**SE 3K04 Assignment 1**

**Part 2: Pacemaker DCM**

**Group 9**

Kevin Cheung

Forrest Herman

Atiqul Islam

Jingming Liu

**Sunday, November 1, 2020**

Table of Contents

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

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

[Requirements:](#_n3bjtivijdrw) 3

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

[WindowManager](#_ee43liizd5iq) 4

[WelcomeWindow](#_to3ok129q808) 4

[LoginWindow](#_v62xkt1n56rr) 4

[RegisterWindow](#_o84ey0uf1j3) 5

[MainWindow](#_qodukb8zkyw) 5

[modeSelectorPopup](#_py563wzguacq) 6

[errorPopup](#_lxf01wjybgzq) 7

[errorMaxPopup](#_3qrs0rgc1dk8) 8

[successPopup](#_vfoio3buhjbh) 8

[Database](#_lhsdkicegmis) 8

[PacemakerApp](#_4mblsi9zsgku) 9

[Global Functions](#_y83325ufpyqm) 9

[**Variables**](#_xitlhabf0ke8) **11**

[Global State Variables](#_jutm5t25kfgm) 11

[**Future Changes**](#_lcpp2y331ge8) **15**

[Requirements](#_obyy57llhvdy) 15

[Design Decisions](#_g10stdd2cap6) 15

## Introduction

This DCM, built using Python 3 and Kivy, creates an interface for the Pacemaker software design described in 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.

## Specifications

### Requirements:

* 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.

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 4 buttons. Click “Delete Account” to delete the current account, “Logout” to sign out of the current account. The “Choose Pacing Mode” button will give a popup with 4 buttons, press the desired pacing mode to set it. You can see the current pacing mode on the main screen left of the button. The last button is the “Change Programmable Parameters” button in the middle of the screen. Here you can change your desired parameter by inputting a number into the popup window with the text input. Click the “Set Value” button on the popup to set the value. You can see the updated parameter on the main screen under the button.

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 board’s model # or identifier will be shown beside the light.

## Modules

### 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.

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.
* 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

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
* 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

### 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

### 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 = num
* Global Variables:
  + LRL

setURL(num):

* takes a float parameter num as input, and sets the global URL = num
* Global Variables:
  + URL

setAtrAmp(num):

* takes an int parameter num as input, and sets the global AtrAmp = num
* Global Variables:
  + AtrAmp

setVentAmp(num):

* takes an int parameter num as input, and sets the global VentAmp = num
* Global Variables:
  + VentAmp

setAtrPulseWidth(num):

* takes an int parameter num as input, and sets the global AtrPulseWidth = num
* Global Variables:
  + AtrPulseWidth

setVentPulseWidth(num):

* takes an int parameter num as input, and sets the global VentPulseWidth = num
* Global Variables:
  + VentPulseWidth

setVRP(num):

* takes an int parameter num as input, and sets the global VRP = num
* Global Variables:
  + VRP

setARP(num):

* takes an int parameter num as input, and sets the global ARP = num
* Global Variables:
  + ARP

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

## 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
  + URL - 2
  + AtrAmp - 3
  + AtrPulseWidth - 4
  + ARP - 5
  + VentAmp - 6
  + VentPusleWidth - 7
  + VRP - 8
    - 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**

**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 - **Partial Success, need to change error label to be more precise**
* Invalid character (ie: ;) for “Last Name”
  + Result: “Username not allowed or is already taken” error - **Partial Success, need to change error label to be more precise**
* 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 - **Partial Success, need to change error label to be more precise**
* Register with username and password that already exists
  + Result: “Username not allowed or is already taken” error - **Success**

**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**

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

Testing methods:

* Use non-numerical submission (!@#, asdf, etc.)
  + Result: the variable changes to the submission - **Fail, to be changed. Must allow only integer or double inputs.**
* Use values outside of parameter ranges (**for example:** amplitude V = 100V or -20 V, LRL = 5 bpm, AtrPulseWidth = 0 ms, VRP = 1000ms, etc.)
  + Result: the variable changes to the submission - **Fail, to be changed. Must implement bounds for each variable to respect the range of the parameters.**

## 

## Future Changes

### Requirements

* Finalize the DCM, adding communications between DCM and Pacemaker.
* Add DOO, AOOR, VOOR, AAIR, VVIR, DOOR, and DDDR mode options.
* The user interface shall be capable of visually indicating when a different PACEMAKER device is approached than was previously interrogated.
* Add restrictions on the input variables to ensure proper syntax
* Add the functionality for the doctor to see past programmable parameters used

### Design Decisions

* Create a graph of heartbeats and paces.
* Optimize the method for generating unique popups. Limit the number of functions and use input parameters instead.