Pacemaker Design

SFWRENG 3K04

Ventricle Capitalists (Group 7)

Aurora Byrdon (400065667), Arthur Faron (400075996), Yansong (Kevin) Hu (400079343), David Lui (400069997), Michelle Monte (001132205), Erin Puersten (400057681), Daniel Su (400068075)

Fall 2018

Alan Wassyng

**Table of Contents**

1.0 Modules

1.1 Class Login Window

1.2 Class Home Window

2.0 Future Requirements

3.0 Future Design Decisions

1. References

**1.0 Module Documentation**

**1.1 Module Documentation: Class Login Window**

**1.1.1 Purpose**

The purpose of the Login Window Class is to provide a basic GUI where the user may login and register. Simple file I/O allows the user to register in one session, close the terminal, and log back in to another session. A successful login will transfer the user to the home window explained below. The login window also contains the following features: ABOUT (displays information about the software), USERS (lists the currently registered users), and QUIT (terminates the login window).

**1.1.2 Information Hiding (Secret)**

There is no information hiding in this module.

**1.1.3 Public Functions and Parameters**

The Popup() function is public and can be used in all classes – it is defined outside of all classes. This function opens a popup window with the specified title and text. The user is provided with an ‘OKAY’ button that they can click to accept the message and dismiss the window.

**1.1.4 Black-Box Behaviour**

The black-box behaviour of each function/method found inside or outside any of the classes is outlined in the table below:

|  |  |  |
| --- | --- | --- |
| Function | Input | Output |
| \_\_add\_uname\_pword | self.uname.get()  self.pword.get()  self.pcheck.get()  self.unames | \*summarized in additional table below |
| \_\_check\_uname\_pword | self.un.get()  self.pw.get()  self.unames | \*summarized in additional table below |
| \_\_users |  |  |
| \_\_about |  |  |
| \_\_no\_user |  |  |
| \_\_wrong\_password |  |  |
| \_\_successful\_registration |  |  |
| \_\_too\_many\_users |  |  |
| \_\_pass\_no\_match |  |  |
| \_\_pass\_no\_match |  |  |
| \_\_pass\_too\_short |  |  |
| \_\_user\_exists |  |  |
| \_\_successful\_login |  |  |

\_\_add\_uname\_pword and \_\_check\_uname\_pword follow the logic laid out below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| \_\_add\_uname\_pword  (Register) | >= 10 users registered | | | Error |
| < 10 users registered | Username already exists | | Error |
| Username does not already exist | Password and Password check do not match | Error |
| Password is < 6 characters | Error |
| Password and Password check match and Password >= 6 characters | Successful Registration |
| \_\_check\_uname\_pword  (Login) | Username does not already exist | | | Error |
| Username already exists | Password does not match | | Error |
| Password matches | | Successful Login |

**1.1.5 Global Variables**

This class has global variables (attributes) as outlined below:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| self.unames | List | A list which contains the registered usernames. When the program is started, this list is filled with the usernames in the csv file and anytime a new user is registered their name is added to the csv file. |
| self.pwords | List | A list which contains the registered passwords. The index of the password will match the index of the corresponding username. Passwords are stored in the csv file exactly as usernames are. |
| self.uname | Entry | An entry box (using tkinter) which can return the username for registration |
| self.pword | Entry | An entry box (using tkinter) which can return the password for registration |
| self.pcheck | Entry | An entry box (using tkinter) which can return the password confirmation for registration |
| self.un | Entry | An entry box (using tkinter) which can return the username for login |
| self.pw | Entry | An entry box (using tkinter) which can return the password for login |

**1.1.6 Data Structure**

[Unsure what to put here]

**1.1.7 Private Functions**

Within the Login\_Window class there are 13 private functions (methods). These methods can only be used within the class Login\_Window as they are only needed there.

|  |  |
| --- | --- |
| Method | Description |
| \_\_init\_\_ | Creates the entry fields, text, and buttons for the login screen |
| \_\_add\_uname\_pword | Verifies the submitted username, password, and password confirm and adds them to the lists (self.unames and self.pwords) if they are valid |
| \_\_check\_uname\_pword | Verifies the submitted username and password and opens the Login\_Window if they are valid |
| \_\_users | Displays a popup window with the registered list of usernames |
| \_\_about | Displays a popup window with information about the software |
| \_\_no\_user | Displays a popup window with an error |
| \_\_wrong\_password | Displays a popup window with an error |
| \_\_successful\_registration | Displays a popup window with an error |
| \_\_too\_many\_users | Displays a popup window with an error |
| \_\_pass\_no\_match | Displays a popup window with an error |
| \_\_pass\_no\_match |  |
| \_\_pass\_too\_short | Displays a popup window with an error |
| \_\_user\_exists | Displays a popup window with an error |
| \_\_successful\_login | Opens the Home\_Window |

**1.2 Module Documentation: Class Home Window**

**1.2.1 Purpose**

The purpose of the Home Window Class is provide a GUI where the logged in used may program modes, states, and parameters to send to a Pacemaker. The user may choose to open up egram data (currently blank), view more information on modes and states, change patients (returns the user to the login screen) , or quit the session.

The parameters were made with respect to table [table number] and table [table number] from the [pacemaker documentation].

**1.2.2 Information Hiding**

Information hiding is used to keep the telemetry and serial communication separate from the rest of the code wherever possible to reduce the change that will need to be implemented later. In functions below, everything that needs to change to allow for serial communications can be done without touching the codes outside.

|  |  |  |
| --- | --- | --- |
| Class | Method | Description |
| Home\_Window | \_\_send\_param | Now: method prints the current parameters and mode to the terminal  Future: method transmits the parameters and mode to the Pacemaker then gives confirmation that the information was correctly stored |
| Home\_Window | \_\_start\_egram | Now: method opens a new empty window  Future: method opens a new window and receives egram data from the Pacemaker which is presented graphically |

**1.2.3 Public Functions and Parameters**

The Popup() function is public and can be used in all classes – it is defined outside of all classes. This function opens a popup window with the specified title and text. The user is provided with an ‘OKAY’ button that they can click to accept the message and dismiss the window.

**1.2.4 Black-Box Behaviour**

|  |  |  |
| --- | --- | --- |
| \_\_change\_\_mode |  |  |
| \_\_change\_\_state |  |  |
| \_\_send\_\_param |  |  |
| \_\_more\_\_info |  |  |
| \_\_start\_\_egram |  |  |

**1.2.5 Global Variables**

This class has global variables (attributes) as outlined below:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**1.2.6 Data Structure**

[not sure what to put here]

**1.2.7 Private Functions**

Within the Home\_Window class there are 6 private functions.

|  |  |
| --- | --- |
| Method | Description |
| \_\_init\_\_ | Creates the dropdown menus, check boxes, buttons, and text for the home screen |
| \_\_change\_\_mode | Depending on the selected mode, different parameters are disabled or enabled for user manipulation |
| \_\_change\_\_state | Depending on the selected state, different modes are disabled or enabled |
| \_\_send\_\_param | Prints the current parameters by state to the terminal (subject to change) |
| \_\_more\_\_info | Creates a popup window with information on the states and modes |
| \_\_start\_\_egram | Opens the Egram\_Window |

**1.2 Module Documentation: Class Egram Window**

**1.3.1 Purpose**

**1.3.2 Information Hiding**

**1.3.3 Public Functions and Parameters**

The Popup() function is public and can be used in all classes – it is defined outside of all classes. This function opens a popup window with the specified title and text. The user is provided with an ‘OKAY’ button that they can click to accept the message and dismiss the window.

**1.3.4 Black-Box Behaviour**

**1.3.5 Global Variables**

**1.3.6 Data Structure**

**1.3.7 Private Functions**

**2.0 Future Requirements**

2.1 Serial Communications

The main future requirements to account for involve serial communications with the Pacemaker. The program will need to send parameter data to the Pacemaker (and verify that it was sent correctly) and receive information regarding telemetry (new devices must be recognized, as must a change of devices) and electrograms. These electrograms will need to be displayed to the user in a GUI window. See a short summary of the serial communications requirements for future assignments below:

Serial Communications Requirements:

* Indicate when a new device is approaching
* Send parameter data to the Pacemaker include mode, state, and relevant parameters for the chosen mode and state
* Verify that parameter data was correctly recorded on the Pacemaker
* Receive electrogram data from the Pacemaker (1) for a single sensor and (2) for both atrial and ventricular sensors
* Stop receiving electrogram data from the Pacemaker
* Indicate when telemetry is lost due to noise
* Indicate when telemetry is lost because device is out of range

2.2 Modes

In the future, we will also need to consider new modes (Assignment 2: VOO, AOO, VVI, AAI; Assignment 3: DDD, DDDR) and ensure that they are correctly programmed. It must also be possible to dynamically change between any two modes without restarting the device. See a short summary of the mode requirements for future assignments below:

Mode Requirements:

* VOO, AOO, VVI, AAI
* DDD, DDDR
* Dynamically change between two modes without restarting the device

**3.0 Future Design Decisions**

To meet the future requirements, some changes and additions will have to be mode to the current model.

3.1 Serial Communications

The serial communications requirements can be met by:

Serial Communications Design Decisions:

3.2 Modes

The mode requirements can be met by:

Mode Design Decisions:

**4.0 References**

[pacemaker documentation]