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 | self.unames | Summarized in 1.1.7 |
| \_\_about | None | Summarized in 1.1.7 |
| \_\_no\_user | None | Summarized in 1.1.7 |
| \_\_wrong\_password | None | Summarized in 1.1.7 |
| \_\_successful\_registration | None | Summarized in 1.1.7 |
| \_\_too\_many\_users | None | Summarized in 1.1.7 |
| \_\_pass\_no\_match | None | Summarized in 1.1.7 |
| \_\_pass\_no\_match | None | Summarized in 1.1.7 |
| \_\_pass\_too\_short | None | Summarized in 1.1.7 |
| \_\_user\_exists | None | Summarized in 1.1.7 |
| \_\_successful\_login | self.master  self.homeWindow  self.loginWindow | Summarized in 1.1.7 |

\_\_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 confirming the registration |
| \_\_too\_many\_users | Displays a popup window with an error |
| \_\_pass\_no\_match | Displays a popup window with an error |
| \_\_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 user may set the parameters for the various Pacemaker modes. The user may also choose to view electrocardiogram data (to be implemented), 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**

|  |  |  |
| --- | --- | --- |
| **Function** | **Input** | **Output** |
| \_\_change\_\_mode | self.mode | Enables and disables option menus, emulating the behaviour shown in Table 6 of the Pacemaker reference document. |
| \_\_change\_\_state | self.state |  |
| \_\_send\_\_param | self.mode  self.pulseRate  self.vAmp  self.vPulseWidth | Now: A popup window showing the data to be updated and their current values  Future: The pacemaker with the updated values, and a window confirming a successful upload |
| \_\_more\_\_info | none | An information window describing the different operational modes and states |
| \_\_start\_\_egram | none | Now: A placeholder popup window for the electrocardiogram  Future: A window displaying the electrocardiogram from the |

**1.2.5 Global Variables**

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

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| self.mode | StringVar | Keeps track of selected pacemaker mode. |
| self.modeMenu | OptionMenu | Handle for a dropdown menu containing a selection of all pacemaker modes, as described in Table 6 under the Pacemaker reference document |
| self.state | StringVar | Keeps track of selected connection state |
| self.stateMenu | OptionMenu | Handle for a dropdown menu used to select the connection mode to the pacemaker. |
| self.pulseRate | StringVar | Keeps track of the currently selected pacing rate. |
| self.pr | OptionMenu | Handle for a dropdown menu used to select the pacing rate of the pacemaker, in paces per minute |
| self.dynAVDelay | IntVar | Stores whether the Dynamic AV Delay is enabled or not |
| self.davd | Checkbutton | Checkbox that toggles the Dynamic AV Delay |
| self.fixedAVDelay | StringVar | Keeps track of the selected Fixed AV Delay |
| self.favd | OptionMenu | Handle for a dropdown menu used to select the fixed AV delay on the pacemaker, in milliseconds |
| self.sensedAVDelayOffset | StringVar | Keeps track of the selected Sensed AV Offset |
| self.savdo | OptionMenu | Handle for a dropdown menu used to select the Sensed AV Delay Offset, in milliseconds |
| self.maxSensorRate | StringVar | Keeps track of the current selected Maximum Sensor Rate |
| self.msr | OptionMenu | Handle for a dropdown menu used to select the Maximum Sensor Rate, in ppm (points per minute) |
| self.activityThreshold | StringVar | Keeps track of the currently selected Activity Threshold (Given as a list of string constants) |

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** | **Description** |
| self.at | OptionMenu | Handle for a dropdown menu used to select the Activity Threshold for the pacemaker. |
| self.reactionTime | StringVar | Keeps track of the selected Reaction Time for the pacemaker |
| self.ret | OptionMenu | Handle for a dropdown menu used to select the pacemaker’s Reaction Time |
| self.responseFactor | StringVar | Keeps track of the selected Response Factor for the pacemaker |
| self.rf | OptionMenu | Handle for a dropdown menu used to select the pacemaker Response Factor |
| self.recoveryTime | StringVar | Keeps track of the selected Recovery Time for the pacemaker |
| self.rt | OptionMenu | Handle for a dropdown menu used to select the pacemaker Recovery Time |
| self.aPulseWidth | StringVar | Keeps track of the selected Atrial Pulse Width setting |
| self.apw | OptionMenu | Handle for a dropdown menu used to select the Atrial Pulse Width, given in milliseconds |
| self.aAmp | StringVar | Keeps track of the selected Atrial Amplitude setting |
| self.aa | OptionMenu | Handle for a dropdown menu used to select the pacemaker Atrial Amplitude, in Volts |
| self.aSensitivity | StringVar | Keeps track of the selected Atrial Sensitivity |
| self.ase | OptionMenu | Dropdown menu used to select the pacemaker Atrial Sensitivity, in millivolts |
| self.aPulseWidth | StringVar | Keeps track of the selected Atrial Pulse Width (APW) |
| self.apw | OptionMenu | Handle for a dropdown menu used to select the APW, in ms |
| self.rateSmoothing | StringVar | Keeps track of the current rate smoothing percentage |
| self.rs | OptionMenu | Handle for a dropdown menu used to select the rate smoothing percentage |
| self.hys | StringVar | Keeps track of the current hysteresis setting |
| self.h | OptionMenu | Handle for a dropdown menu used to select the hysteresis, in ppm. |
| self.atrfm | Checkbutton | Handle for a checkbutton that keeps track of whether the Atrial Tachycardia Response (ATR) is enabled. |
| self.atrDuration | StringVar | Keeps track of the currently selected ATR duration – how long the device remains in the ATR state |
| self.atrd | OptionMenu | Handle for a dropdown menu used to select the ATR duration, in cc (cardiac cycles) |
| self.atrFallbackTime | StringVar | Keeps track of the currently selected ATR fallback time |
| self.atrft | OptionMenu | Handle for a dropdown menu used to select the ATR fallback time, in minutes. |
| self.vPulseWidth | StringVar | Keeps track of the selected Ventricular Pulse Width setting |
| self.vpw | OptionMenu | Handle for a dropdown menu used to select the Ventricular Pulse Width, given in milliseconds |
| self.vAmp | StringVar | Keeps track of the selected Ventricular Amplitude setting |
| self.va | OptionMenu | Handle for a dropdown menu used to select the pacemaker Ventricular Amplitude, in Volts |
| **Attribute** | **Type** | **Description** |
| self.vSensitivity | StringVar | Keeps track of the selected Ventricular Sensitivity setting |
| self.vs | OptionMenu | Handle for a dropdown menu used to select the pacemaker Ventricular Sensitivity, in millivolts |
| self.vRefractPeriod | StringVar | Keeps track of the selected Ventricular Refractory Period setting |
| self.vrp | OptionMenu | Handle for a dropdown menu used to select the pacemaker Ventricular Refractory Period, in milliseconds |
| self.aRefractPeriod | StringVar | Keeps track of the selected Atrial Refractory Period (ARP) |
| self.arp | OptionMenu | Handle for a dropdown menu used to select the pacemaker ARP, in ms |
| self.postVARefractPeriod | StringVar | Keeps track of the selected Post-Ventricular Atrial Refractory Period (PVARP) |
| self.pvarp | OptionMenu | Handle for a dropdown menu used to select the pacemaker ARP, in ms |
| self.postVARefractPeriodExt | StringVar | Keeps track of the selected PVARP Extension |
| self.pvarpe | OptionMenu | Handle for a dropdown menu used to select the pacemaker PVARP extension |

The values that each dropdown menu accept are given in table 7 of the PACEMAKER reference document

**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]