalize fully functional project (1.0)
    - This is the final step and will include completing all documentation for our project as well as finalizing any bugs and errors that have been occurring during the testing phase. After this phase 1.0 is officially released

During the development of the project, each member of the group will be responsible for a specific part of the project. Listed below are the responsibilities of each member and what is expected of them during each phase of the project.

- Starting Development:
    - Jett is in charge of developing the Running Game as well as keeping the team on schedule and assisting in finding solutions to problems during the development of each project.
    - Naima is in charge of developing the Balloon Filling Game as well as updating the Linear App to keep constant updates on the project and any issues that come up during the development.
    - Ben is in charge of the finalization of the Grape Smasher Game as well as updating the Linear App to keep constant updates on the project and any issues that come up during the development. He is also responsible for assisting other team members with issues and problems that come up.
    - Kendall is in charge of developing the front-end UI of the Python project as well as updating the Linear App to keep constant updates on the project and any issues that come up during the development.
- First Implementation of Games and UI:
    - Jett and Naima will be in charge of ensuring the games are all properly prepared for Python application integration and properly documenting all issues that occur during this phase.
    - Ben and Kendall will be responsible for ensuring the Python application is properly able to launch the UI and find and launch game applications from within the application. Any issues that come up should be documented and fixed during this phase.
- Testing Phase:
    - Jett and Naima will be in charge of testing the games in different environments and ensuring all bugs and errors are fixed in most environments. Also ensuring any bugs within live testing are handled and reported in the Linear App.
    - Ben and Kendall are responsible for testing the Python application in different environments and ensuring it works as intended in most environments. Also making sure any issues experienced with live testing are handled and reported in the Linear App.
- Finalize fully functional project (1.0):

- Jett, Naima, and Ben will be in charge of creating documentation for all games including how they are designed, how to adjust settings, basic UI management, etc.
- Kendall will be responsible for documenting the use of the gamepad controller as well as the UI interface that will be used in the Python project.

Conclusion

This Software Design Document provides a clear plan for building the project, ensuring that all design decisions align with the system's goals. It explains the structure of the software, how different parts work together, and what technologies will be used. Some key choices include using Unity and Pygame for game development, integrating biofeedback from exoskeleton sensors, and providing real-time feedback to improve rehabilitation exercises. These choices help create an interactive and engaging experience for users.

This project is important because it makes rehabilitation more effective and engaging. Traditional therapy can be repetitive and unmotivating, but by turning exercises into a game, users stay motivated and committed to their recovery. The system allows researchers to adjust settings, track progress, and customize the experience to fit each user's needs.

The purpose of this document is to serve as a guide for development. It helps keep the project organized, ensures consistency, and provides a reference for future improvements. With a strong design in place, we can build a reliable and useful rehabilitation tool that benefits both users and researchers.