
Assignment 2 - Pacemaker

SFWRENG 3K04

Thomas Chiang

Group 13

December 3rd, 2023

Rohan Chandan - chandr16 - 400379494

Haya Al Sami - alsamih - 400385506

Matthew Ferreira - ferrem15 - 400404811

Kresteen Al Shehadeh - alshehak - 400371813

Samarth Patel - pates199 - 400377664

Bisma Ali - alib10 - 400369507

Table of Contents

| | |
|---|----------|
| 1. Requirements Changes..... | 7 |
| 2. Design Decision Changes..... | 8 |
| 3. Modules..... | 8 |
| 3.1 DCM Modules..... | 8 |
| 3.1.1 Main.py..... | 8 |
| Public Functions..... | 9 |
| 3.1.1.1 open_serial_connection()..... | 9 |
| 3.1.1.2 send_data_via_serial(parameters)..... | 9 |
| 3.1.1.3 close_serial_connection()..... | 9 |
| 3.1.1.4 recieve_data_via_serial()..... | 10 |
| 3.1.1.5 get_curr_user()..... | 10 |
| 3.1.1.6 delete_user(username)..... | 10 |
| Global Variables:..... | 10 |
| Class Window():..... | 10 |
| Public Functions:..... | 10 |
| 3.1.1.7 show_frame(self,cont)..... | 10 |
| 3.1.1.8 reset_entry(self,entry,txt)..... | 11 |
| Global Variables:..... | 11 |
| Private Functions:..... | 11 |
| 3.1.1.9 __init__(self,*args,**kwargs)..... | 11 |
| Class Login(tk.Frame):..... | 11 |
| Public Functions:..... | 12 |
| 3.1.1.10 user_on_focusin(event)..... | 12 |
| 3.1.1.11 user_on_focusout(event)..... | 12 |
| 3.1.1.12 pass_on_focusin(event)..... | 12 |
| 3.1.1.13 pass_on_focusout(event)..... | 12 |
| 3.1.1.14 store_user(user)..... | 12 |
| 3.1.1.15 readUser()..... | 12 |
| Global Variables:..... | 13 |
| Private Functions:..... | 14 |
| 3.1.1.16 __init__(self,*args,**kwargs)..... | 14 |
| Class Register():..... | 14 |
| Public Functions:..... | 14 |
| 3.1.1.17 on_focusin(event)..... | 14 |
| 3.1.1.18 on_focusout(event)..... | 14 |
| 3.1.1.19 usernameExists(username)..... | 14 |
| 3.1.1.20 numRegistered()..... | 15 |
| 3.1.1.21 writeUser()..... | 15 |
| 3.1.1.22 registerCheck()..... | 15 |
| Global Variables:..... | 16 |
| Private Functions:..... | 17 |
| 3.1.1.23 __init__(self,*args,**kwargs)..... | 17 |

| | |
|--|----|
| Class Front():..... | 17 |
| Public Functions:..... | 17 |
| 3.1.1.24 updateMode(*args)..... | 17 |
| 3.1.1.25 displayConnection()..... | 17 |
| 3.1.1.26 sameDevice()..... | 18 |
| 3.1.1.27 on_show_frame(self,event)..... | 18 |
| Global Variables:..... | 18 |
| Private Functions:..... | 18 |
| 3.1.1.28 __init__(self,*args,**kwargs)..... | 18 |
| Class AOO():..... | 18 |
| Public Functions:..... | 19 |
| 3.1.1.29 load_saved_data(self)..... | 19 |
| 3.1.1.30 is_digit_check(P)..... | 19 |
| 3.1.1.31 save_text()..... | 19 |
| 3.1.1.32 inputs_correct()..... | 19 |
| 3.1.1.33 reset_ecg()..... | 20 |
| 3.1.1.34 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)..... | 20 |
| 3.1.1.35 update_ecg_continuously(self)..... | 20 |
| 3.1.1.36 generate_ecg()..... | 21 |
| 3.1.1.37 stop_ecg_update(self)..... | 21 |
| 3.1.1.38 resume_ecg_update(self)..... | 21 |
| Global Variables:..... | 22 |
| Private Functions:..... | 22 |
| 3.1.1.39 __init__(self,*args,**kwargs)..... | 22 |
| Class AAI():..... | 22 |
| Public Functions:..... | 22 |
| 3.1.1.40 load_saved_data(self)..... | 22 |
| 3.1.1.41 is_digit_check(P)..... | 22 |
| 3.1.1.42 save_text()..... | 23 |
| 3.1.1.43 inputs_correct()..... | 23 |
| 3.1.1.44 reset_ecg()..... | 23 |
| 3.1.1.45 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)..... | 24 |
| 3.1.1.46 update_ecg_continuously(self)..... | 24 |
| 3.1.1.47 generate_ecg()..... | 24 |
| 3.1.1.48 stop_ecg_update(self)..... | 25 |
| 3.1.1.49 resume_ecg_update(self)..... | 25 |
| Global Variables:..... | 25 |
| Private Functions:..... | 25 |
| 3.1.1.50 __init__(self,*args,**kwargs)..... | 25 |
| Class VOO():..... | 26 |
| Public Functions:..... | 26 |
| 3.1.1.51 load_saved_data(self)..... | 26 |

| | |
|--|----|
| 3.1.1.52 is_digit_check(P)..... | 26 |
| 3.1.1.53 save_text()..... | 26 |
| 3.1.1.54 inputs_correct()..... | 27 |
| 3.1.1.55 reset_ecg()..... | 27 |
| 3.1.1.56 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)..... | 27 |
| 3.1.1.57 update_ecg_continuously(self)..... | 28 |
| 3.1.1.58 generate_ecg()..... | 28 |
| 3.1.1.59 stop_ecg_update(self)..... | 28 |
| 3.1.1.60 resume_ecg_update(self)..... | 28 |
| Global Variables:..... | 29 |
| Private Functions:..... | 29 |
| 3.1.1.61 __init__(self,*args,**kwargs)..... | 29 |
| Class VVI():..... | 29 |
| Public Functions:..... | 29 |
| 3.1.1.62 load_saved_data(self)..... | 29 |
| 3.1.1.63 is_digit_check(P)..... | 30 |
| 3.1.1.64 save_text()..... | 30 |
| 3.1.1.65 inputs_correct()..... | 30 |
| 3.1.1.66 reset_ecg()..... | 30 |
| 3.1.1.67 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)..... | 31 |
| 3.1.1.68 update_ecg_continuously(self)..... | 31 |
| 3.1.1.69 generate_ecg()..... | 32 |
| 3.1.1.70 stop_ecg_update(self)..... | 32 |
| 3.1.1.71 resume_ecg_update(self)..... | 32 |
| Global Variables:..... | 32 |
| Private Functions:..... | 33 |
| 3.1.1.72 __init__(self,*args,**kwargs)..... | 33 |
| Class AOOR():..... | 33 |
| Public Functions:..... | 33 |
| 3.1.1.73 load_saved_data(self)..... | 33 |
| 3.1.1.74 is_digit_check(P)..... | 33 |
| 3.1.1.75 save_text()..... | 33 |
| 3.1.1.76 inputs_correct()..... | 34 |
| 3.1.1.77 chosen_activity_thresh(self, *args)..... | 34 |
| 3.1.1.78 reset_ecg()..... | 34 |
| 3.1.1.79 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)..... | 35 |
| 3.1.1.80 update_ecg_continuously(self)..... | 35 |
| 3.1.1.81 generate_ecg()..... | 35 |
| 3.1.1.82 stop_ecg_update(self)..... | 36 |
| 3.1.1.83 resume_ecg_update(self)..... | 36 |
| Global Variables:..... | 36 |
| Private Functions:..... | 36 |

| | |
|---|----|
| 3.1.1.84 __init__(self,*args,**kwargs)..... | 36 |
| Class VOOR():..... | 37 |
| Public Functions:..... | 37 |
| 3.1.1.85 load_saved_data(self)..... | 37 |
| 3.1.1.86 is_digit_check(P)..... | 37 |
| 3.1.1.87 save_text()..... | 37 |
| 3.1.1.88 inputs_correct()..... | 38 |
| 3.1.1.89 chosen_activity_thresh(self, *args)..... | 38 |
| 3.1.1.90 reset_ecg()..... | 38 |
| 3.1.1.91 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)..... | 38 |
| 3.1.1.92 update_ecg_continuously(self)..... | 39 |
| 3.1.1.93 generate_ecg()..... | 39 |
| 3.1.1.94 stop_ecg_update(self)..... | 39 |
| 3.1.1.95 resume_ecg_update(self)..... | 40 |
| Global Variables:..... | 40 |
| Private Functions:..... | 40 |
| 3.1.1.96 __init__(self,*args,**kwargs)..... | 40 |
| Class AAIR():..... | 40 |
| Public Functions:..... | 41 |
| 3.1.1.97 load_saved_data(self)..... | 41 |
| 3.1.1.98 is_digit_check(P)..... | 41 |
| 3.1.1.99 save_text()..... | 41 |
| 3.1.1.100 inputs_correct()..... | 41 |
| 3.1.1.101 chosen_activity_thresh(self, *args)..... | 42 |
| 3.1.1.102 reset_ecg()..... | 42 |
| 3.1.1.103 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)..... | 42 |
| 3.1.1.104 update_ecg_continuously(self)..... | 43 |
| 3.1.1.105 generate_ecg()..... | 43 |
| 3.1.1.106 stop_ecg_update(self)..... | 43 |
| 3.1.1.107 resume_ecg_update(self)..... | 43 |
| Global Variables:..... | 44 |
| Private Functions:..... | 44 |
| 3.1.1.108 __init__(self,*args,**kwargs)..... | 44 |
| Class VVIR():..... | 44 |
| Public Functions:..... | 44 |
| 3.1.1.109 load_saved_data(self)..... | 44 |
| 3.1.1.110 is_digit_check(P)..... | 45 |
| 3.1.1.111 save_text()..... | 45 |
| 3.1.1.112 inputs_correct()..... | 45 |
| 3.1.1.113 chosen_activity_thresh(self, *args)..... | 46 |
| 3.1.1.114 reset_ecg()..... | 46 |
| 3.1.1.115 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)..... | 46 |

| | |
|---|-----------|
| 3.1.1.116 update_ecg_continuously(self)..... | 47 |
| 3.1.1.117 generate_ecg()..... | 47 |
| 3.1.1.118 stop_ecg_update(self)..... | 47 |
| 3.1.1.119 resume_ecg_update(self)..... | 47 |
| Global Variables:..... | 48 |
| Private Functions:..... | 48 |
| 3.1.1.120 __init__(self, parent, controller)..... | 48 |
| 4. Testing..... | 48 |
| 4.1 Running Program..... | 48 |
| 4.2 User Registration..... | 49 |
| 4.2.1 Test Cases:..... | 49 |
| 4.3 User Login | |
| 4.3.1 Test cases:..... | 52 |
| 4.4 Welcome Page..... | 53 |
| 4.5 AOO Operating Mode..... | 55 |
| 4.5.1 Test Cases:..... | 56 |
| 4.6 Other Operating Modes..... | 59 |

1. Requirements Changes

For our DCM design, we are required to develop a working GUI for the pacemaker user to interact with. The GUI includes a welcome screen, login page, register page, and functionality to access 8 different required operating modes along with their corresponding programmable parameters. The GUI is able to establish a serial connection with the pacemaker device in order to send and receive data. This data is utilized by each operating mode along with their respective programmable parameters to output atrial and ventricle egram data as a graph that updates in real time. Looking into the future, refining the aesthetic and visual look of the GUI, as well as ensuring we handle the serial transfer of data to and from the pacemaker device will be the main focus. The amount of operating modes as well as their respective programmable parameters are requirements that are likely to change. Some visual and user experience focused requirements may also be likely to be added, such as a minimum text size to reduce eye strain of the user.

We decided on using Python for coding the program that will allow direct communication with our Pacemaker. We also decided on using the Python library Tkinter for developing the interfaces of the program. The reason behind choosing Python was because of our previous experience with the language and how simple it is. Furthermore, Python is the easiest choice for serial communication with the microcontroller. Tkinter was chosen because of how common it is for creating graphical user interfaces in python. It is also quite simple and easy to learn. We decided on using the csv library to store data, such as different users and their last programmable parameter inputs as it was simple and easy to implement. We felt that having a dedicated database for such a small dataset (10 users max) was not worth the implementation time. We also decided on using the matplotlib library for the configuration of the graphs for the different parameters of each pacemaker mode. Along with matplotlib, we also used the NumPy library for the processing of arrays and data. Both of these libraries are used to process the parameter information and generate 2D plots using the data inputted through the user interface. Another python library we decided to import is the Shutil library which allows file operations and collections, this can be copying, removing, and creating data within files. This library helps us manage the different files for our module which hold important data regarding parameters and user information. We used the serial library to handle the connection and data transfer to and from the pacemaker device, as it is the most common and well documented library in Python for this purpose. The threading module in the code is used to run the ECG graph updates in a separate thread, allowing the GUI to remain responsive. The time module, particularly time.sleep(), is employed to manage the update intervals of the ECG graph and to enable pause functionality without freezing the GUI. Strings are used for serial data communication due to their ease of encoding and decoding, making them ideal for text-based data transmission. Arrays are chosen for efficiently handling and organizing multiple data elements, simplifying the process of sending and receiving structured data via serial.

2. Design Decision Changes

The use of one singular Python module makes the code hard to read and edit. Because of this, it is likely that going forward, we will switch to a modularized design, with multiple files split up accordingly. This lack of modularity is especially evident in the operating modes. Currently, each operating mode is its own class. It is likely that this will be changed to one operating mode parent class with each operating mode being a child of said parent. This will help with repeated logic and allow the efficient reuse of functions for each operating mode. The way the parameters are inputted for each mode is also something that may change in the future, as currently there are many repeated functions for each class. Currently, all of our functions are public and can be accessible by any class anywhere in the code. We may change this going forward and turn some of the public functions private if they are not being used elsewhere. In general, our code design is likely to be completely restructured to a simpler, more modular design that utilizes parent classes and inheritance more succinctly and efficiently.

3. Modules

Similar to Assignment 1, we have used one module to store all of our interface programming. This module holds all of our classes, functions, and variables involved in the logic and display of our user login and registration screens, welcome screen, basic user interfaces for accessing the pacemaker modes, and serial connection and data transfer. Our general code structure is through the use of classes as different screens or 'frames' using a parent class to store all of them. Each child class represents a different screen, and the functions and variables within the class help to create the graphical user interface and logic required for the interface to run smoothly.

3.1 DCM Modules

3.1.1 Main.py

As of assignment 1, Main.py contains the entire DCM design. It holds the programming for the main interface of the DCM. The interface allows the user to login as an existing user, register as a new user, and provides the user access to the Pacemaker information, such as the different operating modes and parameters. Considering there is no serial communication between the pacemaker and the DCM yet, the interface holds no valuable information. The DCM is programmed as a placeholder for the data that will be communicated by the Pacemaker by the end of assignment 2. For the purpose of documentation, we will treat each class, or 'frame', as a module and mention each public and private function, as well as global variables respective to the class.

Public Functions

- Because we are using python, every function implemented is technically a public function, as there is no existence of private functions that cannot be accessed unless inside of the class it is defined in, with the exception of the constructor method for each class. Currently, our design does not require the use of private functions, again, with the exception of constructors. As mentioned in ‘Design Decisions Changes that are Likely’, this will most likely change in the future.

3.1.1.1 open_serial_connection()

- This function takes no parameters. When called, this function tries to establish a serial connection to the defined communication port. In terms of blackbox behaviour, the function’s input is the communication port and the output is either an established serial connection with the pacemaker, or a printed error statement in the console (for debugging purposes). More specifically, the function utilizes the serial library to attempt a connection with the defined communication port at the defined bandwidth within a try block using the method `Serial()`. If successful, it returns `true`. If unsuccessful, a debug message is printed into the console and the function returns `false`.

3.1.1.2 send_data_via_serial(parameters)

- This function takes the parameter ‘parameters’, which holds the data we are trying to send to the pacemaker. The function takes an array of data as input, and tries to write the data to the pacemaker as output. If unsuccessful, the function will output an error message box, telling the user that the transfer was not successful. More specifically, the function takes the array of parameters and turns it into a string. It then encodes the data and tries to write it to the pacemaker using the methods `ser.write()` and `encode()`. If an error occurs, the function prints a debug message in the console and outputs an error popup box using the `showerror()` method to let the user know.

3.1.1.3 close_serial_connection()

- This function takes no parameters. When called, the function tries to end the connection between the DCM and the pacemaker. It takes the status of the connection as a boolean input and as output, closes the connection. If unsuccessful, an error message is printed to the console. More specifically, it uses a try-except block to check the status of the connection using the `ser.isOpen()` method and closes the connection using the `ser.close()` method, and prints a debug message otherwise.

3.1.1.4 recieve_data_via_serial()

- This function takes no parameters. The function takes data from the pacemaker as input and outputs the decoded data, or prints an error message if unsuccessful. More specifically, the function reads the first 16 bits from the pacemaker using the `ser.read()` method and strips and decodes the data before returning it, using the `decode()` and `strip()` methods. If a connection is not established, an error message is printed into the console.

3.1.1.5 get_curr_user()

- This function takes no parameters. When called, this function opens 'currentuser.csv', a csv file that we use to store the data of the user currently logged in, and returns that data as an array. This data includes the user's username, password, and all operating parameters.

3.1.1.6 delete_user(username)

- This function takes the parameter 'username', which represents the username of the current user. This function is responsible for deleting the profile that matches the passed in 'username'. In terms of blackbox behaviour, the function takes 'username' as input and updates the users.csv file. More specifically, when called, the function opens a temp file and the users.csv file and iterates through the first column (where the usernames are stored) of each row. If the username does not match the parameter 'username', it rewrites the entire row into the tempfile using the `writerow()` method. It then replaces the users.csv file with the temp file by using the `shutil` library and the `move()` method.

Global Variables:

FONT: The variable 'FONT' stores the font used for each widget on every frame.

SER: The variable `ser` is used to manage a serial port connection in the program, enabling communication with the pacemaker through serial communication.

Class Window():

This class serves as the parent class for all subsequent classes that act as new frames.

Public Functions:

3.1.1.7 show_frame(self,cont)

- This function takes the parameter 'cont', which is the name of a screen or 'frame'. The function is responsible for changing the interface to display the appropriate

screen/'frame'. This function accepts one parameter as input as discussed earlier, and the output of the function is the display of the frame name that was passed in. Internally, the function accesses the dictionary "self.frames" which holds every frame written in the program with a corresponding name in a key : value pair. The corresponding frame is passed through "self.frames" and assigned to the variable "frame". The method "tkraise" is then used to display the frame assigned to the variable "frame". The method event_generate() is used to generate an event every time this function is used, which will be useful later on in the code to update any data values.

3.1.1.8 reset_entry(self,entry,txt)

- The function accepts two parameters: "entry" and "txt". The parameter "entry" represents the entry widget that will be reset and the parameter "txt" which is the text value that will be placed in the entry widget after resetting. The purpose of this function is to reset the content in an entry widget. In terms of blackbox behaviour, the function's input includes an entry widget and a string and the output is the reset entry widget with the string stored in the 'txt' parameter. More specifically, the function first clears all of the contents of the "entry" widget by calling on the "entry.delete" method. Then the function inserts a specified text, 'txt' into the widget by calling on the "entry.insert" method.

Global Variables:

container: The variable 'container' holds the tkinter widget tk.Frame().

self.frames: The variable 'self.frames' is a dictionary used to hold all child classes of the class Window, which is every frame for the DCM design.

Private Functions:

3.1.1.9 __init__(self,*args,**kwargs)

- This is the constructor for the class Window. The constructor sets up the container used to hold each frame, or child class, and calls the show_frame() function with 'Login' as the parameter to start the program.

Class Login(tk.Frame):

This class serves as the login frame of the DCM design.

Public Functions:

3.1.1.10 user_on_focusin(event)

- This function takes the parameter 'event', which is an event that happens within the design based on user action. The function takes the event of focusing in on the usernameLog entry widget as input and clears the text in the widget as output. Specifically, the function deletes the text stored in the usernameLog entry widget using the widget.delete() method.

3.1.1.11 user_on_focusout(event)

- This function takes the parameter 'event', which is an event that happens within the design based on user action. The function takes the event of focusing out of the usernameLog entry widget as input and clears the text in the widget as output. Specifically, the function reads the usernameLog entry widget and, if empty, inserts the default text "Username" back into the widget using the widget.insert() method.

3.1.1.12 pass_on_focusin(event)

- This function is the exact same as 'user_on_focusin(event)', but takes the input as focusing in on the passwordLog entry widget. Specifically, the function deletes the stored text in the passwordLog entry widget using the widget.delete() method. The function then uses widget.config() to hide the text with '*'.

3.1.1.13 pass_on_focusout(event)

- This function is the exact same as 'user_on_focusout(event)', but takes the input as focusing out on the passwordLog entry widget. Specifically, the function reads the text stored in the passwordLog entry widget and, if empty, inserts the default text "Password" using the widget.insert() method. The function then uses widget.config() to unhide the text.

3.1.1.14 store_user(user)

- This function takes 'user' as a parameter. The function takes the 'user' parameter, an array, as input and writes the information into the CSV file 'currentuser.csv' as output. Specifically, by utilizing the csv library, this function opens the file in write mode and uses the csv method csv.writerow() in order to write the data into the file.

3.1.1.15 readUser()

- This function has no parameters. The function takes the user inputted username and password from the respective entry widgets as inputs and either updates the frame and

clears all widgets or gives an error message displayed in a widget depending on if the username or password are correct. More specifically, the function reads the user's entry widget inputs for the usernameLog and passwordLog widgets and assigns it to the variables "username" and "password". The function then opens the csv file 'users.csv' in read mode. For each row in the file, if at the 0th and 1st index is the username and password that is stored in the respective variables "username" and "password", it calls function store_user(row), using the current row as the parameter for the function. The function then clears the error message widget using widget.config() and calls the reset_entry() functions for the usernameLog and passwordLog entry widgets used. Finally, the function show_frame() is called to update the screen. If the inputs do not match, an error message is displayed in the respective widget using the method entry.config() and editing the text in said widget.

Global Variables:

label: The variable 'label' is a tkinter label widget that has the text "Login" displayed.

usernameLog: The variable 'usernameLog' is an entry widget used to collect the user's input. It has the default text "Username" inside and is binded to the events '<FocusIn>' and '<FocusOut>', which are used for functions 'user_on_focusin()' and 'user_on_focusout()'.

passwordLog: The variable 'passwordLog' is an entry widget used to collect the user's input. It has the default text "Password" inside and is binded to the events '<FocusIn>' and '<FocusOut>', which are used for functions 'pass_on_focusin()' and 'pass_on_focusout()'.

errmsg: The variable 'errmsg' is a label widget that has no default text. It is used to let the user know if they have incorrect inputs.

loginB: The variable 'loginB' is a button widget with the text "Login". When pressed, it calls the function 'readUser()'.

RegisterB: The variable 'RegisterB' is a button widget with the text "Register an Account". When pressed, it calls 'show_frame()' using the Register frame as a parameter, and then clears the usernameLog, passwordLog, and errmsg widgets using the 'reset_entry()' function and widget.config() method respectively.

Private Functions:

*3.1.1.16 __init__(self,*args,**kwargs)*

- This is the constructor for the class Login. The constructor sets up the container used to hold each frame, or child class, and calls the show_frame() function with 'Login' as the parameter to start the program.

Class Register():

This class serves as a register frame for users to register accounts

Public Functions:

3.1.1.17 on_focusin(event)

- This function takes the parameter 'event', which is an event that happens within the design based on user action. The function takes the event of focusing in on the usernameLog, passwordLog, or confirmLog entry widget as input and clears the text in the widget as output. Specifically, the function deletes the text stored in the username entry widget using the widget.delete() method.

3.1.1.18 on_focusout(event)

- This function takes the parameter 'event', which is an event that happens within the design based on user action. The function takes the event of focusing out of the usernameLog, passwordLog, or confirmLog entry widget as input and clears the text in the widget as output. Specifically, the function reads the entry widget and, if empty, inserts the default text 'Username' or 'Password' back into the widget using the widget.insert() method. If the passwordLog or confirmLog instance is called, it also unhides the entry by using widget.config().

3.1.1.19 usernameExists(username)

- This function takes the parameter 'username', which is a string holding the inputted username. This function checks whether the inputted username already exists within the 'users.csv' file. The function takes this parameter as input and as output, returns True if the username exists in 'users.csv' and False otherwise. Specifically, the function opens 'users.csv' in read mode and loops through each row in the csv file, comparing the 0th index in the row to the username parameter. If they are equal, the function returns True. If the function exits the loop, meaning that there were no matching usernames within the file, the function returns False.

3.1.1.20 *numRegistered()*

- This function takes no parameters. The function is used to determine the number of users that are registered in the CSV file “users.csv” which holds each registered user on a separate line. In terms of the blackbox behaviour, the input would be the contents in the ‘users.csv’ file and the output would be the number of rows the file contains, or the number of users registered in the DCM. Specifically, the function opens the file “users.csv” in read mode and, using the `sum()` method, counts each row in the file and stores the number of lines in the variable ‘user_count’. Once the function is done reading the file, it returns ‘user_count’.

3.1.1.21 *writeUser()*

- This function takes no parameters. This function is used to write the user inputs into the ‘users.csv’ file and thereby create a new user in the DCM. The function takes inputs from the `usernameLog` and `passwordLog` and it outputs a new line in the ‘users.csv’ file. More specifically, the function stores the user inputs from `usernameLog` and `passwordLog` widgets into the variables ‘username’ and ‘password’. It then formats the data into an array in the order of username, password, and then blank entries meant to be placeholders for the operating mode parameters. It stores this array in the variable ‘write_data’. The function then opens the ‘users.csv’ in append mode and, using the `writerow()` method, writes the contents of ‘write_data’ into the file.

3.1.1.22 *registerCheck()*

- This function takes no parameters. This function is used for the user’s registration process. That is, ensuring the information entered into the entry widgets is valid for creating a new user. In terms of blackbox behaviour, the function takes inputs from the `usernameLog`, `passwordLog`, and `confirmLog` entry widgets and outputs a popup message indicating the registration was successful or reveals an error message if the inputs were incorrect. Specifically, the function stores the inputs from `usernameLog`, `passwordLog`, and `confirmLog` into the ‘username’, ‘password’, and ‘confirmP’ variables, respectively. The function then checks if the username variable holds any spaces. If so, it uses the `widget.config()` method to edit the `errmsg` widget and display the text “Username cannot contain spaces.”. If not, it then checks if the username variable holds the text “Username”. If so, it changes the `errmsg` widget using the same method to display the text “Please input a valid username.”. If not, it then checks if the password variable holds the values “Password”, “Confirm Password”, or if it contains a space. If so, it updates the `errmsg` widget to display

the text “Please input a valid password”. If these conditions are also false, the function continues to check if the total number of registered users is greater than or equal to 10 using the function numRegistered(). If so, the program does not allow the user to register and outputs the error message “Registration is closed. Maximum users reached.”. If not, the program continues to the next elif statement which checks if the confirmP variable equals the password variable. This checks that the values entered in both the “password” and “confirm password” entry widgets are the same. If they are not the same, an error message is displayed stating “Passwords do not match.”. Then the function processes its last elif statement which checks if the username entered already exists in the CSV file with the registered users, using the function UsernameExists(username). If the username already exists, another error message is displayed which states “Username already exists. Please use a different Username.”. Lastly, if the user’s inputted username and password are valid after going through the series of if-statements, then the function will call the writeUser() function to register the user and append the username and password into the CSV file “users.csv”. It then uses the showinfo() method to display a pop up message box that lets the user know they have successfully registered. The function then resets all the widgets using the reset_entry() function and finally updates the frame to the Login frame by calling the show_frame() function.

Global Variables:

usernameLog: The variable ‘usernameLog’ is an entry widget used to collect the user’s input. It has the default text “Username” inside and is binded to the events ‘<FocusIn>’ and ‘<FocusOut>’, which are used for functions ‘on_focusin()’ and ‘on_focusout()’.

passwordLog: The variable ‘passwordLog’ is an entry widget used to collect the user’s input. It has the default text “Password” inside and is binded to the events ‘<FocusIn>’ and ‘<FocusOut>’, which are used for functions ‘on_focusin()’ and ‘on_focusout()’.

confirmLog: The variable ‘confirmLog’ is an entry widget used to collect the user’s input. It has the default text “Confirm Password” inside and is binded to the events ‘<FocusIn>’ and ‘<FocusOut>’, which are used for functions ‘on_focusin()’ and ‘on_focusout()’.

errmsg: The variable ‘errmsg’ is a label widget that has no default text. It is used to let the user know if they have incorrect inputs.

RegisterB: The variable 'RegisterB' is a button widget with the text "Register". When pressed, it calls the 'registerCheck()' function as well as the focus_set() method.

LoginB: The variable 'loginB' is a button widget with the text "Back to Login". When pressed, it calls the function 'show_frame' with the frame Login as a parameter. It also calls the functions 'reset_entry' on usernameLog, passwordLog, and confirmLog. It then calls widget.config to unhide the text in passwordLog and confirmLog and to empty the text in errormsg. It then calls the method focus_set().

Private Functions:

*3.1.1.23 __init__(self,*args,**kwargs)*

- This is the constructor for the class Register. The constructor sets up the container used to hold each frame, or child class, and calls the show_frame() function with 'Login' as the parameter to start the program.

Class Front():

This class serves as a front page where the user is able to select operating modes

Public Functions:

*3.1.1.24 updateMode(*args)*

- The purpose of this function is to update the interface of the DCM depending on the pacing mode chosen by the user. The function considers a drop-down menu with the four pacing modes focused on in assignment 1. The function has one parameter for input which represents the value of the selected pacemaker mode. In terms of blackbox behaviour, the input of the function relates to the one parameter discussed earlier, which is the selected pacemaker mode by the user, and the function does not return any particular value as output but it does change the interface based on the chosen pacemaker mode. Internally, the function works in a simple manner: there are 4 if-statements that check which pacemaker mode is selected. Based on the selected pacemaker mode, the program displays the corresponding frame and its parameters.

3.1.1.25 displayConnection()

- This simple function is responsible for displaying a message onto the frame that lets the user know the device is connected. There are no input parameters since the

function only runs one line of code. In terms of blackbox behaviour, there is no input to the function but the output is simple a displayed message that reads “device connected”

3.1.1.26 sameDevice()

- Similar to the previous function, this function is also responsible for printing out a message to let the user know the device that is connected is the same device previously connected. There are no inputs to the function since it runs one line of code. The blackbox behaviour is the same as the displayConnection() function; there are no inputs and the output is a simple message to let the user know that the device connected to the DCM is the same.

3.1.1.27 on_show_frame(self,event)

- This function simply prints out a message at the top of the welcome frame that states “Welcome, {username}”. There are no input parameters in this function considering it only contains one line of code, which is the CSV configuration. In terms of blackbox behaviour, there are no input parameters but the code uses the variable assigned to the username to output the welcome message.

Global Variables:

welcome: This variable holds the reference to a label widget which is created inside of the class and displays a welcome message personalised for the DCM user. As a global variable in this class, it can be accessed by any method within the class Front().

Private Functions:

*3.1.1.28 __init__(self,*args,**kwargs)*

- This is the constructor for the class Front. The constructor sets up the container used to hold each frame, or child class, and calls the show_frame() function with ‘Login’ as the parameter to start the program.

Class AOO():

This class serves as the AOO operating mode page where users may input numerical values and display graphs and save data for the user

Public Functions:

3.1.1.29 load_saved_data(self)

- This function is used to load the saved data of the current user. It uses the `get_curr_user()` function and sets the filename to “users.csv”. This function opens the csv file and searches for a row where the first element matches `row[0]`, the value of the current users. If the row is found, it will extract the data from the four values. This function throws an error if a `ValueError` is found and lets the user know that if the data is from a different mode, they must input data from the current operating mode. This function also has labels that allow the data values to be displayed to let the user know which values were last inputted into the system. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOO, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.30 is_digit_check(P)

- This function checks if the input is a valid number. It will return true if the characters only contain numbers or if the string is empty, and it returns false if P contains any other characters. The black box parameters for this function is the input P which takes any numerical value, and the output is a boolean value that either returns true or false depending on if it is a number or not respectively.

3.1.1.31 save_text()

- This function first checks if the inputs are valid by calling the function `inputs_correct()`. If the inputs are valid, then it will write the data into the data variable which holds the parameters for the specific AOO mode. The function then opens the users.csv file, and creates a new temporary file that will take any of the changes and write them into the new temp file created. Then it replaces the original file with the temporary file. An array called `curr_data` is created which includes the username, password, and the updated values. It writes this data into the `currentusers.csv` file. We then call the `generate_ecg()` function to create a graph. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOO, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.32 inputs_correct()

- This function checks for if the inputs are correct. It checks for all of the different limitations based on the Pacemaker document on page 34. This function is called in the above function `save_text()`. This function is used differently in the 8 classes AOO,

AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes to the limitation of the parameters set in the Pacemaker document

3.1.1.33 reset_ecg()

- The purpose of this function is to reset the ECG simulation, stopping any graph updates, clears the signal data, clears the graph if the window is open, and resets the time counter. There are no particular parameters for this function besides “self”, which is a defining method in a python function. This function works by first checking if the graph is running and if it is, it calls on the function “self.stop_ecg_update()” which stops the ECG update loop. The function clears the ECG signal data by initializing an empty array to the data array and resets the time counter to zero. Lastly, the function clears the graph. In terms of the blackbox behaviour, there are no particular inputs as there are no parameters, but the function takes in the ECG graph and outputs a cleared graph.

3.1.1.34 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)

- The purpose of this function is to simulate an ECG waveform based on the data taken in by the parameters. The parameters are duration, lrl, url, atrial_amp, and atrial_pulse_width. The function works by first setting the time step to the time vector “dt”. It then initializes the ECG signal and calculates the beat duration at LRL and URL. It then creates the ECG waveform by running a while loop until time_lapsed exceeds the specified duration. The beat iterates between LRL and URL. Then the function calculates the P-wave, QRS complex, and T-wave. The function creates the gaussian components for the ECG signal for P-wave, QRS complex, and T-wave and adds them to the ECG signal. Lastly, the time lapse variable is updated and the beat duration is added. In terms of the blackbox behaviour, the function takes in duration, lrl, url, atrial_amp, and atrial_pulse_width as inputs, and returns an ECG graph and the time vector “t”.

3.1.1.35 update_ecg_continuously(self)

- The purpose behind this function is to create and update an ECG signal with the option of pausing the signal as well. There are no particular input parameters besides “self” which is a defining method in a python function. This function works by first initializing a few variables: “segment_duration”, “self.running”, and “self.paused”.

When `self.running` is true, a loop continuously runs and if `self.paused` is true, the loop is paused for 1 second then continues the iteration of the loop. In the loop, a try-block is executed where the stored variables are fetched for the AOO mode and the function “`generate_ecg_waveform`” is called to generate a new segment of the ecg waveform. If there is no ECG waveform present then the function is returned. The new segment fetched is then concatenated to the current ECG signal data and the time is then incremented accordingly. The current ECG data is then cleared from the graph and the new data is plotted. The function then sleeps based on the segment duration. If an error occurs in the try block then the function will sleep for 1 second and continue to the next iteration in the while loop. In terms of the blackbox behaviour, there are no particular inputs to the function besides the stored variables of the class and outputs an updated ECG waveform.

3.1.1.36 generate_ecg()

- This function creates a(gcf) graph based upon the inputted data from parameters for the set operating mode. This function uses the matplotlib library and the numpy library.

3.1.1.37 stop_ecg_update(self)

- The purpose of this function is to pause or stop the ECG update loop and therefore stop the ECG graph’s constant updates. There are no input parameters to this function besides “`self`” which is a defining method in a python function. More specifically, this function works by setting “`self.paused`” to True, disables the “stop” button, and enables the “resume” button. In terms of blackbox behaviour, the function does not take in any parameters, it only takes in the updated ECG graph and outputs a paused ECG graph.

3.1.1.38 resume_ecg_update(self)

- The purpose of this function is to resume the ECG graph updates after the user stops the graph. There are no input parameters besides “`self`” which is a defining method in a python function. The function works by first checking if the graph is not running, if it is not running it sets “`self.running`” to True and begins a new thread “`self.ecg_update_thread`”. The function then sets “`self.paused`” to False so the program knows the ECG updates should resume, and then it disables the “resume” button and enables the “stop” button. In terms of blackbox behaviour, the function does not have any input parameters, but rather the input is a paused graph, and the output is an updating ECG graph.

Global Variables:

Global variables for each mode include the label and entry text box for each of their respective programmable parameters listed in the pacemaker document. Each entry box is input guarded by using the `is_digit_check()` method to ensure only numbers are inputted. The buttons that generate and control the ECG waveform as well as the dataflow are also included as global variables.

Private Functions:

*3.1.1.39 `__init__(self,*args,**kwargs)`*

- This is the constructor for the class AOO. The constructor sets up the container used to hold each frame, or child class, and calls the `show_frame()` function with 'Login' as the parameter to start the program.

Class AAI():

This class serves as the AAI operating mode page where users may input numerical values and display graphs and save data for the user

Public Functions:

3.1.1.40 `load_saved_data(self)`

- This function is used to load the saved data of the current user. It uses the `get_curr_user()` function and sets the filename to "users.csv". This function opens the csv file and searches for a row where the first element matches `row[0]`, the value of the current users. If the row is found, it will extract the data from the nine values. This function throws an error if a `ValueError` is found and lets the user know that if the data is from a different mode, they must input data from the current operating mode. This function also has labels that allow the data values to be displayed to let the user know which values were last inputted into the system. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOO, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.41 `is_digit_check(P)`

- This function checks if the input is a valid number. It will return true if the characters only contain numbers or if the string is empty, and it returns false if P contains any other characters. The black box parameters for this function is the input P which takes

any numerical value, and the output is a boolean value that either returns true or false depending on if it is a number or not respectively.

3.1.1.42 save_text()

- This function first checks if the inputs are valid by calling the function `inputs_correct()`. If the inputs are valid, then it will write the data into the data variable which holds the parameters for the specific AAI mode. The function then opens the `users.csv` file, and creates a new temporary file that will take any of the changes and write them into the new temp file created. Then it replaces the original file with the temporary file. An array called `curr_data` is created which includes the username, password, and the updated values. It writes this data into the `currentusers.csv` file. We then call the `generate_ecg()` function to create a graph. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.43 inputs_correct()

- This function checks for if the inputs are correct. It checks for all of the different limitations based on the Pacemaker document on page 34. This function is called in the above function `save_text()`. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes to the limitation of the parameters set in the Pacemaker document

3.1.1.44 reset_ecg()

- The purpose of this function is to reset the ECG simulation, stopping any graph updates, clears the signal data, clears the graph if the window is open, and resets the time counter. There are no particular parameters for this function besides “self”, which is a defining method in a python function. This function works by first checking if the graph is running and if it is, it calls on the function “`self.stop_ecg_update()`” which stops the ECG update loop. The function clears the ECG signal data by initializing an empty array to the data array and resets the time counter to zero. Lastly, the function clears the graph. In terms of the blackbox behaviour, there are no particular inputs as there are no parameters, but the function takes in the ECG graph and outputs a cleared graph.

3.1.1.45 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)

- The purpose of this function is to simulate an ECG waveform based on the data taken in by the parameters. The parameters are duration, lrl, url, atrial_amp, and atrial_pulse_width. The function works by first setting the time step to the time vector “dt”. It then initializes the ECG signal and calculates the beat duration at LRL and URL. It then creates the ECG waveform by running a while loop until time_lapsed exceeds the specified duration. The beat iterates between LRL and URL. Then the function calculates the P-wave, QRS complex, and T-wave. The function creates the gaussian components for the ECG signal for P-wave, QRS complex, and T-wave and adds them to the ECG signal. Lastly, the time lapse variable is updated and the beat duration is added. In terms of the blackbox behaviour, the function takes in duration, lrl, url, atrial_amp, and atrial_pulse_width as inputs, and returns an ECG graph and the time vector “t”.

3.1.1.46 update_ecg_continuously(self)

- The purpose behind this function is to create and update an ECG signal with the option of pausing the signal as well. There are no particular input parameters besides “self” which is a defining method in a python function. This function works by first initializing a few variables: “segment_duration”, “self.running”, and “self.paused”. When self.running is true, a loop continuously runs and if self.paused is true, the loop is paused for 1 second then continues the iteration of the loop. In the loop, a try-block is executed where the stored variables are fetched for the AOO mode and the function “generate_ecg_waveform” is called to generate a new segment of the ecg waveform. If there is no ECG waveform present then the function is returned. The new segment fetched is then concatenated to the current ECG signal data and the time is then incremented accordingly. The current ECG data is then cleared from the graph and the new data is plotted. The function then sleeps based on the segment duration. If an error occurs in the try block then the function will sleep for 1 second and continue to the next iteration in the while loop. In terms of the blackbox behaviour, there are no particular inputs to the function besides the stored variables of the class and outputs an updated ECG waveform.

3.1.1.47 generate_ecg()

- This function creates a(gcf) graph based upon the inputted data from parameters for the set operating mode. This function uses the matplotlib library and the numpy library.

3.1.1.48 stop_ecg_update(self)

- The purpose of this function is to pause or stop the ECG update loop and therefore stop the ECG graph's constant updates. There are no input parameters to this function besides "self" which is a defining method in a python function. More specifically, this function works by setting "self.paused" to True, disables the "stop" button, and enables the "resume" button. In terms of blackbox behaviour, the function does not take in any parameters, it only takes in the updated ECG graph and outputs a paused ECG graph.

3.1.1.49 resume_ecg_update(self)

- The purpose of this function is to resume the ECG graph updates after the user stops the graph. There are no input parameters besides "self" which is a defining method in a python function. The function works by first checking if the graph is not running, if it is not running it sets "self.running" to True and begins a new thread "self.ecg_update_thread". The function then sets "self.paused" to False so the program knows the ECG updates should resume, and then it disables the "resume" button and enables the "stop" button. In terms of blackbox behaviour, the function does not have any input parameters, but rather the input is a paused graph, and the output is an updating ECG graph.

Global Variables:

Global variables for each mode include the label and entry text box for each of their respective programmable parameters listed in the pacemaker document. Each entry box is input guarded by using the `is_digit_check()` method to ensure only numbers are inputted. The buttons that generate and control the ECG waveform as well as the dataflow are also included as global variables.

Private Functions:

*3.1.1.50 __init__(self, *args, **kwargs)*

- This is the constructor for the class AAI. The constructor sets up the container used to hold each frame, or child class, and calls the `show_frame()` function with 'Login' as the parameter to start the program.

Class VOO():

This class serves as the VOO operating mode page where users may input numerical values and display graphs and save data for the user

Public Functions:

3.1.1.51 load_saved_data(self)

- This function is used to load the saved data of the current user. This function has no relevant input parameters. It uses the `get_curr_user()` function and sets the filename to “users.csv”. This function opens the csv file and searches for a row where the first element matches `row[0]`, the value of the current users. If the row is found, it will extract the data from the four values. This function throws an error if a `ValueError` is found and lets the user know that if the data is from a different mode, they must input data from the current operating mode. This function also has labels that allow the data values to be displayed to let the user know which values were last inputted into the system. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.52 is_digit_check(P)

- This function checks if the input is a valid number. It will return true if the characters only contain numbers or if the string is empty, and it returns false if P contains any other characters. The black box parameters for this function is the input P which takes any numerical value, and the output is a boolean value that either returns true or false depending on if it is a number or not respectively.

3.1.1.53 save_text()

- This function first checks if the inputs are valid by calling the function `inputs_correct()`. If the inputs are valid, then it will write the data into the data variable which holds the parameters for the specific VOO mode. The function then opens the users.csv file, and creates a new temporary file that will take any of the changes and write them into the new temp file created. Then it replaces the original file with the temporary file. An array called `curr_data` is created which includes the username, password, and the updated values. It writes this data into the `currentusers.csv` file. We then call the `generate_ecg()` function to create a graph. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.54 inputs_correct()

- This function checks for if the inputs are correct. It checks for all of the different limitations based on the Pacemaker document on page 34. This function is called in the above function `save_text()`. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes to the limitation of the parameters set in the Pacemaker document

3.1.1.55 reset_ecg()

- The purpose of this function is to reset the ECG simulation, stopping any graph updates, clears the signal data, clears the graph if the window is open, and resets the time counter. There are no particular parameters for this function besides “self”, which is a defining method in a python function. This function works by first checking if the graph is running and if it is, it calls on the function “self.stop_ecg_update()” which stops the ECG update loop. The function clears the ECG signal data by initializing an empty array to the data array and resets the time counter to zero. Lastly, the function clears the graph. In terms of the blackbox behaviour, there are no particular inputs as there are no parameters, but the function takes in the ECG graph and outputs a cleared graph.

3.1.1.56 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)

- The purpose of this function is to simulate an ECG waveform based on the data taken in by the parameters. The parameters are duration, lrl, url, atrial_amp, and atrial_pulse_width. The function works by first setting the time step to the time vector “dt”. It then initializes the ECG signal and calculates the beat duration at LRL and URL. It then creates the ECG waveform by running a while loop until time_lapsed exceeds the specified duration. The beat iterates between LRL and URL. Then the function calculates the P-wave, QRS complex, and T-wave. The function creates the gaussian components for the ECG signal for P-wave, QRS complex, and T-wave and adds them to the ECG signal. Lastly, the time lapse variable is updated and the beat duration is added. In terms of the blackbox behaviour, the function takes in duration, lrl, url, atrial_amp, and atrial_pulse_width as inputs, and returns an ECG graph and the time vector “t”.

3.1.1.57 update_ecg_continuously(self)

- The purpose behind this function is to create and update an ECG signal with the option of pausing the signal as well. There are no particular input parameters besides “self” which is a defining method in a python function. This function works by first initializing a few variables: “segment_duration”, “self.running”, and “self.paused”. When self.running is true, a loop continuously runs and if self.paused is true, the loop is paused for 1 second then continues the iteration of the loop. In the loop, a try-block is executed where the stored variables are fetched for the AOO mode and the function “generate_ecg_waveform” is called to generate a new segment of the ecg waveform. If there is no ECG waveform present then the function is returned. The new segment fetched is then concatenated to the current ECG signal data and the time is then incremented accordingly. The current ECG data is then cleared from the graph and the new data is plotted. The function then sleeps based on the segment duration. If an error occurs in the try block then the function will sleep for 1 second and continue to the next iteration in the while loop. In terms of the blackbox behaviour, there are no particular inputs to the function besides the stored variables of the class and outputs an updated ECG waveform.

3.1.1.58 generate_ecg()

- This function creates a(gcf) graph based upon the inputted data from parameters for the set operating mode. This function uses the matplotlib library and the numpy library.

3.1.1.59 stop_ecg_update(self)

- The purpose of this function is to pause or stop the ECG update loop and therefore stop the ECG graph’s constant updates. There are no input parameters to this function besides “self” which is a defining method in a python function. More specifically, this function works by setting “self.paused” to True, disables the “stop” button, and enables the “resume” button. In terms of blackbox behaviour, the function does not take in any parameters, it only takes in the updated ECG graph and outputs a paused ECG graph.

3.1.1.60 resume_ecg_update(self)

- The purpose of this function is to resume the ECG graph updates after the user stops the graph. There are no input parameters besides “self” which is a defining method in a python function. The function works by first checking if the graph is not running, if it is not running it sets “self.running” to True and begins a new thread

“self.ecg_update_thread”. The function then sets “self.paused” to False so the program knows the ECG updates should resume, and then it disables the “resume” button and enables the “stop” button. In terms of blackbox behaviour, the function does not have any input parameters, but rather the input is a paused graph, and the output is an updating ECG graph.

Global Variables:

Global variables for each mode include the label and entry text box for each of their respective programmable parameters listed in the pacemaker document. Each entry box is input guarded by using the `is_digit_check()` method to ensure only numbers are inputted. The buttons that generate and control the ECG waveform as well as the dataflow are also included as global variables.

Private Functions:

*3.1.1.61 __init__(self,*args,**kwargs)*

- This is the constructor for the class VOO. The constructor sets up the container used to hold each frame, or child class, and calls the `show_frame()` function with ‘Login’ as the parameter to start the program.

Class VVI():

This class serves as the VVI operating mode page where users may input numerical values and display graphs and save data for the user

Public Functions:

3.1.1.62 load_saved_data(self)

- This function is used to load the saved data of the current user. It uses the `get_curr_user()` function and sets the filename to “users.csv”. This function opens the csv file and searches for a row where the first element matches `row[0]`, the value of the current users. If the row is found, it will extract the data from the eight values. This function throws an error if a `ValueError` is found and lets the user know that if the data is from a different mode, they must input data from the current operating mode. This function also has labels that allow the data values to be displayed to let the user know which values were last inputted into the system. This function is used differently in the 8 classes AOO, AAI,VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.63 is_digit_check(P)

- This function checks if the input is a valid number. It will return true if the characters only contain numbers or if the string is empty, and it returns false if P contains any other characters. The black box parameters for this function is the input P which takes any numerical value, and the output is a boolean value that either returns true or false depending on if it is a number or not respectively.

3.1.1.64 save_text()

- This function first checks if the inputs are valid by calling the function `inputs_correct()`. If the inputs are valid, then it will write the data into the data variable which holds the parameters for the specific VVI mode. The function then opens the `users.csv` file, and creates a new temporary file that will take any of the changes and write them into the new temp file created. Then it replaces the original file with the temporary file. An array called `curr_data` is created which includes the username, password, and the updated values. It writes this data into the `currentusers.csv` file. We then call the `generate_ecg()` function to create a graph. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.65 inputs_correct()

- This function checks for if the inputs are correct. It checks for all of the different limitations based on the Pacemaker document on page 34. This function is called in the above function `save_text()`. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes to the limitation of the parameters set in the Pacemaker document

3.1.1.66 reset_ecg()

- The purpose of this function is to reset the ECG simulation, stopping any graph updates, clears the signal data, clears the graph if the window is open, and resets the time counter. There are no particular parameters for this function besides “self”, which is a defining method in a python function. This function works by first checking if the graph is running and if it is, it calls on the function “`self.stop_ecg_update()`” which stops the ECG update loop. The function clears the ECG signal data by initializing an empty array to the data array and resets the time counter to zero. Lastly, the function clears the graph. In terms of the blackbox

behaviour, there are no particular inputs as there are no parameters, but the function takes in the ECG graph and outputs a cleared graph.

3.1.1.67 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)

- The purpose of this function is to simulate an ECG waveform based on the data taken in by the parameters. The parameters are duration, lrl, url, atrial_amp, and atrial_pulse_width. The function works by first setting the time step to the time vector “dt”. It then initializes the ECG signal and calculates the beat duration at LRL and URL. It then creates the ECG waveform by running a while loop until time_lapsed exceeds the specified duration. The beat iterates between LRL and URL. Then the function calculates the P-wave, QRS complex, and T-wave. The function creates the gaussian components for the ECG signal for P-wave, QRS complex, and T-wave and adds them to the ECG signal. Lastly, the time lapse variable is updated and the beat duration is added. In terms of the blackbox behaviour, the function takes in duration, lrl, url, atrial_amp, and atrial_pulse_width as inputs, and returns an ECG graph and the time vector “t”.

3.1.1.68 update_ecg_continuously(self)

- The purpose behind this function is to create and update an ECG signal with the option of pausing the signal as well. There are no particular input parameters besides “self” which is a defining method in a python function. This function works by first initializing a few variables: “segment_duration”, “self.running”, and “self.paused”. When self.running is true, a loop continuously runs and if self.paused is true, the loop is paused for 1 second then continues the iteration of the loop. In the loop, a try-block is executed where the stored variables are fetched for the AOO mode and the function “generate_ecg_waveform” is called to generate a new segment of the ecg waveform. If there is no ECG waveform present then the function is returned. The new segment fetched is then concatenated to the current ECG signal data and the time is then incremented accordingly. The current ECG data is then cleared from the graph and the new data is plotted. The function then sleeps based on the segment duration. If an error occurs in the try block then the function will sleep for 1 second and continue to the next iteration in the while loop. In terms of the blackbox behaviour, there are no particular inputs to the function besides the stored variables of the class and outputs an updated ECG waveform.

3.1.1.69 generate_ecg()

- This function creates a(gcf) graph based upon the inputted data from parameters for the set operating mode. This function uses the matplotlib library and the numpy library.

3.1.1.70 stop_ecg_update(self)

- The purpose of this function is to pause or stop the ECG update loop and therefore stop the ECG graph's constant updates. There are no input parameters to this function besides "self" which is a defining method in a python function. More specifically, this function works by setting "self.paused" to True, disables the "stop" button, and enables the "resume" button. In terms of blackbox behaviour, the function does not take in any parameters, it only takes in the updated ECG graph and outputs a paused ECG graph.

3.1.1.71 resume_ecg_update(self)

- The purpose of this function is to resume the ECG graph updates after the user stops the graph. There are no input parameters besides "self" which is a defining method in a python function. The function works by first checking if the graph is not running, if it is not running it sets "self.running" to True and begins a new thread "self.ecg_update_thread". The function then sets "self.paused" to False so the program knows the ECG updates should resume, and then it disables the "resume" button and enables the "stop" button. In terms of blackbox behaviour, the function does not have any input parameters, but rather the input is a paused graph, and the output is an updating ECG graph.

Global Variables:

Global variables for each mode include the label and entry text box for each of their respective programmable parameters listed in the pacemaker document. Each entry box is input guarded by using the `is_digit_check()` method to ensure only numbers are inputted. The buttons that generate and control the ECG waveform as well as the dataflow are also included as global variables.

Private Functions:

*3.1.1.72 __init__(self,*args,**kwargs)*

- This is the constructor for the class VVI. The constructor sets up the container used to hold each frame, or child class, and calls the show_frame() function with 'Login' as the parameter to start the program.

Class AOOR():

This class serves as the AOOR operating mode page where users may input numerical values and display graphs and save data for the user

Public Functions:

3.1.1.73 load_saved_data(self)

- This function is used to load the saved data of the current user. It uses the get_curr_user() function and sets the filename to "users.csv". This function opens the csv file and searches for a row where the first element matches row[0], the value of the current users. If the row is found, it will extract the data from the nine values. This function throws an error if a ValueError is found and lets the user know that if the data is from a different mode, they must input data from the current operating mode. This function also has labels that allow the data values to be displayed to let the user know which values were last inputted into the system. This function is used differently in the 8 classes AOO, AAI,VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.74 is_digit_check(P)

- This function checks if the input is a valid number. It will return true if the characters only contain numbers or if the string is empty, and it returns false if P contains any other characters. The black box parameters for this function is the input P which takes any numerical value, and the output is a boolean value that either returns true or false depending on if it is a number or not respectively.

3.1.1.75 save_text()

- This function first checks if the inputs are valid by calling the function inputs_correct(). If the inputs are valid, then it will write the data into the data variable which holds the parameters for the specific AOOR mode. The function then opens the users.csv file, and creates a new temporary file that will take any of the

changes and write them into the new temp file created. Then it replaces the original file with the temporary file. An array called `curr_data` is created which includes the username, password, and the updated values. It writes this data into the `currentusers.csv` file. We then call the `generate_ecg()` function to create a graph. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.76 inputs_correct()

- This function checks for if the inputs are correct. It checks for all of the different limitations based on the Pacemaker document on page 34. This function is called in the above function `save_text()`. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes to the limitation of the parameters set in the Pacemaker document

*3.1.1.77 chosen_activity_thresh(self, *args)*

- This function checks the selected value of the activity threshold combobox, or dropdown menu, and assigns a value to the activity Thresh variable based on the selected option. In terms of blackbox behaviour, the input would be the user's selection from the dropdown menu and the output would be the assigned number based on that selection.

3.1.1.78 reset_ecg()

- The purpose of this function is to reset the ECG simulation, stopping any graph updates, clears the signal data, clears the graph if the window is open, and resets the time counter. There are no particular parameters for this function besides "self", which is a defining method in a python function. This function works by first checking if the graph is running and if it is, it calls on the function "`self.stop_ecg_update()`" which stops the ECG update loop. The function clears the ECG signal data by initializing an empty array to the data array and resets the time counter to zero. Lastly, the function clears the graph. In terms of the blackbox behaviour, there are no particular inputs as there are no parameters, but the function takes in the ECG graph and outputs a cleared graph.

3.1.1.79 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)

- The purpose of this function is to simulate an ECG waveform based on the data taken in by the parameters. The parameters are duration, lrl, url, atrial_amp, and atrial_pulse_width. The function works by first setting the time step to the time vector “dt”. It then initializes the ECG signal and calculates the beat duration at LRL and URL. It then creates the ECG waveform by running a while loop until time_lapsed exceeds the specified duration. The beat iterates between LRL and URL. Then the function calculates the P-wave, QRS complex, and T-wave. The function creates the gaussian components for the ECG signal for P-wave, QRS complex, and T-wave and adds them to the ECG signal. Lastly, the time lapse variable is updated and the beat duration is added. In terms of the blackbox behaviour, the function takes in duration, lrl, url, atrial_amp, and atrial_pulse_width as inputs, and returns an ECG graph and the time vector “t”.

3.1.1.80 update_ecg_continuously(self)

- The purpose behind this function is to create and update an ECG signal with the option of pausing the signal as well. There are no particular input parameters besides “self” which is a defining method in a python function. This function works by first initializing a few variables: “segment_duration”, “self.running”, and “self.paused”. When self.running is true, a loop continuously runs and if self.paused is true, the loop is paused for 1 second then continues the iteration of the loop. In the loop, a try-block is executed where the stored variables are fetched for the AOO mode and the function “generate_ecg_waveform” is called to generate a new segment of the ecg waveform. If there is no ECG waveform present then the function is returned. The new segment fetched is then concatenated to the current ECG signal data and the time is then incremented accordingly. The current ECG data is then cleared from the graph and the new data is plotted. The function then sleeps based on the segment duration. If an error occurs in the try block then the function will sleep for 1 second and continue to the next iteration in the while loop. In terms of the blackbox behaviour, there are no particular inputs to the function besides the stored variables of the class and outputs an updated ECG waveform.

3.1.1.81 generate_ecg()

- This function creates a(gcf) graph based upon the inputted data from parameters for the set operating mode. This function uses the matplotlib library and the numpy library.

3.1.1.82 stop_ecg_update(self)

- The purpose of this function is to pause or stop the ECG update loop and therefore stop the ECG graph's constant updates. There are no input parameters to this function besides "self" which is a defining method in a python function. More specifically, this function works by setting "self.paused" to True, disables the "stop" button, and enables the "resume" button. In terms of blackbox behaviour, the function does not take in any parameters, it only takes in the updated ECG graph and outputs a paused ECG graph.

3.1.1.83 resume_ecg_update(self)

- The purpose of this function is to resume the ECG graph updates after the user stops the graph. There are no input parameters besides "self" which is a defining method in a python function. The function works by first checking if the graph is not running, if it is not running it sets "self.running" to True and begins a new thread "self.ecg_update_thread". The function then sets "self.paused" to False so the program knows the ECG updates should resume, and then it disables the "resume" button and enables the "stop" button. In terms of blackbox behaviour, the function does not have any input parameters, but rather the input is a paused graph, and the output is an updating ECG graph.

Global Variables:

Global variables for each mode include the label and entry text box for each of their respective programmable parameters listed in the pacemaker document. Each entry box is input guarded by using the `is_digit_check()` method to ensure only numbers are inputted with the exception of activity threshold, which is controlled using the `chosen_activity_thresh()` method described above. The buttons that generate and control the ECG waveform as well as the dataflow are also included as global variables.

Private Functions:

*3.1.1.84 __init__(self,*args,**kwargs)*

- This is the constructor for the class AOOR. The constructor sets up the container used to hold each frame, or child class, and calls the `show_frame()` function with 'Login' as the parameter to start the program.

Class VOOR():

This class serves as the VOOR operating mode page where users may input numerical values and display graphs and save data for the user

Public Functions:

3.1.1.85 load_saved_data(self)

- This function is used to load the saved data of the current user. It uses the `get_curr_user()` function and sets the filename to “users.csv”. This function opens the csv file and searches for a row where the first element matches `row[0]`, the value of the current users. If the row is found, it will extract the data from the nine values. This function throws an error if a `ValueError` is found and lets the user know that if the data is from a different mode, they must input data from the current operating mode. This function also has labels that allow the data values to be displayed to let the user know which values were last inputted into the system. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOO, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.86 is_digit_check(P)

- This function checks if the input is a valid number. It will return true if the characters only contain numbers or if the string is empty, and it returns false if P contains any other characters. The black box parameters for this function is the input P which takes any numerical value, and the output is a boolean value that either returns true or false depending on if it is a number or not respectively.

3.1.1.87 save_text()

- This function first checks if the inputs are valid by calling the function `inputs_correct()`. If the inputs are valid, then it will write the data into the data variable which holds the parameters for the specific VOOR mode. The function then opens the users.csv file, and creates a new temporary file that will take any of the changes and write them into the new temp file created. Then it replaces the original file with the temporary file. An array called `curr_data` is created which includes the username, password, and the updated values. It writes this data into the `currentusers.csv` file. We then call the `generate_ecg()` function to create a graph. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOO, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.88 inputs_correct()

- This function checks for if the inputs are correct. It checks for all of the different limitations based on the Pacemaker document on page 34. This function is called in the above function `save_text()`. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes to the limitation of the parameters set in the Pacemaker document

*3.1.1.89 chosen_activity_thresh(self, *args)*

- This function checks the selected value of the activity threshold combobox, or dropdown menu, and assigns a value to the activity Thresh variable based on the selected option. In terms of blackbox behaviour, the input would be the user's selection from the dropdown menu and the output would be the assigned number based on that selection.

3.1.1.90 reset_ecg()

- The purpose of this function is to reset the ECG simulation, stopping any graph updates, clears the signal data, clears the graph if the window is open, and resets the time counter. There are no particular parameters for this function besides "self", which is a defining method in a python function. This function works by first checking if the graph is running and if it is, it calls on the function "self.stop_ecg_update()" which stops the ECG update loop. The function clears the ECG signal data by initializing an empty array to the data array and resets the time counter to zero. Lastly, the function clears the graph. In terms of the blackbox behaviour, there are no particular inputs as there are no parameters, but the function takes in the ECG graph and outputs a cleared graph.

3.1.1.91 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)

- The purpose of this function is to simulate an ECG waveform based on the data taken in by the parameters. The parameters are duration, lrl, url, atrial_amp, and atrial_pulse_width. The function works by first setting the time step to the time vector "dt". It then initializes the ECG signal and calculates the beat duration at LRL and URL. It then creates the ECG waveform by running a while loop until time_lapsed exceeds the specified duration. The beat iterates between LRL and URL. Then the function calculates the P-wave, QRS complex, and T-wave. The function

creates the gaussian components for the ECG signal for P-wave, QRS complex, and T-wave and adds them to the ECG signal. Lastly, the time lapse variable is updated and the beat duration is added. In terms of the blackbox behaviour, the function takes in duration, lrl, url, atrial_amp, and atrial_pulse_width as inputs, and returns an ECG graph and the time vector “t”.

3.1.1.92 update_ecg_continuously(self)

- The purpose behind this function is to create and update an ECG signal with the option of pausing the signal as well. There are no particular input parameters besides “self” which is a defining method in a python function. This function works by first initializing a few variables: “segment_duration”, “self.running”, and “self.paused”. When self.running is true, a loop continuously runs and if self.paused is true, the loop is paused for 1 second then continues the iteration of the loop. In the loop, a try-block is executed where the stored variables are fetched for the AOO mode and the function “generate_ecg_waveform” is called to generate a new segment of the ecg waveform. If there is no ECG waveform present then the function is returned. The new segment fetched is then concatenated to the current ECG signal data and the time is then incremented accordingly. The current ECG data is then cleared from the graph and the new data is plotted. The function then sleeps based on the segment duration. If an error occurs in the try block then the function will sleep for 1 second and continue to the next iteration in the while loop. In terms of the blackbox behaviour, there are no particular inputs to the function besides the stored variables of the class and outputs an updated ECG waveform.

3.1.1.93 generate_ecg()

- This function creates a gcg graph based upon the inputted data from parameters for the set operating mode. This function uses the matplotlib library and the numpy library.

3.1.1.94 stop_ecg_update(self)

- The purpose of this function is to pause or stop the ECG update loop and therefore stop the ECG graph’s constant updates. There are no input parameters to this function besides “self” which is a defining method in a python function. More specifically, this function works by setting “self.paused” to True, disables the “stop”button, and enables the “resume” button. In terms of blackbox behaviour, the function does not

take in any parameters, it only takes in the updated ECG graph and outputs a paused ECG graph.

3.1.1.95 resume_ecg_update(self)

- The purpose of this function is to resume the ECG graph updates after the user stops the graph. There are no input parameters besides “self” which is a defining method in a python function. The function works by first checking if the graph is not running, if it is not running it sets “self.running” to True and begins a new thread “self.ecg_update_thread”. The function then sets “self.paused” to False so the program knows the ECG updates should resume, and then it disables the “resume” button and enables the “stop” button. In terms of blackbox behaviour, the function does not have any input parameters, but rather the input is a paused graph, and the output is an updating ECG graph.

Global Variables:

Global variables for each mode include the label and entry text box for each of their respective programmable parameters listed in the pacemaker document. Each entry box is input guarded by using the `is_digit_check()` method to ensure only numbers are inputted with the exception of activity threshold, which is controlled using the `chosen_activity_thresh()` method described above. The buttons that generate and control the ECG waveform as well as the dataflow are also included as global variables.

Private Functions:

*3.1.1.96 __init__(self,*args,**kwargs)*

- This is the constructor for the class VOOR. The constructor sets up the container used to hold each frame, or child class, and calls the `show_frame()` function with ‘Login’ as the parameter to start the program.

Class AAIR():

This class serves as the AAIR operating mode page where users may input numerical values and display graphs and save data for the user

Public Functions:

3.1.1.97 load_saved_data(self)

- This function is used to load the saved data of the current user. It uses the `get_curr_user()` function and sets the filename to “users.csv”. This function opens the csv file and searches for a row where the first element matches `row[0]`, the value of the current users. If the row is found, it will extract the data from the fourteen values. This function throws an error if a `ValueError` is found and lets the user know that if the data is from a different mode, they must input data from the current operating mode. This function also has labels that allow the data values to be displayed to let the user know which values were last inputted into the system. This function is used differently in the 8 classes AOO, AAI,VOO, VVI, AOO, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.98 is_digit_check(P)

- This function checks if the input is a valid number. It will return true if the characters only contain numbers or if the string is empty, and it returns false if P contains any other characters. The black box parameters for this function is the input P which takes any numerical value, and the output is a boolean value that either returns true or false depending on if it is a number or not respectively.

3.1.1.99 save_text()

- This function first checks if the inputs are valid by calling the function `inputs_correct()`. If the inputs are valid, then it will write the data into the data variable which holds the parameters for the specific AAIR mode. The function then opens the users.csv file, and creates a new temporary file that will take any of the changes and write them into the new temp file created. Then it replaces the original file with the temporary file. An array called `curr_data` is created which includes the username, password, and the updated values. It writes this data into the `currentusers.csv` file. We then call the `generate_ecg()` function to create a graph. This function is used differently in the 8 classes AOO, AAI,VOO, VVI, AOO, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.100 inputs_correct()

- This function checks for if the inputs are correct. It checks for all of the different limitations based on the Pacemaker document on page 34. This function is called in the above function `save_text()`. This function is used differently in the 8 classes AOO,

AAI,VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes to the limitation of the parameters set in the Pacemaker document

*3.1.1.101 chosen_activity_thresh(self, *args)*

- This function checks the selected value of the activity threshold combobox, or dropdown menu, and assigns a value to the activity Thresh variable based on the selected option. In terms of blackbox behaviour, the input would be the user's selection from the dropdown menu and the output would be the assigned number based on that selection.

3.1.1.102 reset_ecg()

- The purpose of this function is to reset the ECG simulation, stopping any graph updates, clears the signal data, clears the graph if the window is open, and resets the time counter. There are no particular parameters for this function besides "self", which is a defining method in a python function. This function works by first checking if the graph is running and if it is, it calls on the function "self.stop_ecg_update()" which stops the ECG update loop. The function clears the ECG signal data by initializing an empty array to the data array and resets the time counter to zero. Lastly, the function clears the graph. In terms of the blackbox behaviour, there are no particular inputs as there are no parameters, but the function takes in the ECG graph and outputs a cleared graph.

3.1.1.103 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)

- The purpose of this function is to simulate an ECG waveform based on the data taken in by the parameters. The parameters are duration, lrl, url, atrial_amp, and atrial_pulse_width. The function works by first setting the time step to the time vector "dt". It then initializes the ECG signal and calculates the beat duration at LRL and URL. It then creates the ECG waveform by running a while loop until time_lapsed exceeds the specified duration. The beat iterates between LRL and URL. Then the function calculates the P-wave, QRS complex, and T-wave. The function creates the gaussian components for the ECG signal for P-wave, QRS complex, and T-wave and adds them to the ECG signal. Lastly, the time lapse variable is updated and the beat duration is added. In terms of the blackbox behaviour, the function takes in duration, lrl, url, atrial_amp, and atrial_pulse_width as inputs, and returns an ECG graph and the time vector "t".

3.1.1.104 update_ecg_continuously(self)

- The purpose behind this function is to create and update an ECG signal with the option of pausing the signal as well. There are no particular input parameters besides “self” which is a defining method in a python function. This function works by first initializing a few variables: “segment_duration”, “self.running”, and “self.paused”. When self.running is true, a loop continuously runs and if self.paused is true, the loop is paused for 1 second then continues the iteration of the loop. In the loop, a try-block is executed where the stored variables are fetched for the AOO mode and the function “generate_ecg_waveform” is called to generate a new segment of the ecg waveform. If there is no ECG waveform present then the function is returned. The new segment fetched is then concatenated to the current ECG signal data and the time is then incremented accordingly. The current ECG data is then cleared from the graph and the new data is plotted. The function then sleeps based on the segment duration. If an error occurs in the try block then the function will sleep for 1 second and continue to the next iteration in the while loop. In terms of the blackbox behaviour, there are no particular inputs to the function besides the stored variables of the class and outputs an updated ECG waveform.

3.1.1.105 generate_ecg()

- This function creates a(gcf) graph based upon the inputted data from parameters for the set operating mode. This function uses the matplotlib library and the numpy library.

3.1.1.106 stop_ecg_update(self)

- The purpose of this function is to pause or stop the ECG update loop and therefore stop the ECG graph’s constant updates. There are no input parameters to this function besides “self” which is a defining method in a python function. More specifically, this function works by setting “self.paused” to True, disables the “stop”button, and enables the “resume” button. In terms of blackbox behaviour, the function does not take in any parameters, it only takes in the updated ECG graph and outputs a paused ECG graph.

3.1.1.107 resume_ecg_update(self)

- The purpose of this function is to resume the ECG graph updates after the user stops the graph. There are no input parameters besides “self” which is a defining method in

a python function. The function works by first checking if the graph is not running, if it is not running it sets “self.running” to True and begins a new thread “self.ecg_update_thread”. The function then sets “self.paused” to False so the program knows the ECG updates should resume, and then it disables the “resume” button and enables the “stop” button. In terms of blackbox behaviour, the function does not have any input parameters, but rather the input is a paused graph, and the output is an updating ECG graph.

Global Variables:

Global variables for each mode include the label and entry text box for each of their respective programmable parameters listed in the pacemaker document. Each entry box is input guarded by using the `is_digit_check()` method to ensure only numbers are inputted with the exception of activity threshold, which is controlled using the `chosen_activity_thresh()` method described above. The buttons that generate and control the ECG waveform as well as the dataflow are also included as global variables.

Private Functions:

*3.1.1.108 __init__(self,*args,**kwargs)*

- This is the constructor for the class AAIR. The constructor sets up the container used to hold each frame, or child class, and calls the `show_frame()` function with ‘Login’ as the parameter to start the program.

Class VVIRQ:

This class serves as the VVIR operating mode page where users may input numerical values and display graphs and save data for the user

Public Functions:

3.1.1.109 load_saved_data(self)

- This function is used to load the saved data of the current user. Its input is the “users.csv” file and its output are the text widgets used to display the values. It uses the `get_curr_user()` function and sets the filename to “users.csv”. This function opens the csv file and searches for a row where the first element matches `row[0]`, the value of the current users. If the row is found, it will extract the data from the thirteen values. This function throws an error if a `ValueError` is found and lets the user know

that if the data is from a different mode, they must input data from the current operating mode. This function also sets up labels that allow the data values to be displayed to let the user know which values were last inputted into the system. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in.

3.1.1.110 is_digit_check(P)

- This function takes the parameter 'P' and checks if the input is a valid number. It will return true if the characters only contain numbers or if the string is empty, and it returns false if P contains any other characters. The black box parameters for this function is the input P which takes any numerical value, and the output is a boolean value that either returns true or false depending on if it is a number or not respectively.

3.1.1.111 save_text()

- This function first checks if the inputs are valid by calling the function `inputs_correct()`. If the inputs are valid, then it will write the data into the data variable which holds the parameters for the specific VVIR mode. The function then opens the `users.csv` file, and creates a new temporary file that will take any of the changes and write them into the new temp file created. Then it replaces the original file with the temporary file. Then an array called `data` is created which includes all the inputted programmable parameters. We then call `send_data_via_serial(data)` to send the data to the pacemaker device. An array called `curr_data` is created which includes the username, password, and the updated values. It writes this data into the `currentusers.csv` file. We then call the `generate_ecg()` function to create a graph. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes of the amount of parameters it takes in. In terms of blackbox behaviour, the input would be the text the user inputted into the parameter entry boxes and the output would be the transfer of this data into the `users.csv` file and the pacemaker device

3.1.1.112 inputs_correct()

- This function checks for if the inputs are correct. It checks for all of the different limitations based on the Pacemaker document on page 34. In terms of blackbox behaviour, the input is the user inputted values in the respective programmable parameters textbox, and the output is a boolean value that evaluates to true or false depending on if the inputs match the regulations. This function is called in the above

function `save_text()`. This function is used differently in the 8 classes AOO, AAI, VOO, VVI, AOOR, VOOR, AAIR, and VVIR with very minor changes to the limitation of the parameters set in the Pacemaker document.

*3.1.1.113 chosen_activity_thresh(self, *args)*

- This function checks the selected value of the activity threshold combobox, or dropdown menu, and assigns a value to the activity Thresh variable based on the selected option. In terms of blackbox behaviour, the input would be the user's selection from the dropdown menu and the output would be the assigned number based on that selection.

3.1.1.114 reset_ecg()

- The purpose of this function is to reset the ECG simulation, stopping any graph updates, clears the signal data, clears the graph if the window is open, and resets the time counter. There are no particular parameters for this function besides "self", which is a defining method in a python function. This function works by first checking if the graph is running and if it is, it calls on the function "self.stop_ecg_update()" which stops the ECG update loop. The function clears the ECG signal data by initializing an empty array to the data array and resets the time counter to zero. Lastly, the function clears the graph. In terms of the blackbox behaviour, there are no particular inputs as there are no parameters, but the function takes in the ECG graph and outputs a cleared graph.

3.1.1.115 create_ecg_waveform(self, duration, lrl, url, atrial_amp, atrial_pulse_width)

- The purpose of this function is to simulate an ECG waveform based on the data taken in by the parameters. The parameters are duration, lrl, url, atrial_amp, and atrial_pulse_width. The function works by first setting the time step to the time vector "dt". It then initializes the ECG signal and calculates the beat duration at LRL and URL. It then creates the ECG waveform by running a while loop until time_lapsed exceeds the specified duration. The beat iterates between LRL and URL. Then the function calculates the P-wave, QRS complex, and T-wave. The function creates the gaussian components for the ECG signal for P-wave, QRS complex, and T-wave and adds them to the ECG signal. Lastly, the time lapse variable is updated and the beat duration is added. In terms of the blackbox behaviour, the function takes in duration, lrl, url, atrial_amp, and atrial_pulse_width as inputs, and returns an ECG graph and the time vector "t".

3.1.1.116 update_ecg_continuously(self)

- The purpose behind this function is to create and update an ECG signal with the option of pausing the signal as well. There are no particular input parameters besides “self” which is a defining method in a python function. This function works by first initializing a few variables: “segment_duration”, “self.running”, and “self.paused”. When self.running is true, a loop continuously runs and if self.paused is true, the loop is paused for 1 second then continues the iteration of the loop. In the loop, a try-block is executed where the stored variables are fetched for the AOO mode and the function “generate_ecg_waveform” is called to generate a new segment of the ecg waveform. If there is no ECG waveform present then the function is returned. The new segment fetched is then concatenated to the current ECG signal data and the time is then incremented accordingly. The current ECG data is then cleared from the graph and the new data is plotted. The function then sleeps based on the segment duration. If an error occurs in the try block then the function will sleep for 1 second and continue to the next iteration in the while loop. In terms of the blackbox behaviour, there are no particular inputs to the function besides the stored variables of the class and outputs an updated ECG waveform.

3.1.1.117 generate_ecg()

- This function creates a(gcf) graph based upon the inputted data from parameters for the set operating mode. This function uses the matplotlib library and the numpy library.

3.1.1.118 stop_ecg_update(self)

- The purpose of this function is to pause or stop the ECG update loop and therefore stop the ECG graph’s constant updates. There are no input parameters to this function besides “self” which is a defining method in a python function. More specifically, this function works by setting “self.paused” to True, disables the “stop”button, and enables the “resume” button. In terms of blackbox behaviour, the function does not take in any parameters, it only takes in the updated ECG graph and outputs a paused ECG graph.

3.1.1.119 resume_ecg_update(self)

- The purpose of this function is to resume the ECG graph updates after the user stops the graph. There are no input parameters besides “self” which is a defining method in

a python function. The function works by first checking if the graph is not running, if it is not running it sets “self.running” to True and begins a new thread “self.ecg_update_thread”. The function then sets “self.paused” to False so the program knows the ECG updates should resume, and then it disables the “resume” button and enables the “stop” button. In terms of blackbox behaviour, the function does not have any input parameters, but rather the input is a paused graph, and the output is an updating ECG graph.

Global Variables:

Global variables for each mode include the label and entry text box for each of their respective programmable parameters listed in the pacemaker document. Each entry box is input guarded by using the `is_digit_check()` method to ensure only numbers are inputted with the exception of activity threshold, which is controlled using the `chosen_activity_thresh()` method described above. The buttons that generate and control the ECG waveform as well as the dataflow are also included as global variables.

Private Functions:

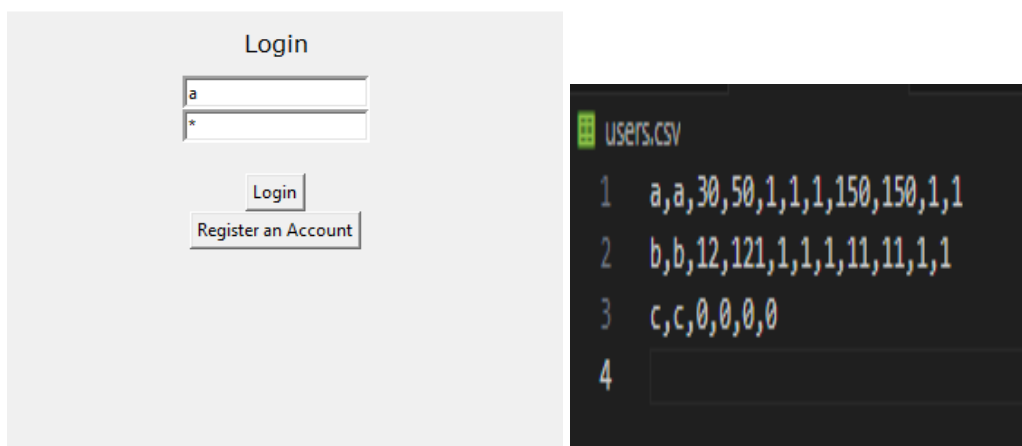
3.1.1.120 __init__(self, parent, controller)

- This is the constructor for the class VVIR. The constructor sets up the frame used for the VVIR class and holds all public functions and variables used in the class.

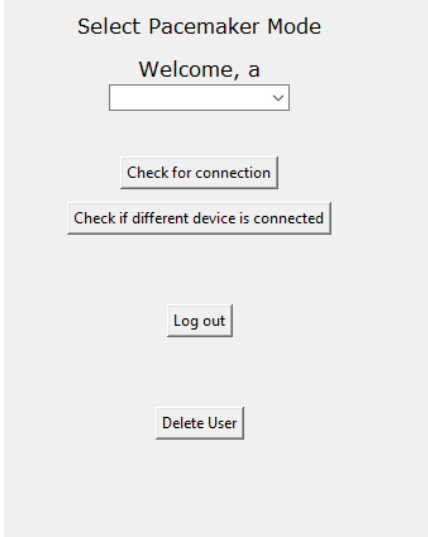
4. Testing

4.1 Running Program

User can login based on correct username and password



Once logged in, the program takes the user to the next page where the user can select different modes, check if the device is connected, check if a different device is connected, or the user may log out

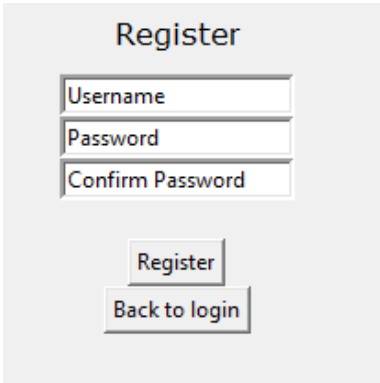


The screenshot shows a web interface titled "Select Pacemaker Mode". Below the title, it says "Welcome, a" followed by a dropdown menu. There are three buttons: "Check for connection", "Check if different device is connected", and "Log out". At the bottom, there is a "Delete User" button.

By clicking the logout button, it takes the user back to the login page where they can click on the register account page to register themselves as a user.

4.2 User Registration

The page prompts the user to add a username, password, and to confirm the password. There are many cases in which the page will not let the user register an account.



The screenshot shows a web interface titled "Register". It contains three input fields: "Username", "Password", and "Confirm Password". Below these fields are two buttons: "Register" and "Back to login".

4.2.1 Test Cases:

In the image below, users are unable to press register without inputting a username first.

Register

Username

Password

Confirm Password

Please input a valid username.

Register

Back to login

This screenshot shows a web registration form titled "Register". It contains three input fields: "Username", "Password", and "Confirm Password". Below the "Username" field, a red error message states "Please input a valid username." The "Password" and "Confirm Password" fields are empty. At the bottom, there are two buttons: "Register" and "Back to login".

In the image below, users are not able to register with just a username.

Register

admin

Password

Confirm Password

Please input a valid password

Register

Back to login

This screenshot shows the same "Register" form. The "Username" field now contains the text "admin". The "Password" and "Confirm Password" fields are empty. A red error message below the "Password" field reads "Please input a valid password". The "Register" and "Back to login" buttons remain at the bottom.

In the image below, users must input a password that matches.

Register

admin

Confirm Password

Passwords do not match.

Register

Back to login

This screenshot shows the "Register" form with "admin" in the "Username" field and "*****" in the "Password" field. The "Confirm Password" field is empty. A red error message below the "Confirm Password" field states "Passwords do not match." The "Register" and "Back to login" buttons are still present at the bottom.

In the image below, users can't have spaces in between username

Register

admin d

*

*

Username cannot contain spaces.

Register

Back to login

In the image below, if the username already exists, users are not allowed to create an account.

The screenshot shows a web application interface. On the left, there is a registration form titled "Register". The form has three input fields: the first contains the text "a", the second contains "*****", and the third contains "*****". Below the form, there is a message: "Username already exists. Please use a different Username." At the bottom of the form, there are two buttons: "Register" and "Back to login". On the right, there is a dark-themed window titled "users.csv" showing a list of users. The list has four rows, each with a number in the first column and a CSV-formatted string in the second column. The strings are: "a,a,30,50,1,1,1,150,150,1,1", "b,b,12,121,1,1,1,11,11,1,1", "c,c,0,0,0,0", and an empty row.

| | users.csv |
|---|-----------------------------|
| 1 | a,a,30,50,1,1,1,150,150,1,1 |
| 2 | b,b,12,121,1,1,1,11,11,1,1 |
| 3 | c,c,0,0,0,0 |
| 4 | |

In the image below, if the csv file hold 10 users, a new account can not be created

[illegible]

In the image below, if the user inputs a correct username and password they are able to register an account.

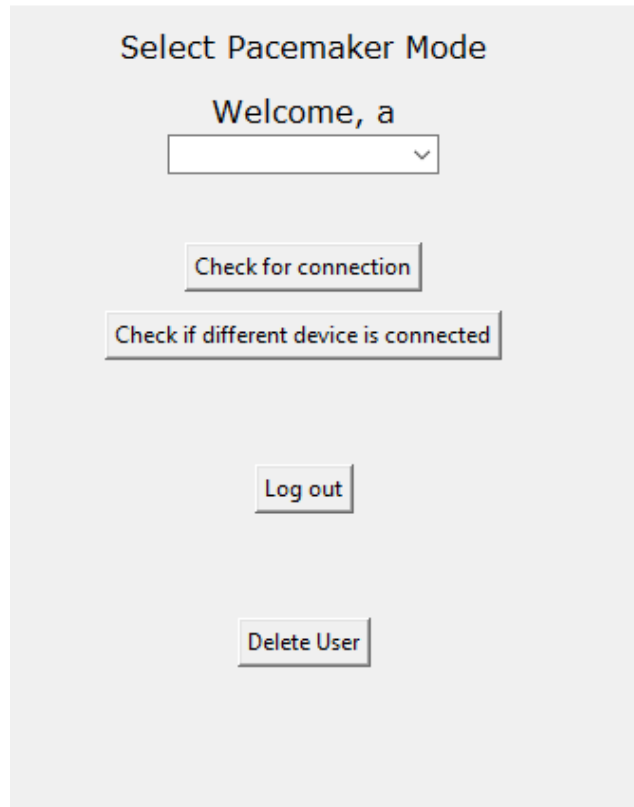
The screenshot displays a web application interface. At the top, there is a 'Register' form with three input fields: the first contains 'admin', the second contains '*****', and the third contains '*****'. Below the form are two buttons: 'Register' and 'Back to login'. A modal dialog box is open in the foreground, featuring a blue information icon and the text 'Successfully registered. Please login to continue.' with an 'OK' button.

4.3 User Login

After the user has successfully registered an account and clicks OK on the pop up message box, the program takes the user back to the Login page where the user can now input their username and password. If the user is to input an invalid username or password, the program will tell the user that the username or password is incorrect and to proceed to try again as displayed in the image below.

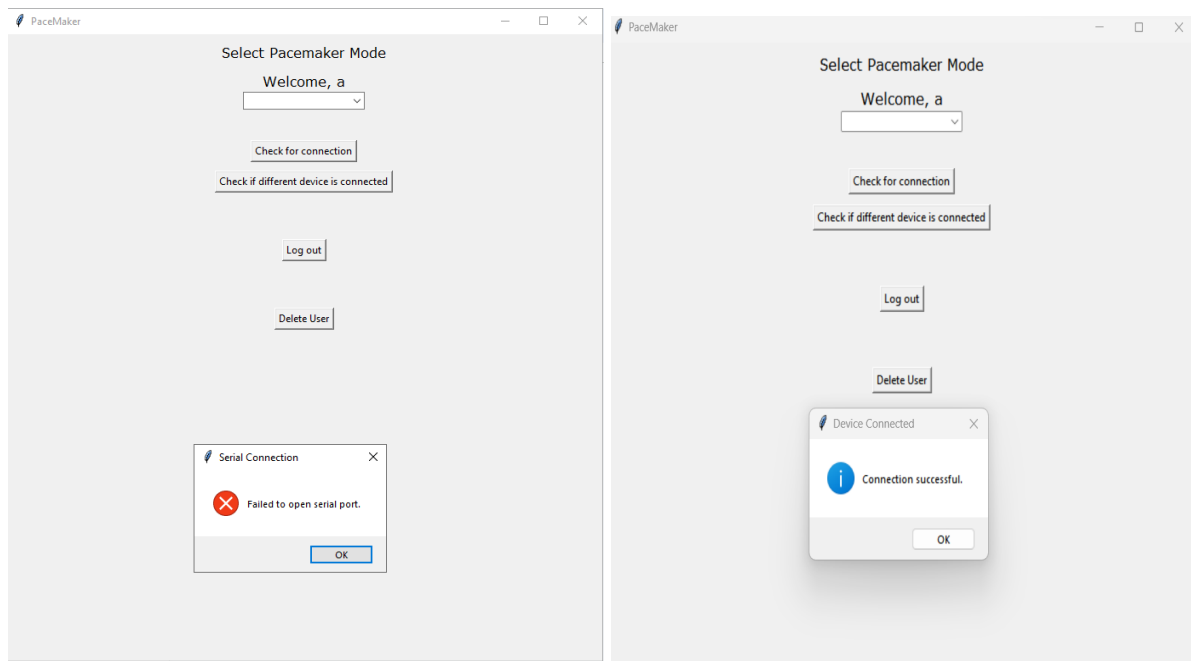
After the user has inputted a correct username and password, the program will take the user to the pacemaker mode selection as described above.

4.4 Welcome Page

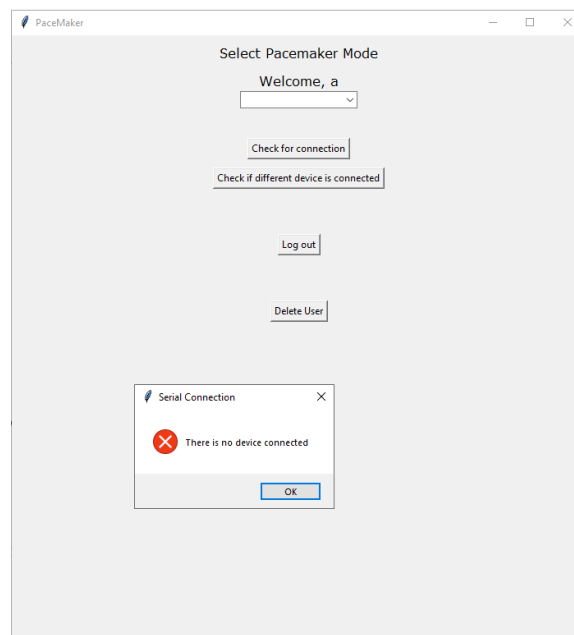


The screenshot shows a web interface titled "Select Pacemaker Mode". Below the title, it says "Welcome, a" followed by a dropdown menu. There are four buttons arranged vertically: "Check for connection", "Check if different device is connected", "Log out", and "Delete User".

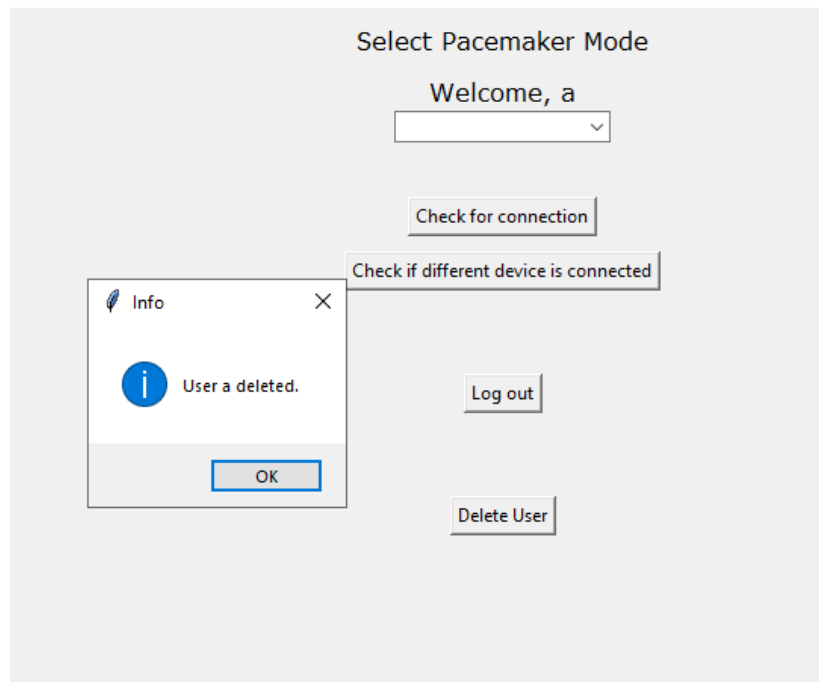
If the user clicks the check for connection button, the program will try to open the serial port. If there is a device connected, it will show the device connected, whereas if the device is not connected, the program will display an error box that says “failed to open the serial port”.



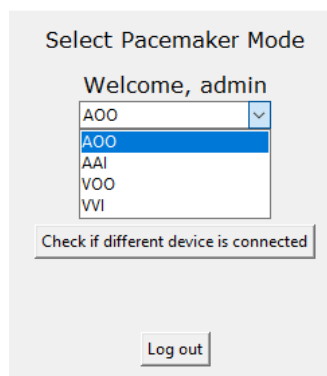
If the user selects to check if a different device is connected, the program will display whether or not a different device is connected or not in a similar manner to the check for connection button.



If the user decides that they want to delete their account as they don't want to use it anymore or want to create a new account but there is no space, they may do so by clicking the delete user button. After clicking the delete user button, they will be shown a messagebox and then they will be taken back to the login page. If they try to login with the same account, they will not be allowed to as that user does not exist any more.



By clicking the drop down menu, the user is able to select between four different modes, AOO, AAI, VOO, and VVI.



4.5 AOO Operating Mode

Upon selecting one of the modes, in this case the AOO mode, they are prompted to input different values for lower rate limit, upper rate limit, atrial amplification, and atrial pulse width.

AOO Mode

Lower Rate Limit

Upper Rate Limit

Atrial Amplification

Atrial Pulse Width

Submit Data

Retrieve Prev Data

Stop ECG

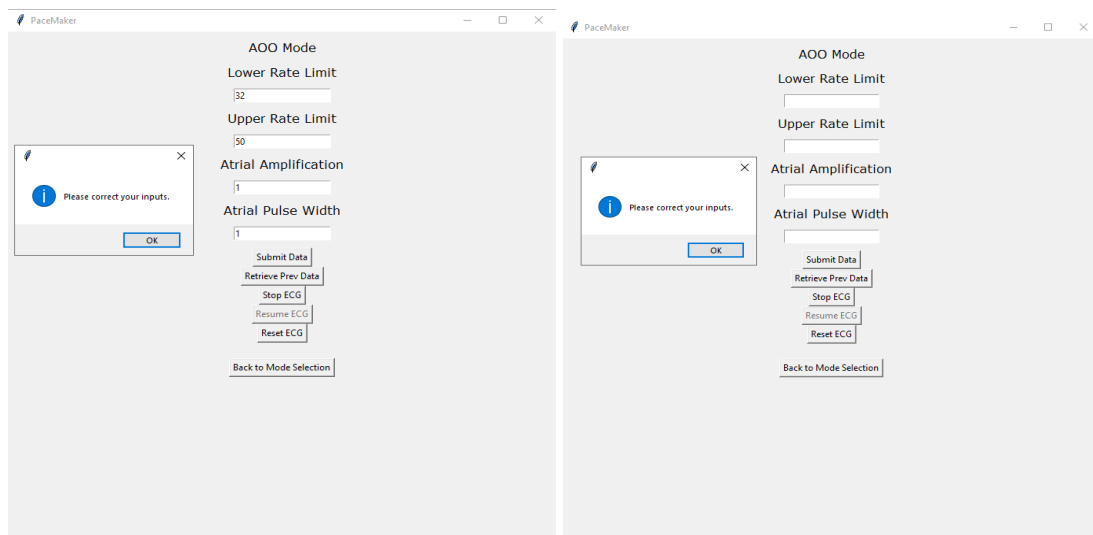
Resume ECG

Reset ECG

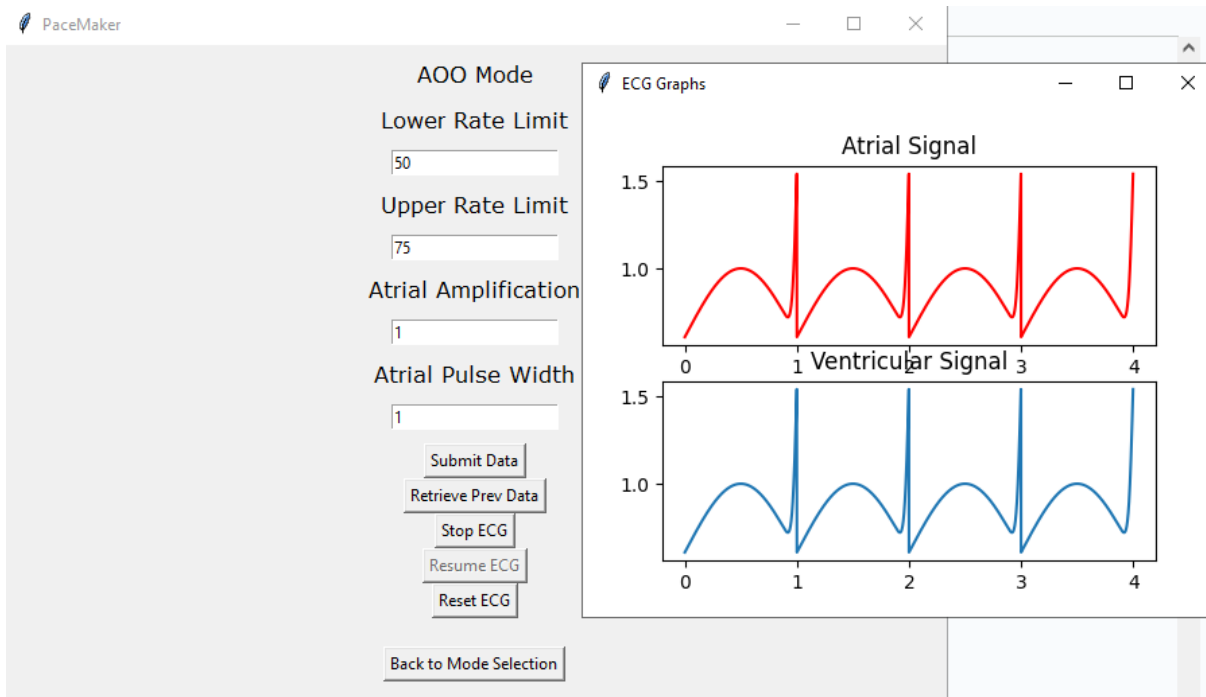
Back to Mode Selection

On this page, there are many cases that the user must follow for their input. They are not allowed to input blank spaces, as it will throw an error pop up when clicking submit data stating that they must correct their inputs. If they are to not follow the limits described on the pacemaker document on page 34, they are prompted with the same error message.

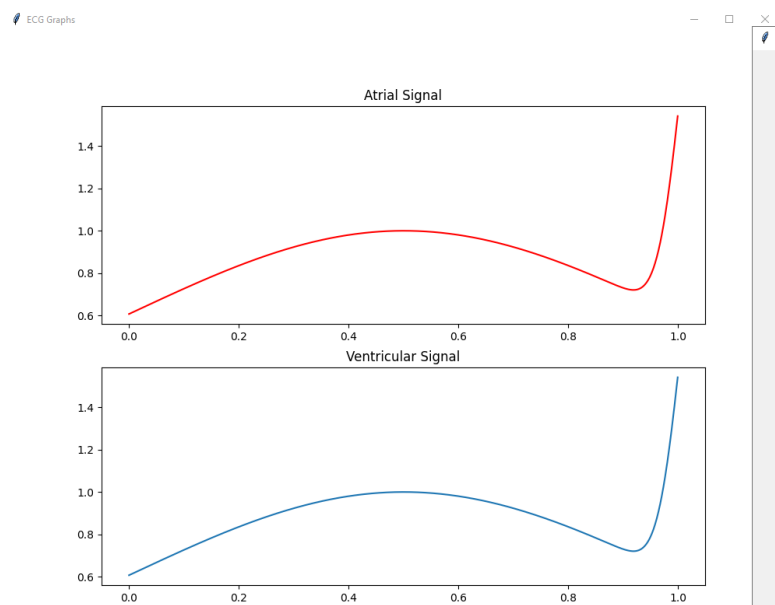
4.5.1 Test Cases:



In this mode, they must follow the correct limits described for the set operating mode. If the user inputs data within the correct range and following the correct increments described in the documentation, once they click the submit data button, data will be submitted and an ecg waveform is created.



In our Pacing modes, the user is now able to stop the ecg, resume the ecg, and reset the ecg. When the Stop ecg button is clicked, the ecg waveform will stop running, and when the resume button is clicked, the ecg will continue from the point it stopped at. If the reset ecg button is clicked, the ecg will reset, and the user must now click resume ecg to begin a new cycle of the ecg. In the image below, we can see that the waveform restarts after clicking the reset button and then the resume button.



In the case where the user logs back into the system and they want to see the previous data that they had submitted to confirm that they are using the correct data for their patient, they can press the Retrieve Prev Data button which will show the data from the previous Submit Data.

AOO Mode

Lower Rate Limit

Upper Rate Limit

Atrial Amplification

Atrial Pulse Width

Submit Data

Retrieve Prev Data

Stop ECG

Resume ECG

Reset ECG

Back to Mode Selection

lower lim: 50.0
upper lim: 75.0
atrial amp: 1.0
atrial pulse: 1.0

If the user wants to go and select a different operating mode, they may do that by selecting the Back to Mode selection.

If the user submits data from a different operating mode and tries to retrieve data in a different operating mode they will be thrown with an error. For example, if the user submits data from the VOO mode, and tries to retrieve the data from the AOO mode, the error message will be shown indicating that the user must input new data for the AOO mode.

VOO Mode

Lower Rate Limit

Upper Rate Limit

Ventricular Amplitude

Ventricular Pulse Width

submit data

Retrieve Prev Data

🔊 Error! Data is not for this op ...
✕

i

Error! Data is not for this op ...

OK

In the CSV file for the above case, it is shown that admin holds data from the AOO mode currently and thus the reason why the user is getting this error.

1 a,a,50.0,75.0,1.0,1.0

4.6 Other Operating Modes

The other three modes share extremely similar cases but differ in terms of the number of parameters that the user may input, and the different limitations for each operating mode.

| AAI Mode | VOO Mode | VVI Mode | AOOR Mode |
|---|---|---|---|
| Lower Rate Limit <input type="text"/> | Lower Rate Limit <input type="text"/> | Lower Rate Limit <input type="text"/> | Lower Rate Limit <input type="text"/> |
| Upper Rate Limit <input type="text"/> | Upper Rate Limit <input type="text"/> | Upper Rate Limit <input type="text"/> | Upper Rate Limit <input type="text"/> |
| Atrial Amplification <input type="text"/> | Ventricular Amplitude <input type="text"/> | Ventricular Amplitude <input type="text"/> | Maximum Sensor Rate <input type="text"/> |
| Atrial Pulse Width <input type="text"/> | Ventricular Pulse Width <input type="text"/> | Ventricular Pulse Width <input type="text"/> | Atrial Amplitude <input type="text"/> |
| Atrial Sensitivity <input type="text"/> | | Ventricular Sensitivity <input type="text"/> | Atrial Pulse Width <input type="text"/> |
| ARP <input type="text"/> | | VRP <input type="text"/> | Activity Threshold <input type="text"/> |
| PVARP <input type="text"/> | | Hysteresis <input type="text"/> | Reaction Time <input type="text"/> |
| Hysteresis <input type="text"/> | | Rate Smoothing <input type="text"/> | Response Factor <input type="text"/> |
| Rate Smoothing <input type="text"/> | | | Recovery Time <input type="text"/> |
| <input type="button" value="submit data"/> | <input type="button" value="submit data"/> | <input type="button" value="submit data"/> | <input type="button" value="Submit Data"/> |
| <input type="button" value="Retrieve Prev Data"/> | <input type="button" value="Retrieve Prev Data"/> | <input type="button" value="Retrieve Prev Data"/> | <input type="button" value="Retrieve Prev Data"/> |
| <input type="button" value="Stop ECG"/> | <input type="button" value="Stop ECG"/> | <input type="button" value="Stop ECG"/> | <input type="button" value="Stop ECG"/> |
| <input type="button" value="Resume ECG"/> | <input type="button" value="Resume ECG"/> | <input type="button" value="Resume ECG"/> | <input type="button" value="Resume ECG"/> |
| <input type="button" value="Reset ECG"/> | <input type="button" value="Reset ECG"/> | <input type="button" value="Reset ECG"/> | <input type="button" value="Reset ECG"/> |
| <input type="button" value="Back to Mode Selection"/> | <input type="button" value="Back to Mode Selection"/> | <input type="button" value="Back to Mode Selection"/> | <input type="button" value="Back to Mode Selection"/> |

| VOOR Mode | AAIR Mode | VVIR Mode |
|---|---|---|
| Lower Rate Limit <input type="text"/> | Lower Rate Limit <input type="text"/> | Lower Rate Limit <input type="text"/> |
| Upper Rate Limit <input type="text"/> | Upper Rate Limit <input type="text"/> | Upper Rate Limit <input type="text"/> |
| Maximum Sensor Rate <input type="text"/> | Maximum Sensor Rate <input type="text"/> | Maximum Sensor Rate <input type="text"/> |
| Ventricular Amplitude <input type="text"/> | Atrial Amplitude <input type="text"/> | Ventricular Amplitude <input type="text"/> |
| Ventricular Pulse Width <input type="text"/> | Atrial Pulse Width <input type="text"/> | Ventricular Pulse Width <input type="text"/> |
| Activity Threshold <input type="text"/> | Atrial Sensitivity <input type="text"/> | Ventricular Sensitivity <input type="text"/> |
| Reaction Time <input type="text"/> | ARP <input type="text"/> | VRP <input type="text"/> |
| Response Factor <input type="text"/> | | |
| Recovery Time <input type="text"/> | | |
| <input type="button" value="Submit Data"/> | <input type="button" value="Submit Data"/> | <input type="button" value="Submit Data"/> |
| <input type="button" value="Retrieve Prev Data"/> | <input type="button" value="Retrieve Prev Data"/> | <input type="button" value="Retrieve Prev Data"/> |
| <input type="button" value="Stop ECG"/> | <input type="button" value="Stop ECG"/> | <input type="button" value="Stop ECG"/> |
| <input type="button" value="Resume ECG"/> | <input type="button" value="Resume ECG"/> | <input type="button" value="Resume ECG"/> |
| <input type="button" value="Reset ECG"/> | <input type="button" value="Reset ECG"/> | <input type="button" value="Reset ECG"/> |
| <input type="button" value="Back to Mode Selection"/> | <input type="button" value="Back to Mode Selection"/> | <input type="button" value="Back to Mode Selection"/> |

Once the user finishes using the program, they may simply click on the X on the top right to close the program.