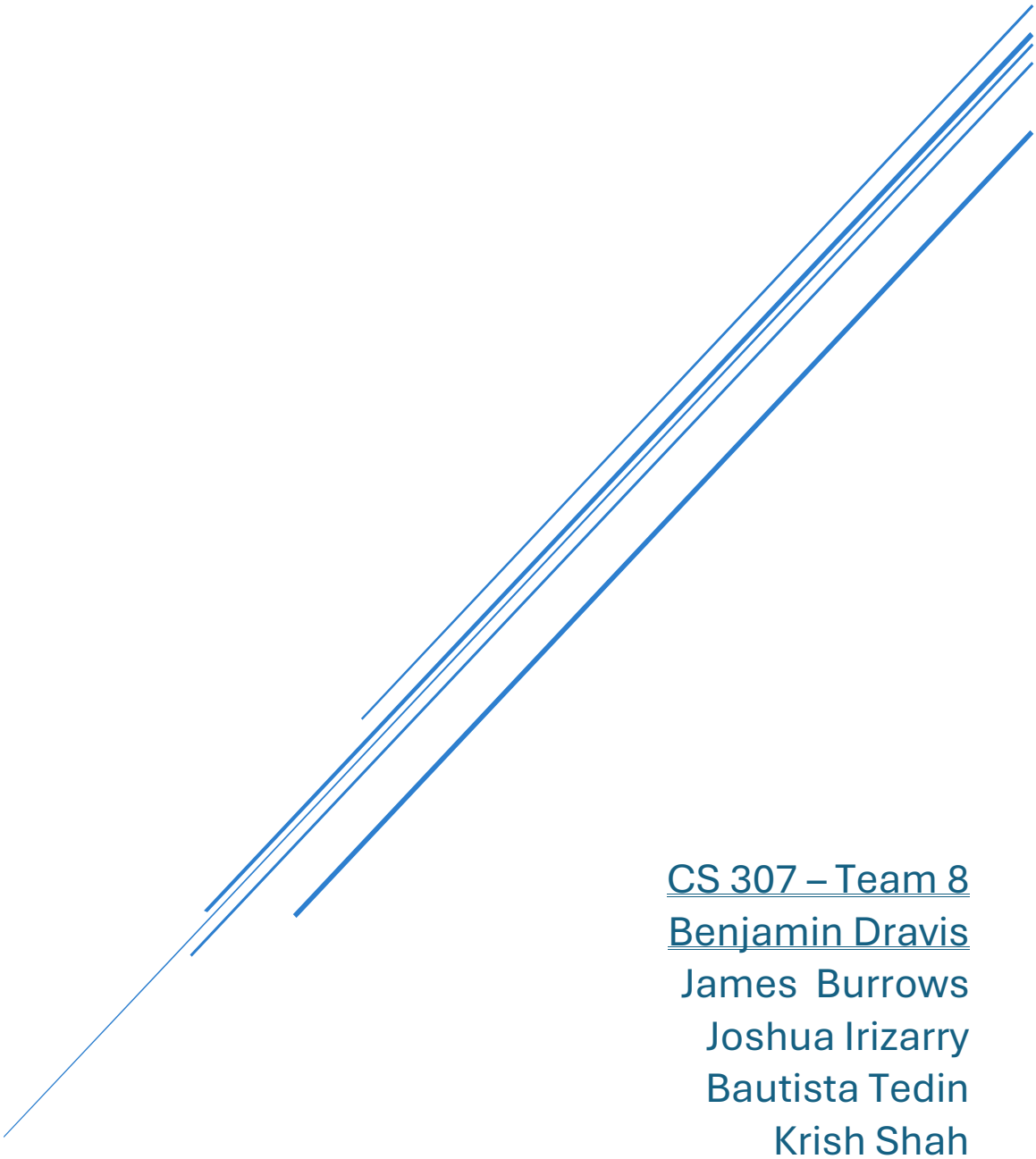
POCKETPAD

Controller Emulation Application

Design Document



CS 307 – Team 8
Benjamin Dravis
James Burrows
Joshua Irizarry
Bautista Tedin
Krish Shah
Jack Fang

TABLE OF CONTENTS

PURPOSE

As technology has advanced into the modern age, gaming has become more and more prevalent in the daily lives of people. For some, it is their job and source of income. For others, it is something that they can do to connect with their friends, and for the rest, it is something that can bring them a moment of joy amidst their day. However, regardless of what circle someone might find themselves in, everyone requires access to the technology required to do so. They need access to the systems and controllers, and recently that has become an expensive venture with systems costing $300 to $600 dollars for a system and another $80 per controller. That does not even consider the price of the games themselves, but it is already a very expensive venture.

As a solution to this problem, they have found ways to circumvent the need for several systems or controllers through the use different platforms such as Steam or Dolphin allowing them to play games that would have once been across a wide variety of platforms on a single accessible platform, but that does not eliminate the need or want for controllers. Gamers have preference of the types of controllers with how they are playing their game, which led to the creation of controller emulation. Controller emulation generally involves using hardware such as adapters to connect a controller to a system it was not intended to be connected to. It has allowed gamers to play some of their favorite titles using their preferred and favorite controllers.

However, controller emulation still has a significant price tag attached to it for a still flawed system. PocketPad seeks to change that through a free iOS application allowing users to emulate different types of gaming controllers with the portability and accessibility of their phone/iPads. Users will no longer be reliant upon different hardware to connect their emulated controller since PocketPad will provide connections to computers both over network and Bluetooth allowing them to play games through services such as Dolphin without the need for wiring or other hardware, and they'll also be able to view and test that connection using a Python interface.

The overall purpose of PocketPad will be to provide users with free and easy to access emulation services without having the financial endeavor to physically have different types of controllers to play certain games, but rather, they could use PocketPad to emulate their favorite controllers from the comfort and ease of a singular iOS application when playing their favorite titles from a wide range of games.

FUNCTIONAL REQUIREMENTS

Users can access guides and information about the iOS application

AS A USER OF THE APPLICATION…

- I would like to have a tutorial for the application so that I can have a guide for general understanding of how to use the application and its numerous different functionalities properly.
- I would like a help menu that will provide quick access to more in-depth troubleshooting guides and FAQs so that I can quickly resolve any issues or questions I may have when using the application.

Users can emulate several different controllers' features in the iOS application

AS A USER OF THE APPLICATION…

- I would like to be able to have an emulated a movement joystick like that of the left stick of some common controllers so that I can control the movement of my characters in the different games that I am playing.
- I would like to be able to have an emulated perspective joystick like that of the right stick of some common controllers so that I can control the perspective of my characters in the different games that I am playing.
- I would like to be able to have an emulated directional pad so that I can use the directional movement functionality of the D-Pad needed for the menus in some newer games and game functionality in most older games.
- I would like to be able to have emulated bumpers so that I can use the functionality that games I like to play have tied to the left and right bumpers on most common controllers such as blocking or displaying emoticons.
- I would like to be able to emulate the diamond of buttons so that I can interact with the different menus and activities such as going through dialog, attacks, option selection, and more that are commonly bound to those buttons and are necessary to play the games.
- I would like to be able to have emulated triggers so that I can use the functionality that games such as shooter or adventure games commonly have bound to those buttons for interaction with certain game mechanics necessary to play certain games.
- I would like to be able to emulate a settings button on controllers so that I can access the settings on certain games that I am playing in order customize my in-game settings whilst playing games.
- I would like my emulated controller to include an option for motion controller so that I can run games such as Mario Kart which can be played with or require motion control features rather than buttons or joysticks. (i.e. rotating the phone to steer).
- I would like my emulated controller to have vibrational haptics so that I can access and use the immersive tools that exist in the games that I am playing as well as play certain games or minigames such as those in Mario Party that rely on vibrational haptics to function properly.
- I would like my emulated controller to have sound haptics so that I can access and options for the immersive tools that exist for setting the environment or style of the game that I am playing as well as play certain games or minigames that make use of the sound haptics to function properly.

Users can emulate several types of controllers in the iOS application

AS A USER OF THE APPLICATION…

- I would like to be able to construct custom controllers using a custom and modular layout of buttons, bumpers, joysticks and more so that I can create and experiment with my own unique style of controllers when playing games.
- I would like to have an accessible controller file within the application to emulate an Xbox controller so that I don't have to spend the time creating my own custom version of an already pre-existing controller for easy access to it.
- I would like to have an accessible controller file within the application to emulate a PlayStation controller so that I don't have to spend the time creating my own custom version of an already pre-existing controller for easy access to it.
- I would like to have an accessible controller file within the application to emulate a GameCube controller so that I don't have to spend the time creating my own custom version of an already pre-existing controller for easy access to it.
- I would like to have an accessible controller file within the application to emulate a Nintendo switch controller so that I don't have to spend the time creating my own custom version of an already pre-existing controller for easy access to it.
- I would like to be able to import and export controller configurations so that I can share my custom layouts with friends or use their presets.

Users can personalize their emulated controllers in the iOS application

AS A USER OF THE APPLICATION…

- I would like to be able to be able to make some modification to the types of controls so that I can use my preferred style of controller such as split versus conjoined D-Pad with my chosen controller.
- I would like to be able to customize the color layout of any given so that I can apply my own style and color preferences to at a given time to the controller that I am using.
- I would like to be able to customize the name of my controller or device when I connect so that I can clearly identify which controller is connected to me and is accepting my inputs.

Users can connect to a given computer/device through the iOS application

AS A USER OF THE APPLICATION…

- I would like a selection option to be able to connect to my chosen device via network so that I can play my chosen games with more people than I would be able to with connections such as Bluetooth.
- I would like a selection option to be able to connect to my chosen device via Bluetooth so that I can play my chosen games with higher connection speed, allowing the controller inputs to run faster than it would with other connections such as network connections.
- I would like to be able to see the eligible devices to connect to in the area in a nice menu so that I can select which specific device that I would like to connect to.
- I would like to be able to disconnect from what would be then the connected device so that I can stop gaming at any time or potentially switch between different devices if there is another game that I would like to play somewhere else.

Users can access different quality-of-life features in the iOS application

AS A USER OF THE APPLICATION…

- I would like to be able to access a library of the commonly used controller files that are on the market as well as the custom controllers that I make myself so that I don't have to manually remake them each time I want to switch between the types of controllers I want to use and simply be able to save and load them as I want.
- I would like to have a settings menu in the application so that the different options for customization and functionality are easily accessible at any given time as well as not taking up space to ruin the layout and functionality of the emulated controller.
- I would like an option to use my phone as a remote mouse/keyboard hybrid in addition to a controller so that I can navigate game menus or settings that require text input.
- (If time allows) As a user of the application, I would like to be able to sync my controller settings across multiple devices using a cloud backup so that I can access my custom layouts anywhere.
- I would like a way to fine-tune dead zones on my emulated analog inputs so that I can eliminate unintended movement or input drift when I am trying to use the controller.

Users can edit characteristics of their emulated controllers in the iOS application

AS A USER OF THE APPLICATION…

- I would like the ability to emulate turbo button functionality so that I can automate repeated button presses in certain games.
- I would like to enable macros for specific button combinations so that I can execute complex in-game actions with a single press.
- I would like a way to fine-tune dead zones on my emulated analog inputs so that I can eliminate unintended movement or input drift when I am trying to use the controller.
- I would like to adjust the sensitivity of my emulated analog inputs so that I can fine-tune the speed and responsiveness of movement based on those preferences.
- (If time allows) I would like an option to set up multi-touch gestures to trigger specific in-game actions, allowing for more advanced inputs beyond standard buttons.

Users can view a Python interface displaying different information on their computer

AS A USER OF THE APPLICATION…

- I would like to be able to be able to pull up a GUI on the connected device so that I know that I can see the processes and what is happening on the server side of the connection as well.
- I would like there to be a portion of the GUI that acts as a connection list so that I can see which devices, if any, are connected to the given server at any moment.
- I would like to be able to see a mock-up of my controller on the GUI so that I can see that it is properly recognizing my controller and the customizations that I potentially put into it beyond the act of simply connecting my phone.
- I would like the mock-up of the controller to highlight the button, joystick, feature, or whatever input that the user is putting into their emulated controller displayed on the GUI so that I can see that my controller is properly sending inputs to the device that it is connected to before playing a game.
- I would like a clear way to differentiate between different controllers and inputs on the GUI so that if I am using the application to play with a group of my friends, we won't get confused by which person's controller corresponds to which when playing our games.
- I would like the application to ask me to automatically switch between different layouts I manually set depending on what game I am playing so that I won't have to go through the process of switching it each time I am wanting to play a different game especially if I intend on playing multiple games in a single session.
- I would like a visual indicator for connection strength displayed alongside the connection list in the QT GUI when using Bluetooth or network mode so that I can troubleshoot input lag or disconnections more easily.

Users can customize the Python Interface on their computer

AS A USER OF THE APPLICATION…

- As a user of the application, I would like to be able to access a settings menu in the computer GUI so that I can toggle wanted or unwanted features from the GUI display to meet my wants/needs at a given time provided by the application regardless of the type of phone/device that I have.
- I would like to be able to customize the GUI's color scheme so that I can make the application personal and to my preferences.

Users can use the Python interface in tandem with their games

AS A USER OF THE APPLICATION…

- (If time allows) I would like to have the option to see an overlay over my games so that I can see my inputs on the screen while playing.

Users can use the client-side application on android devices

AS A USER OF THE APPLICATION

- (If time allows) I would like the application to be android compatible as well as iOS compatible so that I or users like me would be able to use the services and features.

NON-FUNCTIONAL REQUIREMENTS

ARCHITECTURE AND PERFORMANCE

We plan to develop the application with a distinct separation between the mobile client, handled within our iOS application, and the computer server, which users will host locally on their devices. This structure will allow us to effectively divide our work while minimizing compatibility issues between the client and server. Communication between the two will be facilitated using sockets, ensuring seamless data transfer even when there are multiple clients connected. We will implement a TCP-based socket connection to enable fast, real-time communication between the client and server, while securing the connection with TLS to protect user data.

Our iOS mobile application and its client will be built using SwiftUI utilizing its features for developing both a user-friendly application and client to connect with the server. For Bluetooth connectivity, we will use CoreBluetooth, Apple's built-in framework for managing Bluetooth connections. Network communication will be handled with the Network library. Using Swift will also allow us to integrate the networking features that our application requires as well as allowing us to use some built-in iOS features to make the best product possible in terms of service and function.

The server will be written in Python with the socket library built in. It both reads and transmits data sent from our swift client to allow the user to interact with the game that they are trying to play as well as interact with the computer GUI. Like the server, our GUI will also be written in Python with the QT framework, which will allow us to build a robust interface with all the functionality that is needed to ensure good functionality and quality-of-life. Building it using this framework will allow us to implement the robust functionality mentioned in our functional requirements.

USABILITY

Both the iOS application on the iOS device and the Qt PySide6 interface on the computer should be very user friendly and easy to navigate. With all the different features of our final product, it can become unnavigable if we are not careful about what we are doing, so a lot of what we are doing during its development is to ensure the usability of the program. To ensure that we retain good usability, we are going to be careful with how we structure the different features of our applications, which is a huge part of the reason that we are using SwiftUI for this. SwiftUI has a lot of built in functionality allowing us to ensure that our program will work across different sized devices as well as more built-in user-friendly features as well as allowing us to update our computer interface every 10ms for minimal input delay from the iOS application to the QT interface. We also plan on making our application using clean and concise menus as the way to navigate and use different functionalities, and the applications will also have the ability to toggle on and off certain features from those same menus which will help us to ensure the usability of user-friendly nature our team seeks to achieve in our product. Also, we are going to be implementing an in-app tutorial to help to explain a lot of the different features and functionalities of the project as well as general knowledge about how the application can be used. We believe that all these different functionalities will help to ensure our programs' usability.

SCALABILITY

This application will be very scalable in nature depending on the resources of the people using it. Our final product will include the availability of both network and Bluetooth connections. This diverse functionality would be able to handle different numbers of people at any given time. Connection via Bluetooth would have a maximum number of people set to 7 people joining due to preset limitations with Bluetooth connections. Network connections on the other hand would, in theory, be able to handle thousands of people at any given time securely transmitting and receiving data, which is a whole lot more than would ever reasonably be able to connect because this project will be run and hosted locally on a person's personal computer. We plan on ensuring good and secure functionality with the connection allowing for developed scalability within our product.

Going into the future, there is a whole lot of room for growth and development within the project. Initially our intention will be to use our application in tandem with other software such as Dolphin to fully emulate the controller. So, like the current technology implementation it would still be reliant upon outside technology. However, this necessity could be removed in the future given the right investment in time and energy to the project. This would make the application be able to be self-sufficient for its intended function to be used in tandem with games being able to regulate the emulation for different games beyond using the server to send and receive responses, and with this functionality, it could be further developed to be used in tandem with software such as Steam allowing for an even wider variety of games accessible to the user through the use of our application. Depending on the time and effort devoted to this, it could grow in any way the developers see fit or could be useful to the gaming community.

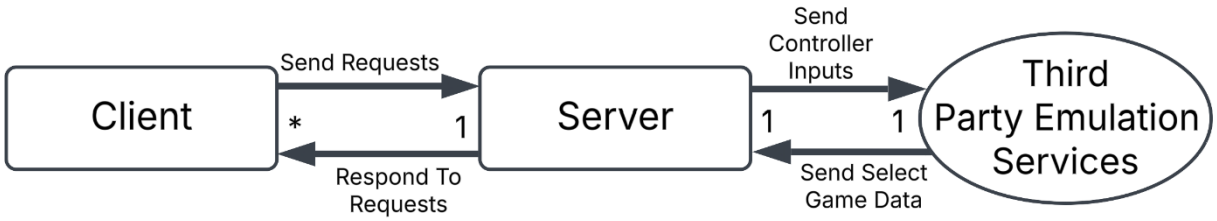
HOSTING/DEPLOYMENT

The application will be hosted in two different ways with the Swift application and pyQT interface being hosted separately. The entire codebase will initially be stored entirely within its own GitHub repository that users can access due to our desire for this to be a free and accessible tool of the gaming community. Using our codebase, the users can then go use our pyQT interface and server locally on their own device so that they or their friends can easily connect using the iOS application that would run and exist on their chosen iOS device. It would allow the users to use their own personal devices/technology in order to run our free application at their convenience.

DESIGN OUTLINE

HIGH LEVEL OVERVIEW

This project will be both an iOS application and a pyQT interface allowing users to connect their iOS devices to compatible computers of their choice to allow them to emulate different gaming controllers to play their games with. Throughout this project, we will be using a client-server model which will allow users of the application to locally host a server on their chosen computer/device allowing multiple iOS users to connect seamlessly by accessing their own data as well as shared data by the users, and using TCP protocols to handle requests, it will allow them to emulate their controller's sending inputs and data from client to server or vice versa as needed for functionality and updating the application and interfaces on both sides of the application's connection.



THE CLIENT

- The client will be built into the iOS application allowing them to interface the server.
- The client will be sending data to the server via TCP protocols, which allows them to update and interact with the server as well as their choice in game.
- The client will receive data from the server in order to update and modify their application based on the saved data that the user has stored within the database at certain events.

THE SERVER

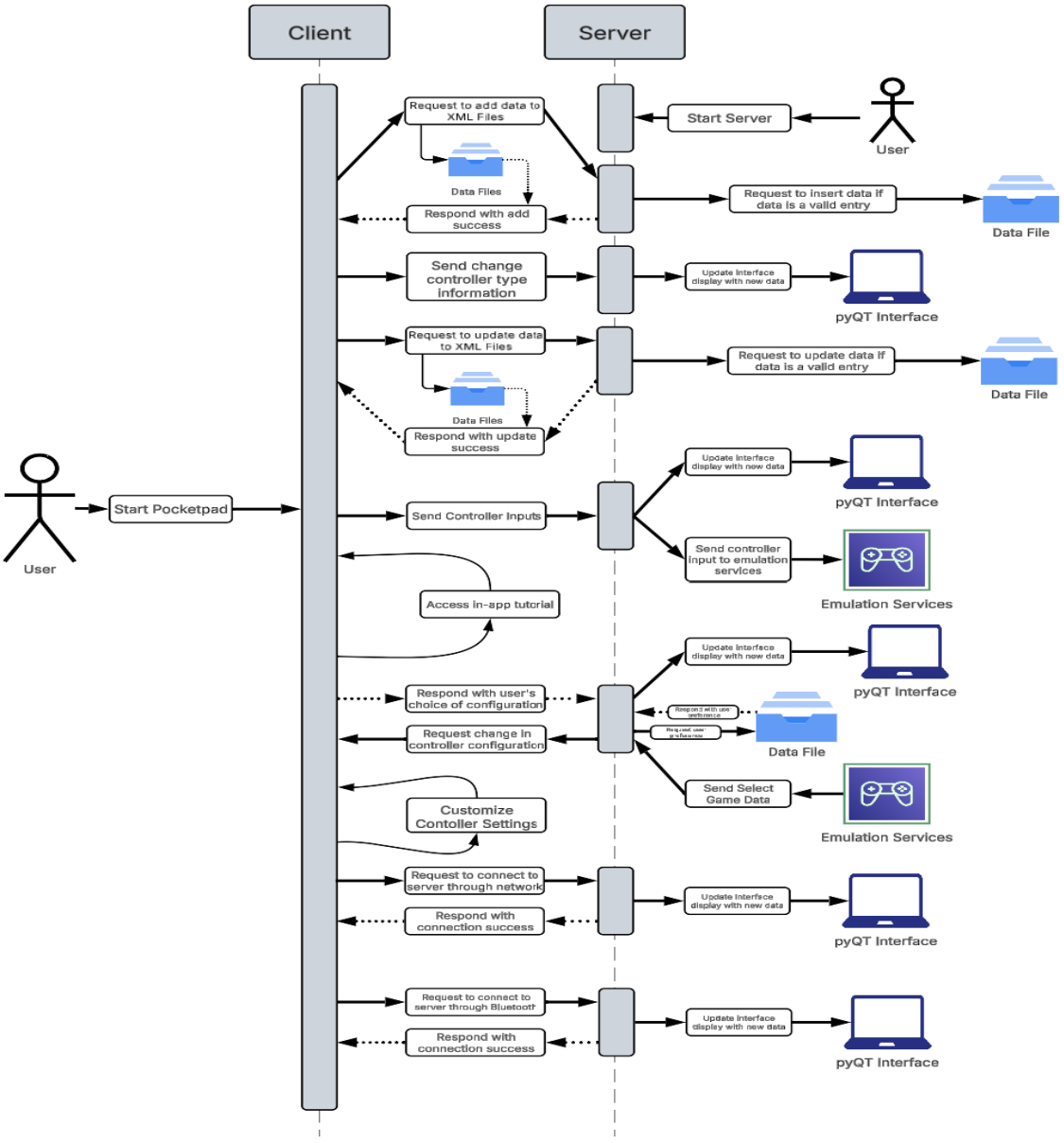
- The server will be built in with a pyQT interface allowing the user to see, test, and interact with data that the client has sent over.
- The server will interact with different gaming and emulation services in order to pass inputs along the pipeline in order to play their intended game.
- The server will use the inputs and data from the client's TCP requests to update both the pyQT display and data files with the nearly acquired data.
- The server will interact and acquire file data in response to events prompted by Pocketpad in order to change or update visualizations both client-side and server-side.

THIRD PARTY EMULATION SERVICES

- The third-party emulator, such as Dolphin, will not be our software, but Pocketpad will work with them.
- The server will parse the TCP controller inputs from the client and pass them to the third-party emulator in a format that it recognizes
- The server will be able to query information about the game from the emulation service for display and/or for functional use, such as controller recommendations based on the emulated game chosen.
- This can be used to implement features like haptics in the controller and responsive lighting with controller UI.

SEQUENCE OF EVENTS: OVERVIEW

The sequence diagram below demonstrates the general functionality of Pocketpad through its client, server, and outside interactions such as that with files and emulations services like Dolphin. To use the application, a user must access both the client-side and server-side of our application. The client-side will be the iOS application for Pocketpad, and through the client-side, users will be able to access all the functionality that we have built in for controller emulation. The server-side allows users to connect their iOS devices to the computer or device hosting the server through the iOS application. It will be the point of connection between the emulated controller and the games that they will be playing through different emulation services such as Dolphin. Once the user is connected to the server, TCP requests and responses will be sent between client and server in order to accomplish the different tasks and functionalities that we are implementing within Pocketpad. These requests will be responsible for transmitting data such as inputs or changes within either the client-side or server-side of our application allowing for full use of the different features we have chosen to implement in our application for the best emulation experience possible. Please look at the sequence of events below for more information about the general overview of our application:



DESIGN ISSUES

Functional Issues:

DO USERS NEED A LOGIN IN ORDER TO ACCESS THE APPLICATION?

OPTIONS:

Option 1: Create a username and password unique to the application

Option 2: Create a login through third party sources such as google or Facebook

Option 3: Allow the user to use the application without login features

DECISION: Option 3

PocketPad chose to go through option 3 to allow users to use the application and its features without the need for certain login features. The only user data or data that will be stored through the use of PocketPad is through the utilization of preexisting iOS functionality to remember application data and application state for the application, and the locally hosted server will only use and interact with user data that the user chooses to share with it or that is necessary for application functionality. As such, nothing that the confidential or compromising for the user will be stored through the use of the application, and alongside the fact that this application is only intended to run locally, this helped us to decide that allowing users to interact with the application without login was the best way to take the project.

HOW CAN WE HELP USERS BETTER USE AND UNDERSTAND THE IOS APPLICATION?

OPTIONS:

Option 1: Provide a ReadMe within our GitHub the functionality and how to use the application

Option 2: Provide a section within the iOS application for FAQ and in-depth application help

Option 3: Provide an interactive tutorial that the user can access from the iOS application

DECISION: All options

At PocketPad, we do not think that one option would suffice for our users, so we chose not to pick at the expense of our users. When it comes to learning, people can find different things helpful whether it is reading, interacting, or simply doing, everyone has their preferred style of learning, and through our application, we aim to provide our users with several formats of learning. Each of the three different options will provide the users with a different format for learning about our application, so as such, we are going to implement all three options within our application. Through this choice we hope to give our users the best interaction with our application that we can by making it as user friendly as we are able through this.

SHOULD THE SERVER'S USER INTERFACE BE CUSTOMIZABLE?

OPTIONS:

Option 1: Have the server remain as is without except for certain events designated by the application

Option 2: Allow the user to customize the coloring of the application regarding different display features

Option 3: Allow the user to toggle on and off certain visualizations that are being displayed

DECISION: Options 2 and 3

At Pocketpad, we seek to provide the best user experience possible when using our application, and we know that a huge part of the user experience is simply how the application looks. The functionality of the application is very important, but it could be the best application in the world and still no one would want to or be happy using it unless it looked good. As such, we wanted to allow our users to customize our server-side interface, and we wanted to do it beyond just simply changing the background color, which is why we are choosing to implement both color customization and visibility of certain features. We know that sometimes depending on the day or mood, people like to be able to customize how their devices look with different device backgrounds or color schemes, so we decided that it would be best to afford users that same ability within our application. Users will be able to customize the coloring of different layout features as well as choose what features will be displayed on the application at a given time. We hope this choice helps us to fulfill our goal making it as user-friendly as possible.

NON-FUNCTIONAL ISSUES:

WHAT PROTOCOL SHOULD WE USE FOR THE NETWORK BACKEND OF POCKETPAD?

OPTIONS:

Option 1: HTTPS

Option 2: TCP

Option 3: UDP

Option 4: WebSockets

DECISION: Option 2

We decided to go with TCP protocols for network connections when writing the backend of PocketPad. As avid game enjoyers ourselves, we, as developers, know that connection speed from input to action is very important when playing games, so we wanted to reflect that when writing the backend for this project. Given the nature of our project and data transmission, a locally hosted server and no real or compromising user data being stored, we decided to prioritize connection speed, and since we won't be transmitting large amounts of data or data that will need very secure connection, TCP seems like the given choice for its fast connection speed, ease of integration with SwiftUI and Python, and ability to integrate with TLS for a more secure connection if/when we deem it needed for PocketPad. Although UDP is a much faster solution, it is also riskier as data transactions aren't always completed. With this in mind, we have decided to go with TCP protocols for our backend in order to ensure things run as efficiently as possible for our users.

HOW SHOULD WE HOST OUR APPLICATION?

OPTIONS:

Option 1: Commercialize the application through subscription to a website and server run by us

Option 2: Create a system that will be the server and database run by us allowing users to use our application

Option 3: Allow users to use and locally host the application on their devices through a GitHub Repository

DECISION: Option 3

We at PocketPad decided that it would be best to allow our users to be able to access our project through a GitHub repository where users can access and then locally host PocketPad on their devices. As developers, we wanted to make a product that was free, easily, and readily accessible for anyone who wanted to use it, especially when gaming can be so expensive these days. Through PocketPad, we want to improve upon existing emulation technology but also make it more readily accessible to those who want it. Given that, it only makes sense for us to use a repository to allow free and ready access to our application, allowing us to fulfill that goal of ours.

WHAT LANGUAGE SHOULD WE IMPLEMENT OUR SERVER-SIDE WITH?

OPTIONS:

Option 1: Python

Option 2: C++

Option 3: JavaScript

DECISION: Option 1

We at PocketPad decided that it would be best to implement our server-side with Python. On our development team, some of our members do not have experience in other languages that we do in Python, which would not do our project justice if we chose to implement it in any other language. A lot of the actual code that our development team will be implementing is something that none of us have worked with before, so doing it in a language that we are not familiar with would only be further detrimental to what we strive to do. As developers, we are striving to make the best possible product that we can for our users, so we are choosing to do it in Python. It will allow us to better construct the application in the first place since we will be more familiar with the language, but it will also allow us to accomplish more with the application and its features. Providing more and better features and implementation for our users is something we strive to do at PocketPad, which is the reason we are choosing to do the server-side in Python.

WHAT FILE TYPE SHOULD WE USE TO STORE CONTROLLER CONFIGURATION?

OPTIONS:

Option 1: JSON

Option 2: SQL

Option 3: XML

DECISION: Option 3

Due to the scope of data storage that we would be using with Pocketpad, we decided that it would be best to use XML file storage for controller configurations. Using this type of system will allow for storage on the user's chosen iOS device without needing to be actively seen or maintained by our users. It will allow them to store their custom configurations and allow those configurations to be easily accessible by the program as needed in the application's functionality. Using XML also affords us the luxury of not needing the size and overhead of other systems like a SQL database which would be unnecessary due to the intended size of data storage and the frequency at which data will be updated. This combined with the fact that XML is easily integrated with Apple's development environment made it the obvious choice for our iOS app client-side.

HOW SHOULD WE STORE USER SETTINGS CLIENT-SIDE?

OPTIONS:

Option1: XML

Option2: UserDefaults

DECISION: Option 2

For Pocketpad's client-side user settings, we had two predominant options for storage: XML files or UserDefaults. Both options allow for the storage of different files such as controller layouts, but storing using UserDefaults offers a much better method for file storage of the two options. Our client-side will be our iOS application built in Swift which has a lot of built in support of storage using UserDefaults making it very easy to implement and interface between data storage and our application. If we were to use a XML file storage system, it would require us to develop multiple processes that would have been automated with the use of a UserDefaults system. Given that we want to do the most for our users with Pocketpad, we decided that it would be best to go with UserDefaults storage since it would provide both an efficient data storage method and afford us more time to implement other features within our application.

HOW SHOULD WE STORE USER SETTINGS SERVER-SIDE?

OPTIONS:

Option1: XML

Option2: SQL Database

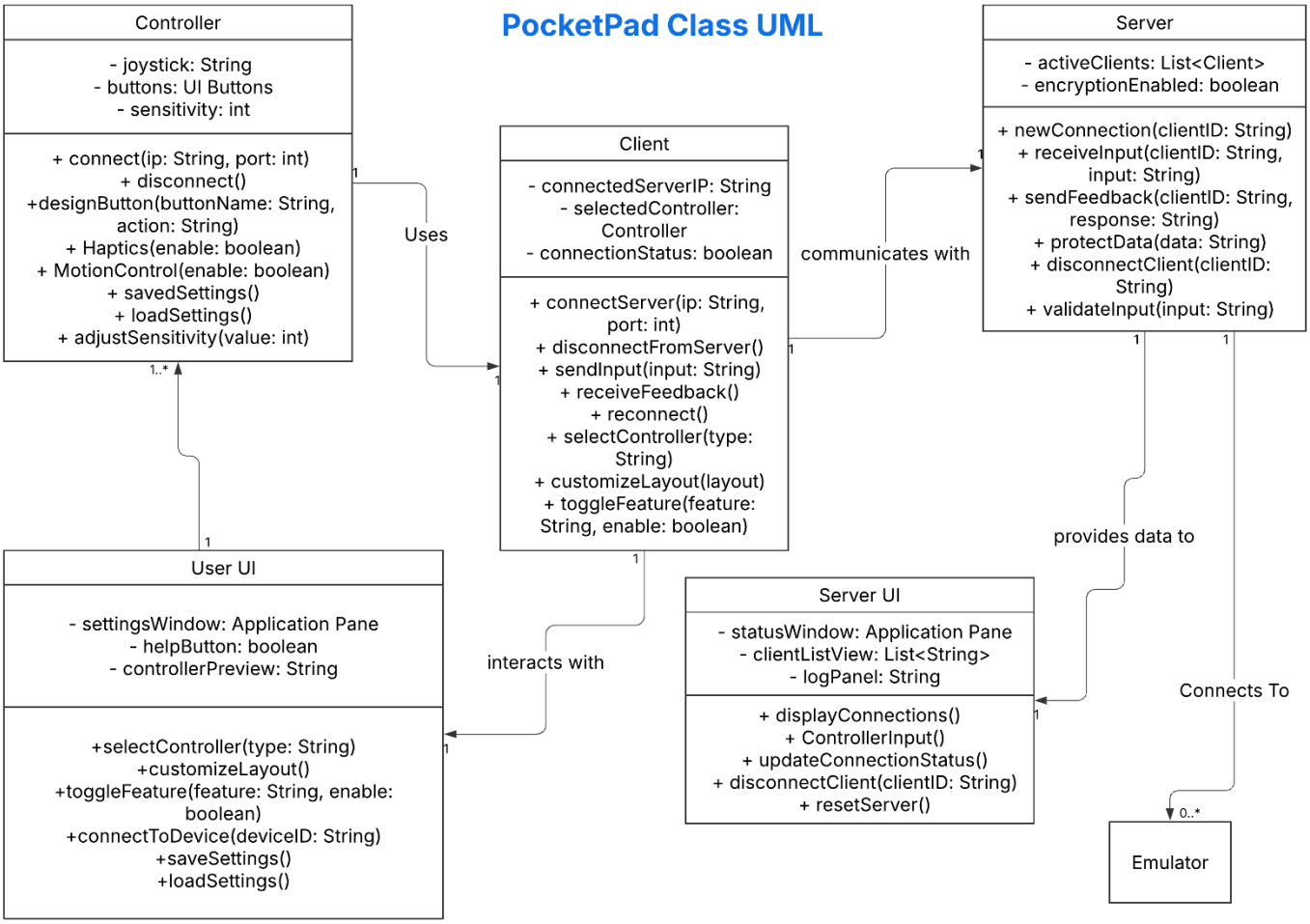
Option3: PySide6 Qsettings

DECISION: Option 3

Given the type of data being stored within Pocketpad, we decided that it would be best if our server-side settings are stored using PySide6 Qsettings. As a part of our project, we are already intending to implement some PySide6 code with our server interface being built using the python's QT framework, so that combined with the fact that will allow users to store persistent settings cross-platform in a very efficient way, especially given the little to no effort that will be required by the user to use and maintain saved/preferred user settings server side of our application. Implementing our server-side data storage with PySide6's Qsettings also affords us the benefit of not requiring extra dependencies to function, which benefits both on our machines when developing and user machines. Since users will be locally hosting the server on their own devices, this choice will simplify not only the storage of user settings but will also require less work of the user to run the application since it will already be added. This integration of PySide6 Qsettings with our already existing user settings made it the clear choice for this.

DESIGN DETAILS

Class Design:



Class Details:

CONTROLLERS CLASS:

- Represents the different types of controllers for games that are being emulated.
- Each controller object is associated with a single user session
- Emulates a real-life controller and its parts:
  - left joystick/right joystick
  - D-Pad/face buttons (A, B, X, Y/1, 2, 3, 4 or equivalent)
  - bumpers/triggers/shoulders/rest buttons
  - start/select/menu buttons
  - motion control (if applicable)
  - haptics (if applicable)
- Supports the typical mapping of Xbox, PlayStation, GameCube, Nintendo Switch
- Option to create and save a custom mapping exists
- Buttons can be changed as users wish.
- Includes personal settings for sensitivity, dead zone, vibration, and motion control
- Connects to central server via Bluetooth or over the network

FUNCTIONS:

- connect(ip: String, port: int):
  - Makes connection between the controller and the server.
- disconnect():
  - Disconnects the controller from the server.
- designButton(buttonName: String, action: String):
  - Gives a function to a specific button.
- Haptics(enable: boolean):
  - Turns on or Off vibration.
- MotionControl(enable: boolean):
  - Turns on or Off motion input.
- savedSettings():
  - Stores the current settings.
- loadSettings():
  - Loads previously saved settings.
- adjustSensitivity(value: int):
  - Modifies the sensitivity.

SERVER CLASS:

- Manages all client-app communication with the game/emulator.
- Acts as a translator for mobile requests to game actions.
- Enables simultaneous multi-client, multi-controller connections.
- Receives information from the clients and gives it to the game to which it's connected.
- information back to the clients regarding connection and whether it registers their input.
- Handles input lag for desynchronization and preferred gameplay options.
- Implements encryption protocols for secure data transmission.
- Enables TCP/IP or Bluetooth connection.
- Supplies API features to control other game emulators (i.e. Dolphin) via remote.

FUNCTIONS:

- newConnection(clientID: String):
  - Manages a new client connection.
- receiveInput(clientID: String, input: String):
  - Manages input data from the client.
- sendFeedback(clientID: String, response: String):
  - Sends messages to clients about status of inputs.
- protectData(data: String):
  - Secures data.
- disconnectClient(clientID: String):
  - Ends a client's connection.
- validateInput(input: String):
  - Checks that the given input is in the correct form before given it to the game.

CLIENT CLASS:

- The device the user interacts with for their gameplay and control.
- The sub-menu for choosing, customizing, and experimenting with virtual controllers
- The function for transmitting controller commands in real-time to the server.
- The function for receiving input sent back from the Server.
- Shows available devices for connection.
- The function for button changing buttons, sensitivity, and haptic feedback.
- Supports multiple connection types (Wi-Fi and Bluetooth).
- Saves and loads user preferences for controller layouts.
- Manages reconnections if the connection is interrupted.

FUNCTIONS:

- connectServer(ip: String, port: int):
  - Connects with the game server.
- disconnectFromServer():
  - Disconnects from the server.
- sendInput(input: String):
  - Sends input to the server.
- receiveFeedback():
  - Receives confirmation from the server of successful connection.
- reconnect():
  - Attempts to reconnect if connection is lost.
- selectController(type: String):
  - Lets user to choose a controller layout.
- customizeLayout(layout:):
  - Enables moving/changing the buttons and features.
- toggleFeature(feature: String, enable: boolean):
  - Turns on or off specific controller feature.

SERVER_UI CLASS:

- Ability to check server status through an interface.
- Displays a list of connected clients and their controllers.
- Shows real-time input data.
- Allows to manage connections.
- Provides logs of data transmission for debugging purposes.
- Shows network strength for each connection.
- Shows an option to reset the server.

FUNCTIONS:

- displayConnections():
  - Shows all active users.
- ControllerInput():
  - Shows input data from connected controllers.
- updateConnectionStatus():
  - Updates the list of active devices.
- disconnectClient(clientID: String):
  - Disconnects client from the server.
- resetServer():
  - Resets all connections and restarts the server.

CLIENT_UI CLASS:

- The main interface of PocketPad.
- Shows a selection of controller presets.
- Lets users create and save custom controller layouts.
- Gives feedback for button presses and controller inputs.
- Shows a list of devices available for connection.
- Shows configuration settings for haptic feedback, motion control, and button sensitivity among other functionalities.
- Includes a help section for troubleshooting issues.
- Allows users to save and load configurations.

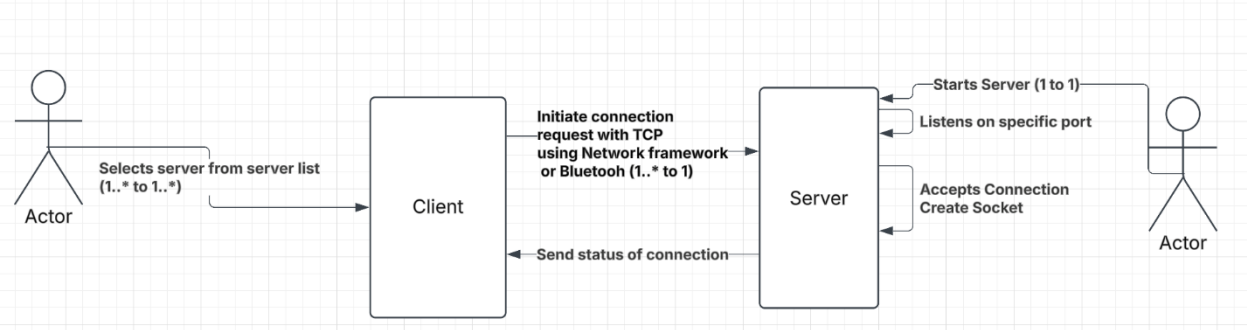
FUNCTIONS:

- selectController(type: String):
  - Allows users to choose an controller layout.
- customizeLayout():
  - Enables users to change buttons and adjust inputs.
- toggleFeature(feature: String, enable: boolean):
  - Turns on or Off a specific controller features.
- connectToDevice(deviceID: String):
  - Connects with a device.
- saveSettings():
  - Saves controller settings.
- loadSettings():
  - Loads a previously saved settings.

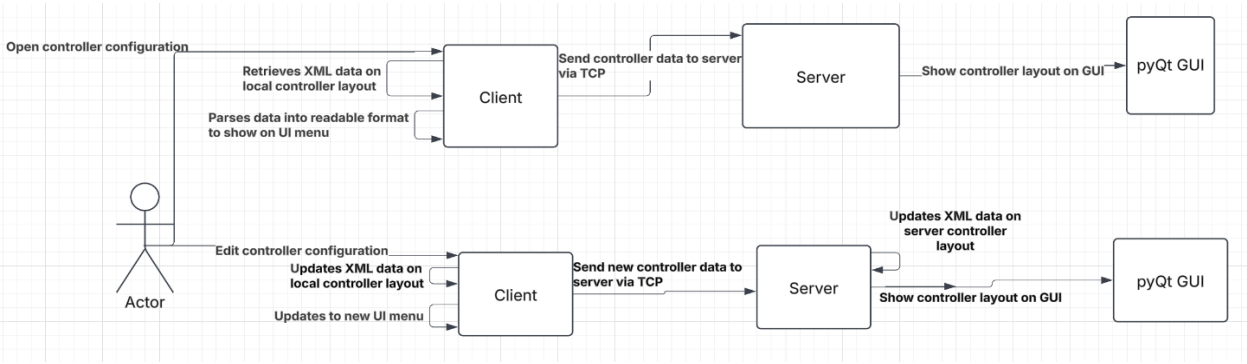
SEQUENCE DIAGRAMS

System Activities:

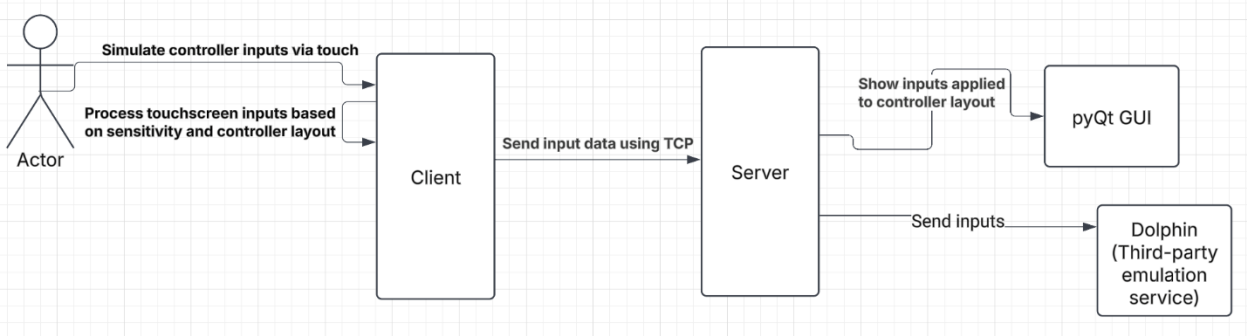
NETWORK CONNECTION:



CONTROLLER CONFIGURATION:



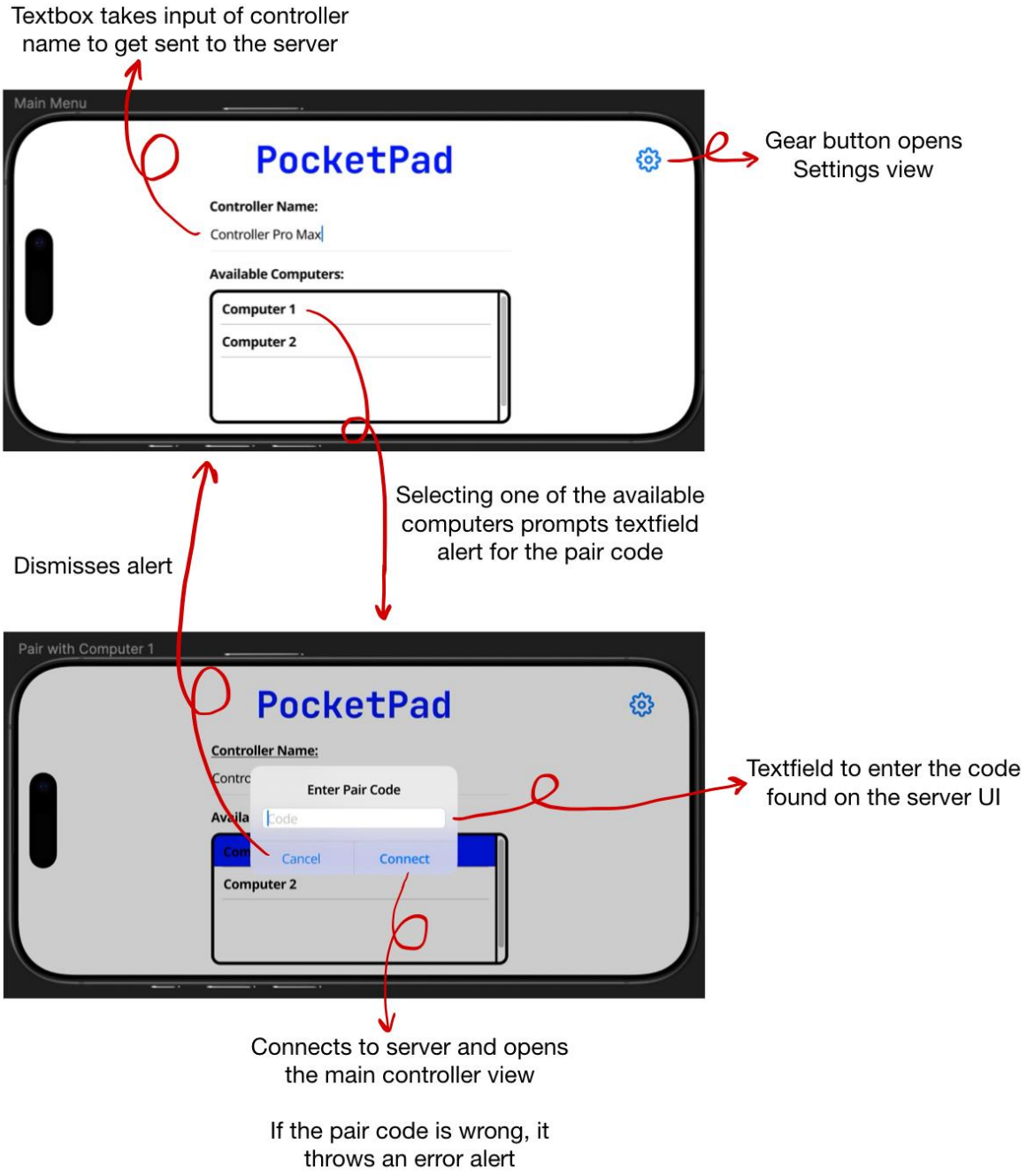
INPUT SENDING:



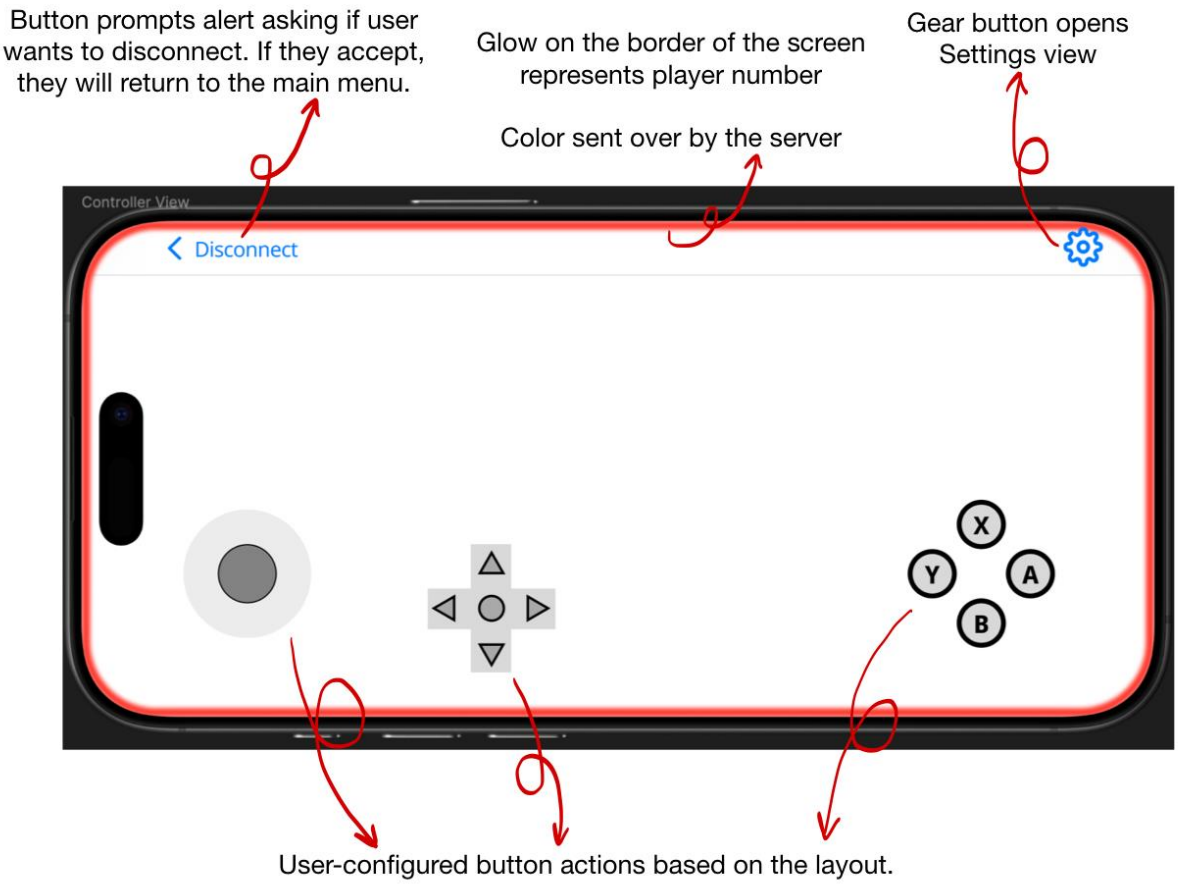
UI MOCKUPS

iOS (Client) UI:

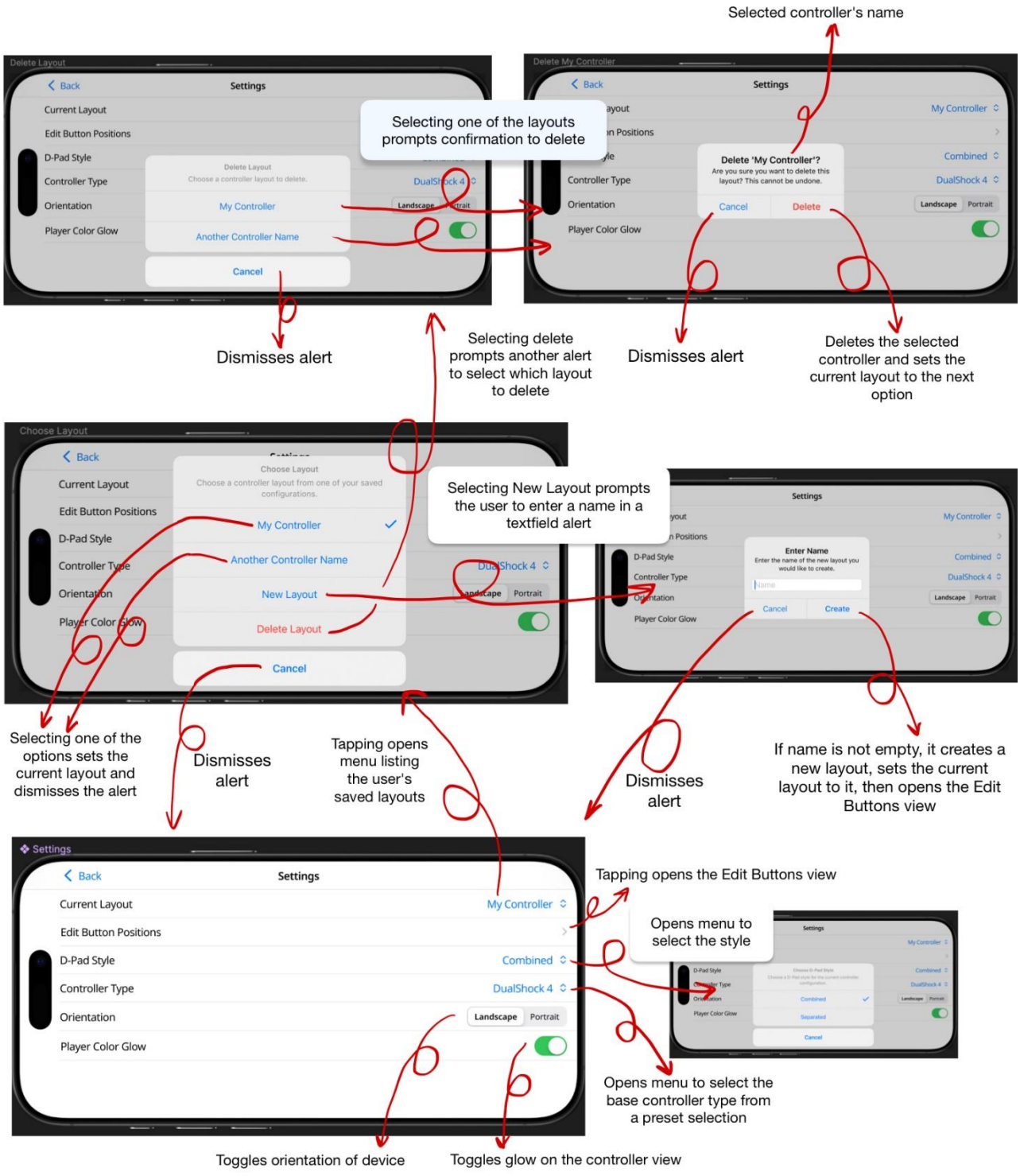
MAIN MENU VIEW:



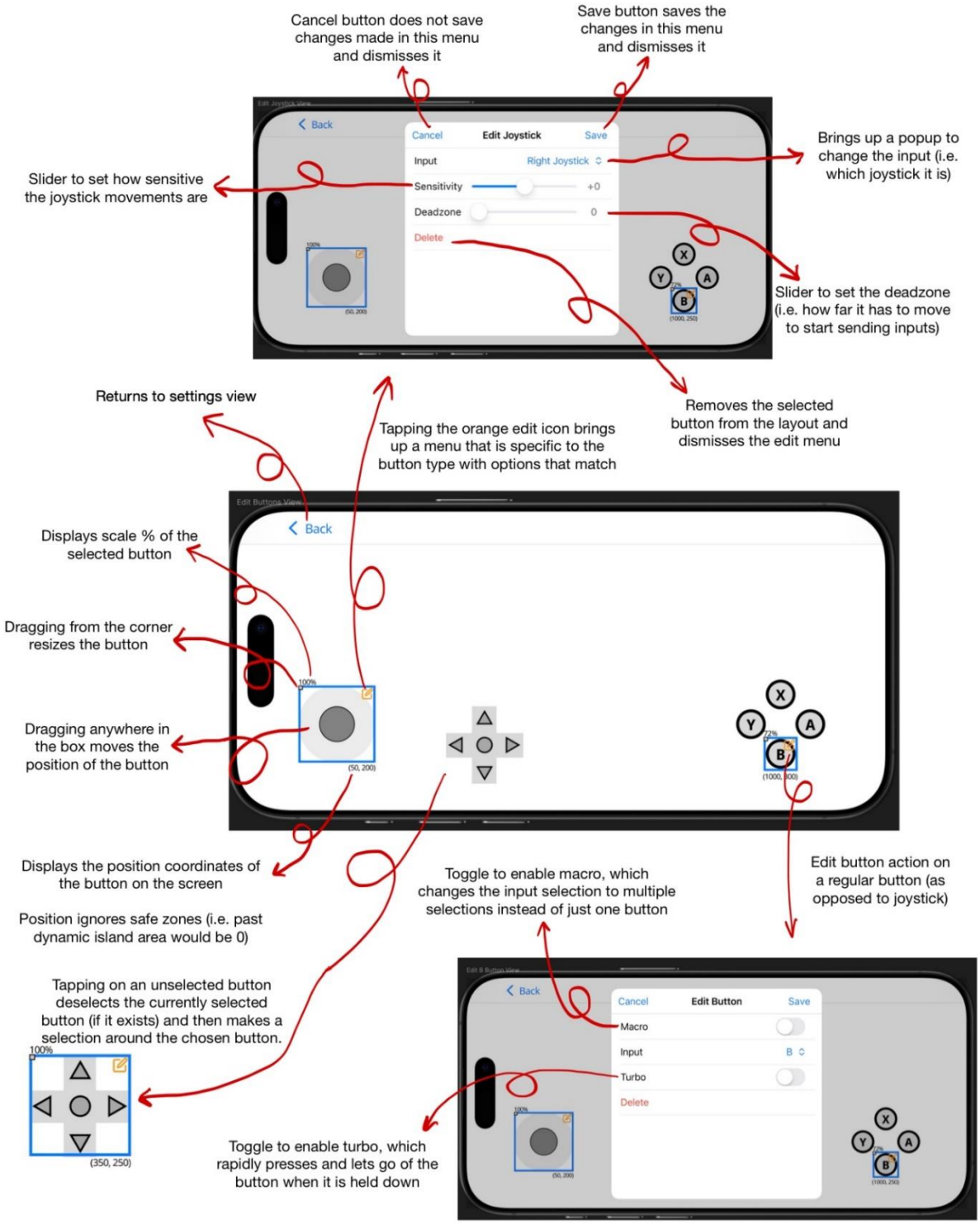
CONTROLLER VIEW:



SETTINGS VIEW:



CONTROLLER LAYOUT EDITOR VIEW:



Python (Server) UI: