**SE 3K04 Assignment 2**

**Part 2: Pacemaker DCM**

**Group 9**

Kevin Cheung

Forrest Herman

Atiqul Islam

Jingming Liu

**Sunday, November 29, 2020**

Table of Contents

[**Introduction**](#_3usk90ddk1o7) **3**

[**Specifications**](#_esky3c71bk32) **3**

[Requirements for Assignment 1:](#_h1rqtnak5a05) 3

[Requirements for Assignment 2:](#_63bhsrob8dpl) 3

[**Modules**](#_koe89ra8size) **4**

[WindowManager](#_ee43liizd5iq) 5

[WelcomeWindow](#_to3ok129q808) 5

[LoginWindow](#_v62xkt1n56rr) 6

[RegisterWindow](#_o84ey0uf1j3) 6

[MainWindow](#_qodukb8zkyw) 7

[modeSelectorPopup](#_py563wzguacq) 9

[errorPopup](#_lxf01wjybgzq) 10

[errorMaxPopup](#_3qrs0rgc1dk8) 11

[paramErrorPopup](#_1p4d3vf8vxvn) 11

[successPopup](#_vfoio3buhjbh) 11

[Database](#_lhsdkicegmis) 11

[PacemakerApp](#_4mblsi9zsgku) 12

[Global Functions](#_y83325ufpyqm) 12

[**Variables**](#_xitlhabf0ke8) **18**

[Global State Variables](#_jutm5t25kfgm) 18

[**Future Changes**](#_mudpihxjxei6) **27**

[Requirements](#_obyy57llhvdy) 27

[Design Decisions](#_g10stdd2cap6) 27

## Introduction

This DCM, built using Python 3 and Kivy, creates an interface for the Pacemaker software design. From part 1, the program features a register and login feature, the ability to set the pacing mode, and view and control the values of all the programmable parameters. In part 2 serial communication for data transfer between the DCM and pacemaker was added. This allowed for control of the pacemaker directly from the DCM, as well as the ability to display an egram of the heartbeat on the user interface. Additional pacing modes and parameters were also added.

## Specifications

### Requirements for Assignment 1:

* Register and login page for 10 local users.
* The user interface shall be capable of utilizing and managing windows for display of text and graphics.
* The user interface shall be capable of processing user positioning and input buttons.
  + Mouse and keyboard input
* The user interface shall be capable of displaying all programmable parameters for review and modification.
  + Lower Rate Limit, Upper Rate Limit, Atrial Amplitude, Atrial Pulse Width, Ventricular Amplitude, Ventricular Pulse Width, VRP, and ARP.
* The user interface shall be capable of visually indicating when the DCM and the device are communicating.

### Requirements for Assignment 2:

* Expand the DCM to include all required modes and parameters
* Implement serial communication to transmit and receive information between the DCM and the Pacemaker
* The system will be able to set, store, transmit programmable parameter data, and verify it is stored correctly on the Pacemaker device
  + A or V Pulse Amplitude Regulated
  + A or V Pulse Width
  + A or V Sensitivity
* The system will be able to display egram data when the user chooses to do so (for either ventricle, atrium, or both).
  + The DCM must receive the egram data from the Pacemaker over the serial communication link in order to display it
* Document how programmable parameters originate at the DCM and are implemented in the device. Show how you can ensure the parameters stored in the Pacemaker are what the doctor input on the DCM. Also justify your choice of the data types used to represent parameters data
* User Guide

When first launching the program you will be greeted with a login screen. To create a new account, click the “Register” button to get started. From there you will be asked to enter your full name, as well as a username and password. The username and password will be used to access the main program. Once an account has been created, click the “Login” button on the main screen and enter your account details.

Once in the main screen, you are greeted with 8 buttons. Each of these buttons have a specific function described below:

* “Delete Account” - deletes the current account
* “Connect” - automatically connects a pacemaker and retrieves serial number if successful.
* “Logout” - signs out of the current account
* “Choose Pacing Mode” - gives a popup with 9 buttons, press the desired pacing mode to set it
  + The current pacing mode is shown on the main screen left of the button
* “Change Programmable Parameters” - changes your desired parameter by the number inputted into the popup window.
  + Click the “Set Value” button on the popup to set the value
  + You can see the updated parameter near the bottom of the main screen
* “Load Previous” - Loads a previous set of parameters created in an earlier session by the current user.
* “Deploy Parameters” - Sends the set programmable parameters displayed above the button to a connected pacemaker
* “Open Heartbeat Graph” - Opens an egram of a connected heart
  + “Start” - Starts reading values from the heart
  + Close, ends communication of egram values and closes the display.

Additionally there is text at the top of the screen displaying the current user logged in, with a light beside it to check if hardware is successfully connected. The light will show red if there is no hardware, green if there is. The identifier will be shown beside the light.

## Modules

Serial Functions

Overview:

* Methods to initiate serial communication

Functions:

* serialConnect()
  + Sets the correct COM port
  + Connects to the appropriate port and notifies the user on if the connection has failed or connected.
* Serial Request()
  + Prepares the board to send information
  + Uses 0x22, which allows the stateflow to call sendParameters()
  + There is a slight delay between serialRequest() and serialReceive() since if you call serialRecieve too fast, the number of bytes sent from the board may not match the amount expected, which can lead to an error.
* serialReceive()
  + Reads the values from the pacemaker then returns it through the return inputRead command.
  + Returns a tuple array of data
* serialSend()
  + Changes the AtrAmp, VentAmp, AtrSens and VentSens values to duty cycles.
  + Takes those duty cycles as well as all the other values of the other parameters and sends them all to the connected pacemaker device.
  + Called via “deploy” button.

### WindowManager

Overview:

* Kivy ScreenManager
* controls the current screen viewed.
  + manageWin.current = "window name here"
* Controls transitions between screens
  + manageWin.transition = “insert transition here”

### WelcomeWindow

Overview:

* Class type: Kivy Screen
* This module includes the first screen the user will visit upon the launch of the application.
* This module allows the user to navigate to either the register page or the login page.
* It also prevents more than 10 users from registering an account.

Functions:

* goToLogin(self):
  + Sets the WindowManager current screen to the login window
  + Uses the Slide Transition, with direction set to “up”
* goToReg(self):
  + Checks the userDatabase.users dictionary to see if less than 10 accounts are saved. If 10 accounts already exist, call accountLimitReached(), which displays a message.
  + Sets the WindowManager current screen to the registration window
  + Uses the Slide Transition, with direction set to “right”

### LoginWindow

Overview:

* Class type: Kivy Screen
* This class generates text fields for the user to enter their username and password. If the user enters correct credentials, they are moved to the main page.
* Otherwise they are given an error message.
* There is also the option to go back to the Welcome window.

Functions:

* btnLogin(self):
  + Creates variables user and password using the corresponding text inputs from user.
  + These values are verified with userDatabase.credentialCheck(user, password).
* btnBack(self):
  + Calls self.reset()
  + Uses the window manager to return to the welcome screen with a “downward” slide transition.
* reset(self):
  + Resets the text fields to empty.

### RegisterWindow

Overview:

* Class type: Kivy Screen
* Generates text fields for First Name, Last Name, username, and password.

Functions:

* regUser(self):
  + Creates variables for all the fields
  + Checks the fields are notEmpty. If not, resets form, self.reset() and calls invalidRegister(), which displays an error message.
  + Checks the entered strings for illegal values. If not, resets form, self.reset() and calls invalidRegister(), which displays an error message.
  + Adds the credentials to the userDatabase with userDatabase.add\_user(username, fname, lname, password)
  + Calls self.reset()
  + Transitions to welcomeWin using slide transition left and calls registerComplete(), which generates a popup “Success”
* notEmpty(self,username,firstName,lastName,password):
  + returns True (1) if all the fields != “” (empty string)
  + else 0
* noBadChars(self,username,firstName,lastName,password):
  + Checks all the variables for illegal chars
* btnBack(self):
  + Calls self.reset()
  + Uses the window manager to return to the welcome screen with a “left” slide transition.
* reset(self):
  + Resets the text fields to empty strings.

### MainWindow

Overview:

* Class type: Kivy Screen
* This is the main page for the application.
* All the pacemaker parameters and the current pacing mode is displayed.
* Also allows the user to delete (top left) and logout of their account (top right).

Functions:

* on\_enter(self, \*args):
  + Called the mainWindow screen is set as the current screen.
  + Displays the active user label as the current user’s First Name
  + Displays the active pacing mode, if set
  + Displays all pacing parameters\*\*\*\*
  + If the hardware is connected, sets the indicator to green, else red.
* logout(self):
  + Called when the user clicks the logout button.
  + Changes the current window to the welcomeWin screen
  + Calls the signOut\_Complete() function
* deleteAccount(self):
  + Calls the database function, remove\_user, sending currentUsername as a parameter
  + Changes the current window to the welcomeWin screen
  + Calls the userDeleted() function
* open\_modeSelector(self):
  + When the change pacing button is clicked, shows the corresponding popup.
* open\_programmableParameters(self):
  + When the programmable parameters button is clicked, shows the corresponding popup.
* open\_heartbeatGraph(self):
  + When the heartbeat graph button is clicked and there is a device connected to the dcm, the egram popup is shown.
* serialConnectMain(self):
  + When the hardware is connected, the connection indicator turns green and enters the serial number of the device that is connected.
  + Otherwise an error displays telling the user the device did not find a device.
* deploy(self):
  + When the pacing mode is not “Not Set”, all the programmable parameters as well as the username and pace mode is written into the “user\_data.txt” file.
  + The data is also deployed via serial communication to the connected device.
* loadPrevious(self):
  + The “user\_data.txt” file is read for every entry in the file that is separated by the string “;”. Essentially, the username, pace mode and all programmable parameters are read.
  + The group of set functions (setLRL, setURL, SetAtrAmp, etc.) is used to load the data from the “user\_data.txt” file to the Pacemaker App.
  + The programmable parameters are then displayed onto the main window of the Pacemaker App.
* reset(self):
  + Sets all programmable parameters as well as pacing mode to “Not Set”.
  + This function is used when a user logs out.

Local Variables:

* currentUsername
  + Holds the username of the current user logged in
* indicatorColour
  + This is a ListProperty([1,0,0,1])
  + RGBA. Defaults to red. Set to green when hardware is connected.

Global Variables:

* pacingMode
  + String, “AOO”,”VOO”, etc.
* Programmable Parameters
  + LRL
  + MSR
  + AtrAmp
  + VentAmp
  + AtrPulseWidth
  + VentOulseWidth
  + VRP
  + ARP
  + AtrSens
  + VentSens
  + reationTime
  + recoverTime
  + AVDelay
  + resFactor
  + AccThreshold1
  + AccThreshold2
  + AccThreshold3
* hardwareConnected
  + Bool, true when board is connected
* popupWindow
  + Variable that represents the current active popup.
  + Used to call popup.dismiss()

### modeSelectorPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout of the pacing mode selector popup

Functions:

* closePopup(self):
  + Dismisses the popup
* setPacingMode(self,mode):
  + Calls the function setPacingModetext and sends the parameter “mode” depending on the button chosen by the user

heartbeatGraphPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout of the heartbeat graph/egram popup

Functions:

* \_\_init\_\_(self,):
  + Initializes two graphs, one for atrium and other for ventricle, and plots the data using MeshLinePlot from the kivy garden library
* startHeartbeat(self):
  + Starts showing the current heartbeat on the graphs
  + Updates the graph every 0.05 seconds
* stopHeartbeat(self):
  + Stops the display of the heartbeat
* get\_value\_atr(self,dt):
  + Uses serial communication to acquire atrial data from the pacemaker and stores it in global variable tupleInput.
  + Sends the data from the pacemaker to the graphs to be plotted
* get\_value\_vent(self,dt):
  + Uses serial communication to acquire ventricular data from the pacemaker and stores it in global variable tupleInput.
  + Sends the data from the pacemaker to the graphs to be plotted

Global Variables:

* tupleInput
  + Variable to store heartbeat data from pacemaker
* ATR\_graphArray
  + Variable to store graph points for plotting atrium heartbeat
* VENT\_graphArray
  + Variable to store graph points for plotting ventricle heartbeat

programmableParameterPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout of the programmable parameters popup

Functions:

* setIndex(self, num):
  + Stores the index of which parameter is to be changed (1 for LRL, 2 for URL, 3 for AtrAmp . . etc)
  + Index is a global variable for usage in other functions
* open\_textInput(self, title):
  + Opens up a popup with text input for input of values to the desired programmable parameter
  + Takes in string ‘title’ to set the title of the popup
* closePopup(self):
  + Dismisses the popup

textInputPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout of the pacing mode selector popup

Functions:

* selectProgParam(self):
  + Stores the corresponding programmable parameter based on the index with the number in the text field
  + Adds several ranges for the different programmable parameters where in which if the appropriate range is not satisfied then a paramErrorPopup() function activates.
* closePopup(self):
  + Dismisses the popup

### errorPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout and contents of the generic error popup

Functions:

* closePopup(self):
  + Dismisses the popup

genericErrorPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout and contents of the generic error popup

Functions:

* closePopup(self):
  + Dismisses the popup

### errorMaxPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout and contents of the maximum accounts reached dialogue popup

Functions:

* closePopup(self):
  + Dismisses the popup

### paramErrorPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout and contents of the invalid parameter range dialogue popup

Functions:

* closePopup(self):
  + Dismisses the popup

### successPopup

Overview:

* Class type: Kivy FloatLayout
* This defines the layout and contents of the generic auto timeout success popup

Functions:

* \_\_init\_\_(self, \*\*kwargs):
  + Calls closePopup() after it’s been displayed for 1 second
* closePopup(self):
  + Dismisses the popup

### Database

Overview:

* Responsible for handling the stored users and their credentials.
* Takes input from a local txt file.

Functions:

* \_\_init\_\_(self, filename):
  + Takes a filename as a parameter.
  + Calls the load() function
* load(self):
  + Open the specified file in read mode
  + Create a dictionary with all the usernames and their credentials found in the file.
  + Close the file
* save(self):
  + Open the file in write mode
  + Write each user (and their credentials) found in the users dictionary to the file
* get\_user(self,username):
  + Checks if the username exists in the users dictionary.
  + If it does, it returns the username and their associated credentials.
* add\_user(self, username,firstName,lastName,password):
  + Checks to be sure the username doesn’t already exist in the dictionary.
  + Adds the username as a “key” and the First Name, Last Name and Password as “values”.
* credentialCheck(self, username, password):
  + used to verify login credentials of a returning user.
  + First checks if the username is found in the users dictionary.
  + Then checks if the entered password matches the stored password value found in the dictionary.
* remove\_user(self, username):
  + Removes the username key and values from the dictionary.

### PacemakerApp

Overview:

* Starts the app. Runs the ScreenManager.

### Global Functions

setPacingModetext(mode):

* takes a string parameter mode as input, and sets the global pacingMode = mode
* Global Variables:
  + pacingMode

setLRL(num):

* takes a float parameter num as input, and sets the global LRL\_value = num
* Typecasts float parameter num to string and sets the global LRL = str(num) +” BPM”
* Prints “LRL: “ + LRL
* Global Variables:
  + LRL\_value
  + LRL

setMSR(num):

* takes a float parameter num as input, and sets the global MSR\_value = num
* Typecasts float parameter num to string and sets the global MSR = str(num) +” BPM”
* Global Variables:
  + MSR\_value
  + MSR

setAtrAmp(num):

* takes an int parameter num as input, and sets the global AtrAmp\_value = num
* Typecasts float parameter num to string and sets the global AtrAmp = str(num) +” V”
* Prints “AtrAmp: “ + AtrAmp
* Global Variables:
  + AtrAmp\_value
  + AtrAmp

setAtrPulseWidth(num):

* takes an int parameter num as input, and sets the global AtrPulseWidth\_value = num
* Typecasts float parameter num to string and sets the global AtrPulseWidth = str(num) +” ms”
* Prints “AtrPulseWidth: “ + AtrPulseWidth
* Global Variables:
  + AtrPulseWidth\_value
  + AtrPulseWidth

setVentAmp(num):

* takes an int parameter num as input, and sets the global VentAmp\_value = num
* Typecasts float parameter num to string and sets the global VentAmp = str(num) +” V”
* Global Variables:
  + VentAmp\_value
  + VentAmp

setVentPulseWidth(num):

* takes an int parameter num as input, and sets the global VentPulseWidth\_value = num
* Typecasts float parameter num to string and sets the global VentPulseWidth = str(num) +” ms”
* Prints “VentPulseWidth: “ + VentPulseWidth
* Global Variables:
  + VentPulseWidth\_value
  + VentPulseWidth

setVRP(num):

* takes an float parameter num as input, and sets the global VRP\_value = num
* Typecasts float parameter num to string and sets the global VRP = str(num) +” ms”
* Prints “VRP: “ + VRP
* Global Variables:
  + VRP\_value
  + VRP

setARP(num):

* takes an float parameter num as input, and sets the global ARP\_value = num
* Typecasts float parameter num to string and sets the global ARP = str(num) +” ms”
* Prints “ARP: “ + ARP
* Global Variables:
  + ARP\_value
  + ARP

setAtrSens(num):

* takes an float parameter num as input, and sets the global AtrSens\_value = num
* Typecasts float parameter num to string and sets the global AtrSens = str(num) +” V”
* Prints “AtrSensitivity: “ + AtrSens
* Global Variables:
  + AtrSens\_value
  + AtrSens

setVentSens(num):

* takes an float parameter num as input, and sets the global VentSens\_value = num
* Typecasts float parameter num to string and sets the global VentSens = str(num) +” V”
* Prints “VentSensitivity: “ + VentSens
* Global Variables:
  + VentSens\_value
  + VentSens

setreactionTime(num):

* takes an float parameter num as input, and sets the global reactionTime\_value = num
* Typecasts float parameter num to string and sets the global reactionTime = str(num) +” ms”
* Prints “reactionTime: “ + reactionTime
* Global Variables:
  + reactionTime\_value
  + reactionTime

setrecoveryTime(num):

* takes an float parameter num as input, and sets the global recoveryTime\_value = num
* Typecasts float parameter num to string and sets the global recoveryTime = str(num) +” ms”
* Prints “recoveryTime: “ + recoveryTime
* Global Variables:
  + recoveryTime\_value
  + recoveryTime

setAVDelay(num):

* takes an float parameter num as input, and sets the global AVDelay\_value = num
* Typecasts float parameter num to string and sets the global AVDelay = str(num) +” ms”
* Prints “AVDelay: “ + AVDelay
* Global Variables:
  + AVDelay\_value
  + AVDelay

setresFactor(num):

* takes an float parameter num as input, and sets the global resFactor\_value = num
* Typecasts float parameter num to string and sets the global resFactor = str(num) +” ”
* Prints “resFactor: “ + resFactor
* Global Variables:
  + resFactor\_value
  + resFactor

setAccThreshold1(num):

* takes an double parameter num as input, and sets the global AccThreshold1\_value = num
* Typecasts double parameter num to string and sets the global AccThreshold1 = str(num) + ” ”
* Prints “Still Threshold: “ + AccThreshold1
* Global Variables:
  + AccThreshold1\_value
  + AccThreshold1

setAccThreshold2(num):

* takes an double parameter num as input, and sets the global AccThreshold2\_value = num
* Typecasts double parameter num to string and sets the global AccThreshold2 = str(num) + ” ”
* Prints “Walking Threshold: “ + AccThreshold2
* Global Variables:
  + AccThreshold2\_value
  + AccThreshold2

setAccThreshold3(num):

* takes an double parameter num as input, and sets the global AccThreshold3\_value = num
* Typecasts double parameter num to string and sets the global AccThreshold3 = str(num) + ” ”
* Prints “Running Threshold: “ + AccThreshold3
* Global Variables:
  + AccThreshold3\_value
  + AccThreshold3

invalidLogin():

* Creates a new popup window object and initializes it with the determined content. Title = “Login Error”
* Calls the Popup class’ function open()
* Local variables:
  + show
    - New object of type errorPopup()
* Global variables:
  + popupWindow
    - New Popup object

invalidRegister():

* Creates a new popup window object and initializes it with the determined content. Title = “Username not allowed or is already taken”
* Calls the Popup class’ function open()
* Local variables:
  + show
    - New object of type errorPopup()
* Global variables:
  + popupWindow
    - New Popup object

registerComplete():

* Creates a new popup window object and initializes it with the determined content. Title = “You are now Registered”
* Calls the Popup class’ function open()
* Local variables:
  + show
    - New object of type errorPopup()
* Global variables:
  + popupWindow
    - New Popup object

signOut\_Complete():

* Creates a new popup window object and initializes it with the determined content. Title = “You have now signed out”
* Calls the Popup class’ function open()
* Local variables:
  + show
    - New object of type errorPopup()
* Global variables:
  + popupWindow
    - New Popup object

accountLimitReached():

* Creates a new popup window object and initializes it with the determined content. Title = “Error”
* Calls the Popup class’ function open()
* Local variables:
  + show
    - New object of type errorPopup()
* Global variables:
  + popupWindow
    - New Popup object

userDeleted():

* Creates a new popup window object and initializes it with the determined content. Title = “Account Deleted”
* Calls the Popup class’ function open()
* Local variables:
  + show
    - New object of type errorPopup()
* Global variables:
  + popupWindow
    - New Popup object

genericError():

* Creates a new popup window object and initializes it with the determined content. Title = “Error”
* Calls the Popup class’ function open()
* Local variables:
  + show
    - New object of type genericErrorPopup()
* Global variables:
  + popupWindow
    - New Popup object

noDeviceError():

* Creates a new popup window object and initializes it with the determined content. Title = “No Device Connected”
* Calls the Popup class’ function open()
* Local variables:
  + show
    - New object of type genericErrorPopup()
* Global variables:
  + popupWindow
    - New Popup object

## Variables

### Global State Variables

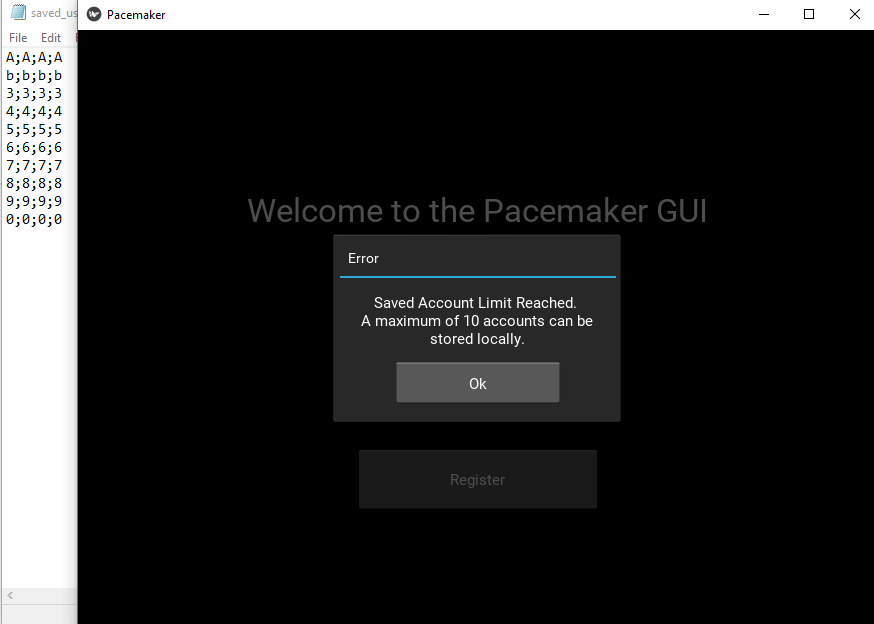
* General Variables:
  + pacingMode
    - Stores the current pacing mode of the pacemaker
    - Used by **display self.display\_active\_pacingMode.text** to display the variable on the main window
    - Used by **def setPacingModetext** to set the variable based on user input
  + hardwareConnected
    - Checks if there is a pacemaker connected to the device
    - Used by **def on\_enter** to determine the colour of the indicator on the mains screen
  + Index
    - Stores the programmable parameter being set - indexed according to the list below
    - Used by **def setIndex** to set the index variable depending on the button pressed in the “Change Programmable Parameters” popup
    - Used by **def selectProgParam** to check which programmable parameter to change
* Programmable Parameters:
  + LRL - 1
  + MSR - 2
  + AtrAmp - 3
  + AtrPulseWidth - 4
  + ARP - 5
  + VentAmp - 6
  + VentPusleWidth - 7
  + VRP - 8
  + AtrSens - 9
  + VentSens - 10
  + reactionTime - 11
  + recoveryTime - 12
  + AVDelay - 13
  + resFactor - 14
  + AccThreshold1 - 15
  + AccThreshold2 - 16
  + AccThreshold3 - 17
    - All are used by their respective “set” parameters (LRL - **def setLRL,** AtrAmp - **def setAtrAmp**, etc.)
* Windows:
  + popupWindow
    - Stores the current popup - used in setting the pacing mode and programmable parameters
    - Used by all popup windows (**modeSelectorPopup, programmableParametersPopup, textInputPopup, errorPopup, errorMaxPopup, sucessPopup**), error popups (**invalidLogin, invalidRegister, registerComplete, signOut\_Complete, accountLimitReached, userDeleated**) and selection on the main screen (**open\_modeSelector, open\_programmableParameters**)
  + popupWindow\_editParameter
    - Stores a second popup window - used in inputting a value into a programmable parameter
    - Used by **popupWindow\_editParameter** in the second popup

Testing

**10 user test -** The user database should only allow 10 users stored locally

Testing methods:

* Inputting 11 different usernames
  + Result: account limit error popup when trying to register the 11th - **Success**
* Inputting 10 usernames, deleting one, register two more
  + Result: account limit error popup when trying to register the second - **Success**

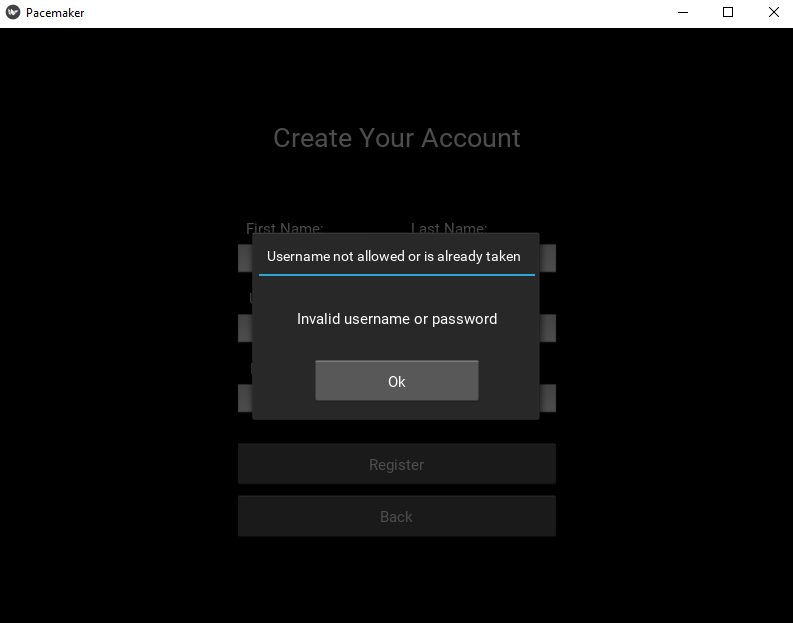
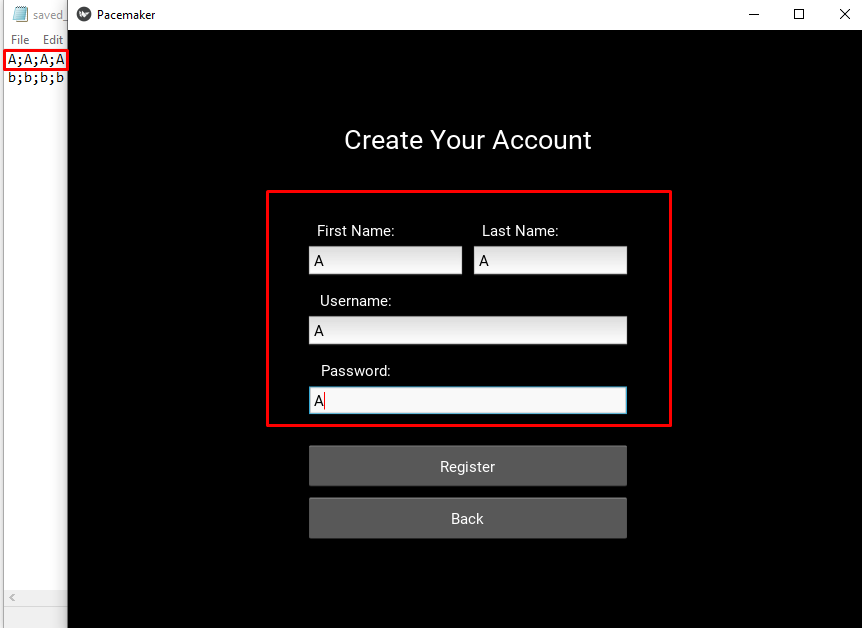
****

**Figure 1 -** The program only allows you to register 10 accounts

**Register testing -** Should check for invalid characters and already registered usernames and passwords

Testing methods:

* Invalid character (ie: ;) for “First Name”
  + Result: “Username not allowed or is already taken” error - **Success**
* Invalid character (ie: ;) for “Last Name”
  + Result: “Username not allowed or is already taken” error - **Success**
* Invalid character (ie: ;) for “Username”
  + Result: “Username not allowed or is already taken” error - **Success**
* Invalid character (ie: ;) for “Password”
  + Result: “Username not allowed or is already taken” error - **Success,**
* Register with username and password that already exists
  + Result: “Username not allowed or is already taken” error - **Success**

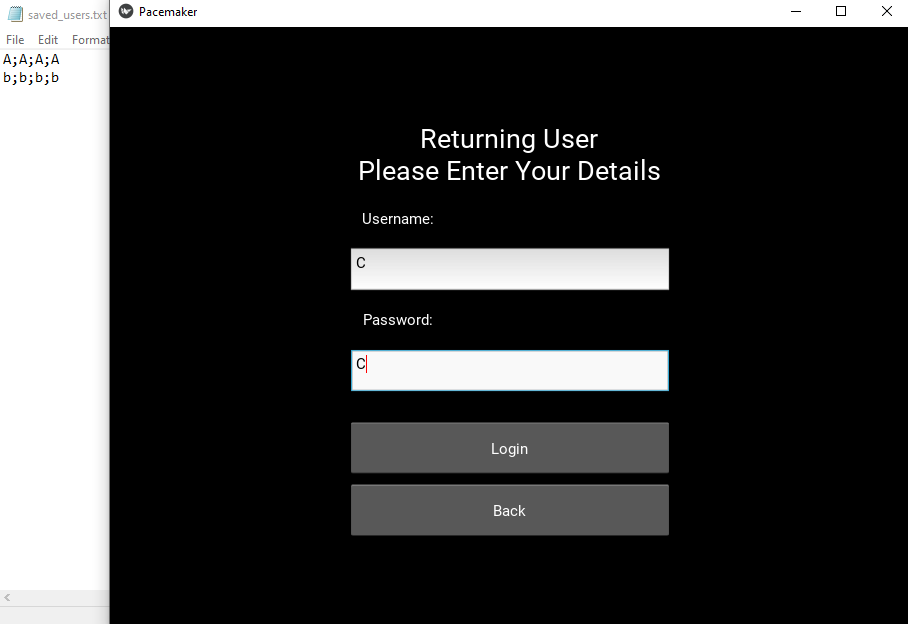
****

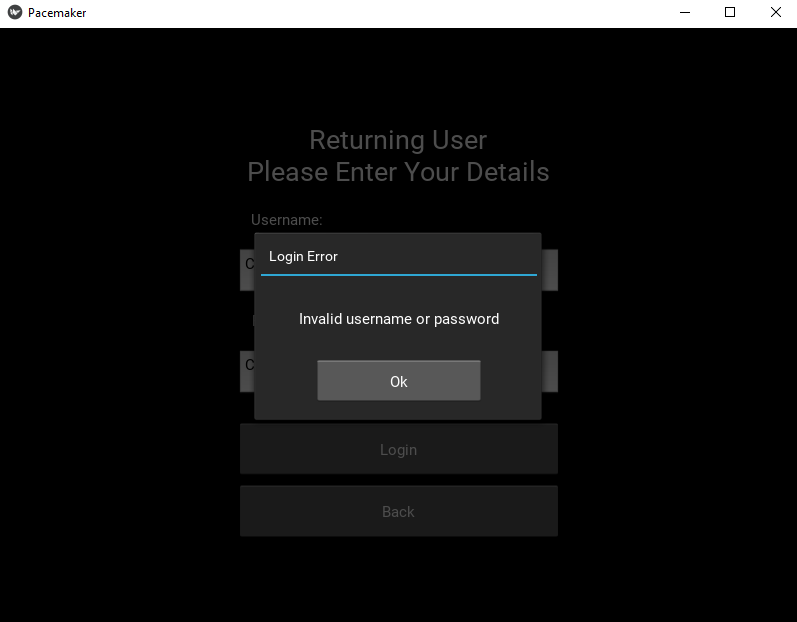
**Figure 2 -** Registering an account that is already taken results in an error popup

**Login Testing -** Should only allow registered users into the main program

Testing methods:

* Try to login with username + password combination that is not registered
  + Result: Login Error - **Success**

****

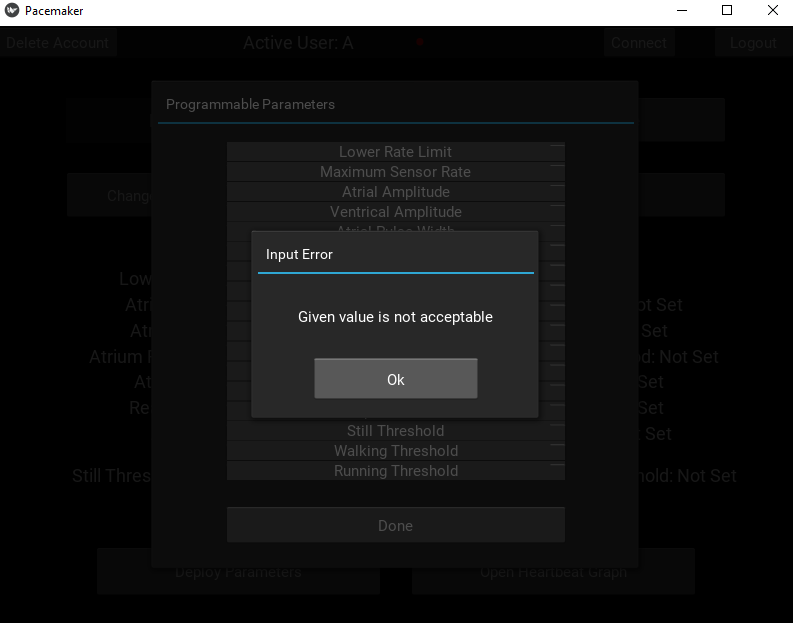
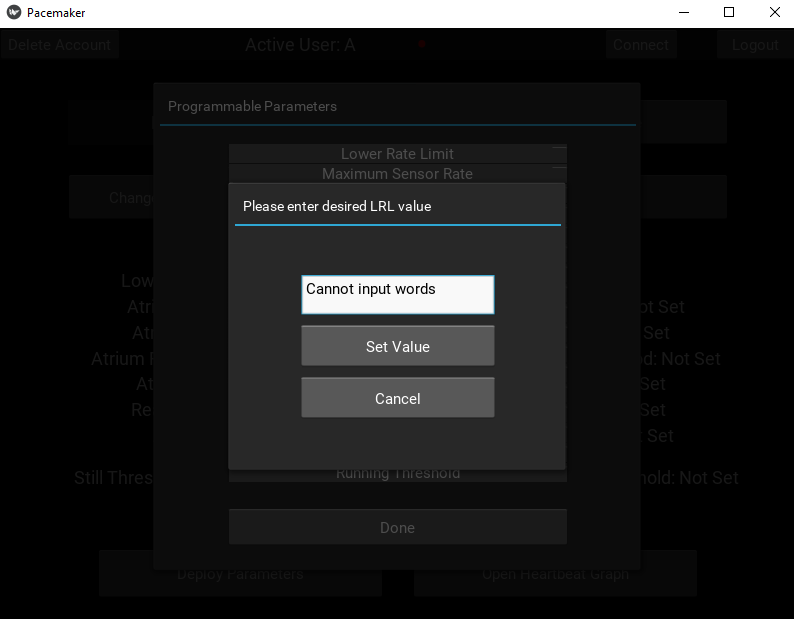
****

**Figure 3 -** When entering an invalid login the app does not let you in

**Programmable Parameters Testing -** Should only allow numerical submissions

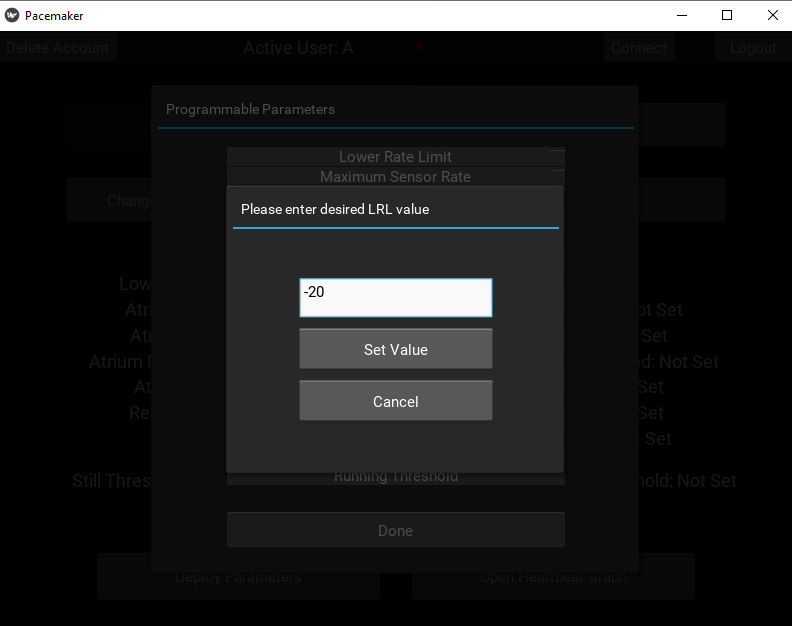
Testing methods:

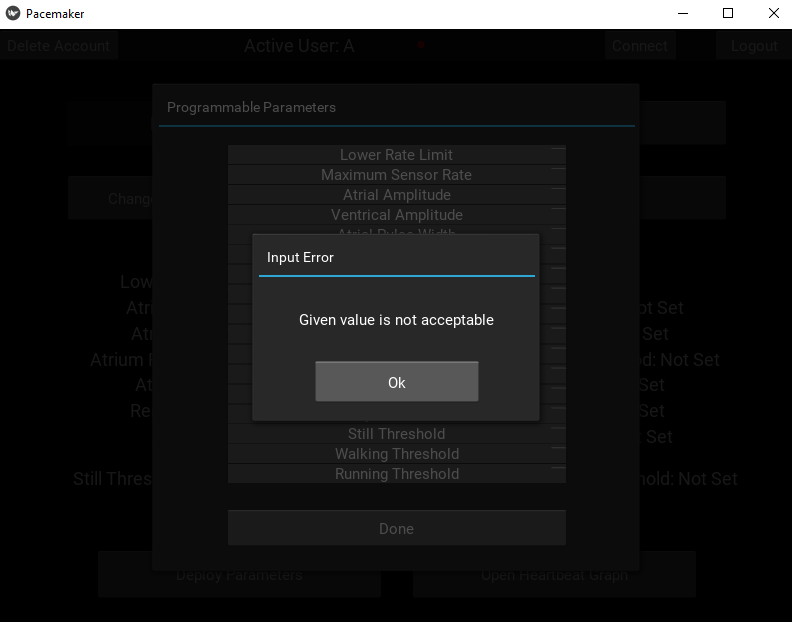
* Use non-numerical submission (!@#, asdf, etc.)
  + Result: Program crashes - **Success**

****

**Figure 4 -** When inputting words (Strings) an error popup appears stating that the input is invalid

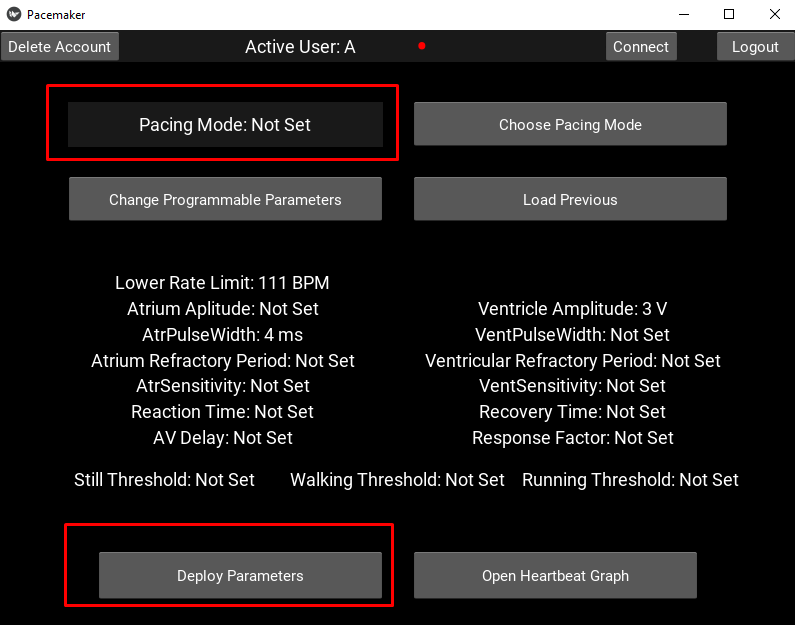
* Use values outside of parameter ranges (**for example:** amplitude V = 100V or -20 V, LRL = 5 bpm, AtrPulseWidth = 0 ms, VRP = 1000ms, etc.)
  + Result: generic error popups - **Success**

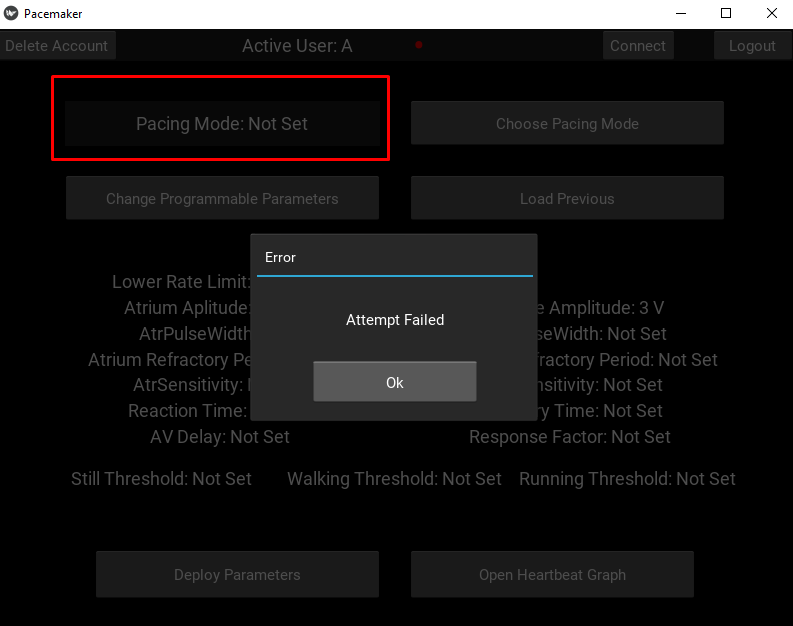
****

****

**Figure 5 -** When inputting an invalid parameter value and error pops up indicating that that value is not valid

* App recognizes that the user can’t deploy the parameters without selecting a pacing mode
  + Result: generic error popups - **Success**

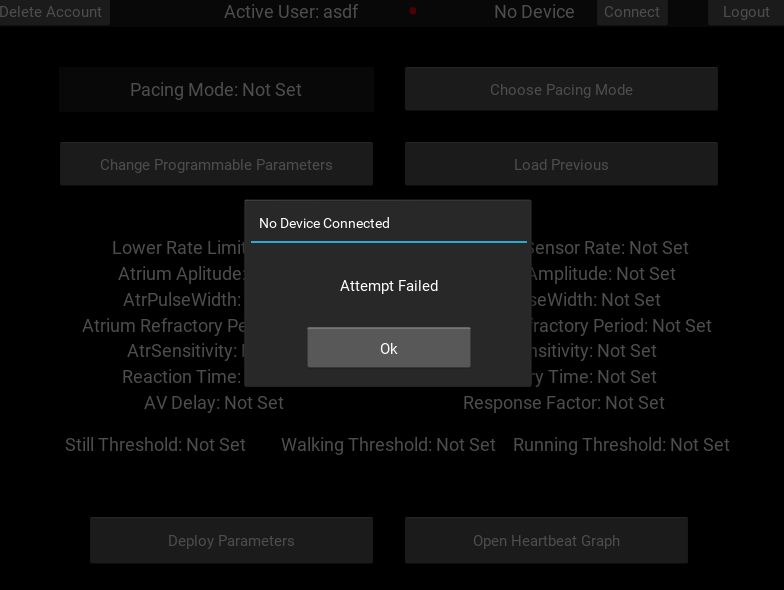
****

****

**Figure 6 -** Attempt to deploy is unsuccessful because a pacing mode is not set

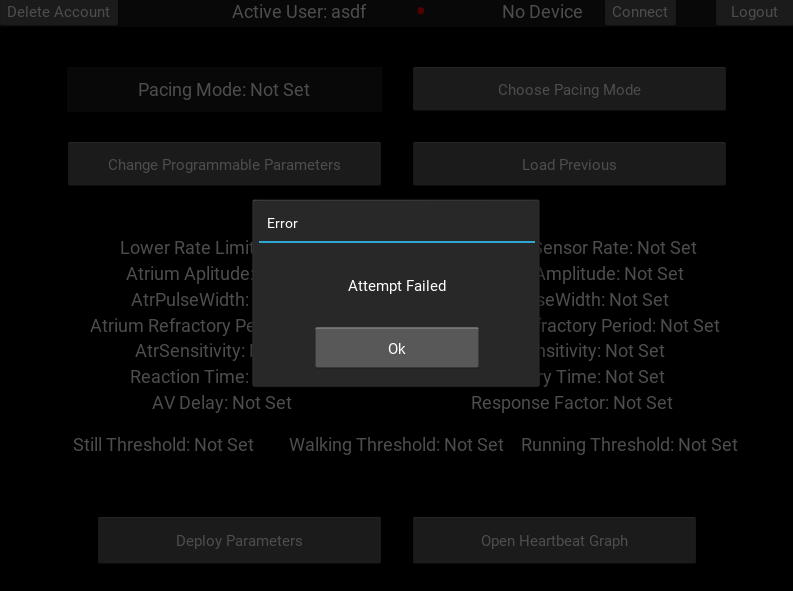
**Connection Testing -** Should show error when no device is connected and connection to the device is attempted

* Trying to open heartbeat graph when there is no connection
  + Result: generic error popup - **Success**

****

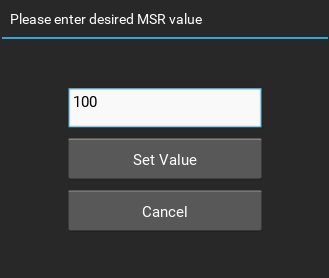
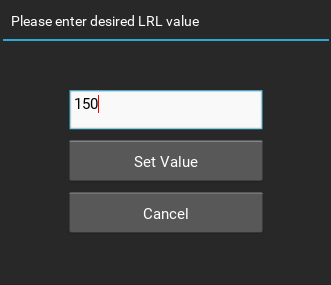
**Figure 7 -** Attempt to open heartbeat graph is unsuccessful because a device is not connected

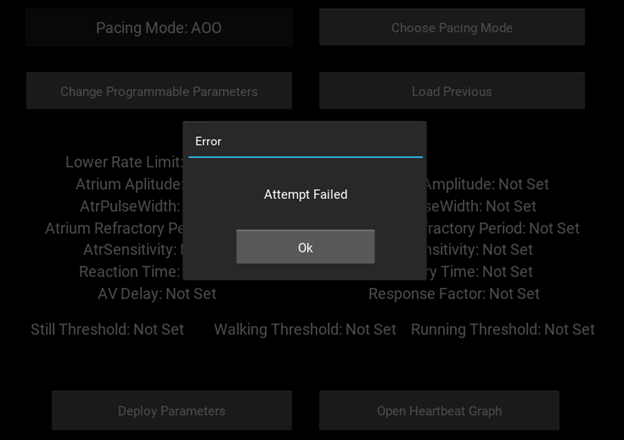
* Trying to deploy parameters and pacing mode when there is no connection
  + Result: generic error popup - **Success**



**Figure 8 -** Attempt to deploy is unsuccessful because no device is connected

## **MSR\_Value >= LRL\_Value Test** - Checks if this condition is true, otherwise the parameters can not be deployed.





**Figure 9** - MSR and LRL values can’t be deployed

## 

## Future Changes

### Requirements

* Add additional modes including DDD, DDI, VDDD, VDDR, etc.
* Add the functionality for the doctor to see past programmable parameters used on previous dates with previous patients specifically (use board serial number)
* Round input values to correct number of decimals according to the pacemaker documentation.
* Ability for DCM to connect to the board and ask what parameters are currently being used. Add all programmable parameters to the serialReceive section. This would mean potentially slower response for egram data which could affect performance.

### Design Decisions

* DCM could tell the user what the bounds are for each parameter.
* Optimize the method for generating unique popups. Limit the number of functions and use input parameters instead.
* Separate modules into different files
* Improve egram graph
  + X-axis shifting with time
  + Improve readability on y-axis
  + Have real time averages displayed
* Improve GUI interface for users
  + Add more colours on buttons for increased visibility
  + Change button layout in popups for easier clicking and readability