**UWROV GUI Documentation**

**Contents:**

* Preface | 1
* Architecture | 1
  + Module Structure | 1-2
  + Main Module | 2
  + Input Module |
    - Controller Module |
  + Display Module |
  + Network Module |

**Preface:**

Our goal for this new iteration of the graphic user interface is to have a reusable, updatable, manageable user interface that can be used for future years. This will ultimately imitate a company setting where new hires will usually “update” the current software rather than remaking new versions of the software every time. This will require persistent documentation and a well-trained team. I hope that everyone will work together to provide a productive learning environment that lasts many years into future. This approach to projects should help develop skills to work with people of many different backgrounds of programming and skill level.

To make sure that we have a manageable and flexible User interface, we will adapt a specific project architecture that will make sure that our project remains organized and manageable.

Our design philosophy will focus on the idea of “modular programming”. This style of programming focuses on separating functionality of a program to individual, independent “modules” that can be modified separately, and switched out without affecting the rest of the program as a whole. When a program is small, it is not very difficult to tell everyone to learn how the program functions as a whole. However, the program develops and gets to a point where it becomes unreasonable to tell everyone working on it to go and learn how the whole program functions. By adapting a modular structure, programmers will only need to focus on each individual “module” that requires only the knowledge of how the module interacts with the “core” and how the module acts. A good modular architecture will lessen the workload of managing clunky and poorly structured programs.

A crude analogy of how a well-modulated program will function is the human body. If an organ in a human body loses function or malfunctions, the only work required to fix it is to fix or replace the organ that is having problems. It is unnecessary to touch the other organs that are working fine. All the organs have their own inner workings and have their individual functions. It would be very difficult to remedy an organ if it had more than one core function. For example, let’s say that an organ had the functionality of a heart and the stomach but the stomach functionality somehow malfunctions. It is cumbersome to try to fix the stomach without messing up the heart functionality of the organ. In conclusion, you would need to work around the heart functionality and hopefully not mess up that functionality to fix the stomach functionality. All in all, it is very efficient to manage separated modules rather than have one that has interconnection throughout all the functions.

Each module should follow the overall guide to ensure that each component does not have too many dependencies on other components.

**Architecture:**

The GUI will be separated into three main modules: main, input, display. There will also be functions that help with networking that will be used in each module that could be considered the networking module. Each module will contain submodules that will be designed as a smaller independent function of each main module.

**Module Structure:**

* **Constants/Variables:**
  + While constants are read only values, make sure to label each variable that is needed for each module component. Keep in mind of variables that interact with other modules and minimize the variables that are shared. Ideally the only variables that interact with multiple modules are ones that are part of the main module.
* **Main Functions**:
  + Similar to variables, minimize interaction with other modules to a few functions and try to fit the whole functionality of the module in functions that can be stand alone. These functions should be the core to each module.
* **Helper Functions:**
  + Helper functions are merely functions to help keep code by providing utility to modules. They can either be module specific or can apply to the entire project.
* **Design goals:**
  + Each module, even the submodules, should be able to be isolated and debugged. This means that if you copy and pasted a submodule onto a new file, it should be able to run in a compatible environment. This is possible if there are almost no dependencies to the module. The only exception would be the core module that interacts with all of the other modules to bring everything together.

Example Documentation:

Intro:

* (description of the module)

Constants:

* Constant name: (Description of each constant)

Fields:

* Field name: (Description of each field)

Functions:

* Function name: (description of functionionality)
  + Paramaters (description of the params)
  + Return values (description of the representation)
  + Exceptions thrown (description of why it was thrown)

**Main Module:**

* Intro: The main module is unlike the other modules and is the core module that accesses the other modules to connect them to each other. It sets up each module so that they function properly and should be the only module that has many dependencies.

**Input Module:**

* Intro: The input module handles all of the functions that will connect input to the GUI. This includes keyboard, buttons, controllers, etc.
* **Controller Module:**

**Display Module:**

* Intro: The display module handles all of the functions that will display information onto the GUI. This means that all visual modules would be considered to be part of the display module.

**Network Module:**

* Intro: The network module infers to all of the functions that handle network requests and organizes all of the information required to handle transfer of information to the server. This also includes the server test environment that should imitate the actual server locally.
* Constants:
  + API\_URL: the URL that all http requests are sent to. This may be the URL of the actual server or a URL to the local test server.