MECHTRON 3K04: Software Development

Assignment 2 – DCM Design

L02

Group 17

Name:

Zichuan You – youz7 – 400358506

Congling Tang – tangc61 – 400363198

Yueyue Sun – sun9 – 400294173

Xinyi Zhou – zhoux171– 400370690

Charlie O’Brien - obriec20- 400305005

Academic Integrity Statement

The student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University. [Charlie O'Brien, 400305005]

The student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University. [Xinyi Zhou, 400370690]

The student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University. [Yueyue Sun, 400294173]

The student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University. [Congling Tang, 400363198]

The student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University. [Zichuan You, 400358506]

Contents

System Overview

Overview

The DCM in assignment is considered as a GUI written in Python 3.10.11 using the customtkinter GUI framwork. The GUI contains the following features and pages: a login page, a sign up page, and the pages for setting and submitting parameters for different modes of the pacemaker after login into the user interface. In addition, the DCM view allows users to check the connection status of device and transmit the input parameters to the device (pacemaker). When the connected pacemaker changes, it should not affect the usage status and internally stored data of other pacemakers and would not impact other users' personal information and data.

In Assignment 1, it is not mandatory to establish an interactive connection between the DCM and the pacemaker. Therefore, data is not actually transmitted to the device to implement the full functionality of a real pacemaker. However, the assignment requires developers to recognize the status of connection and whether the device has been replaced. There are 2 buttons ("refresh" and "check") established to verify the requirements. As developers, we connected a pacemaker and used the serial component to identify the interface's status. However, we couldn't implement the device replacement feature since we did not have access to multiple pacemaker serial numbers. As a workaround, we created a "check" button. Every time it is pressed, it informs the user that the device is different, serving as a substitute for the actual expected result.

Technologies Used

Python

Python was selected as the language to write the GUI due to the various features and convenience of connection to other technologies. Python comes with a built-in standard library that provides a variety of components to assist tasks accomplishment. In assignment 1, there are several components are imported from the standard library: "**serial**" for serial communication, "**hashlib**" for storing and searching user personal information (username and password), and "**tkinter**" for creating pop-up windows to demonstrate warning or information to users.

It's worth noting that the component "**customtkinter**" used for building the GUI framework, is not part of the standard library, and it is downloaded and used by developers. More details on this can be found in the "**customtkinter**" section of "**Technologies Used**".

In addition, another reason for using Python to develop DCM is its excellent cross-platform compatibility due to the interpreter, which means that the same program can have the same functionality on other operating systems (MacOS, Linux, etc.), eliminating the need for users to worry about OS-specific adaptations.

Finally, Python leaves ample room for improvement and module addition, which makes future development and optimization more accessible [1].

PyCharm

PyCharm is a specialized Integrated Development Environment (IDE) designed specifically for Python. In the context of using Python as a development language, it provides functions

like code autocompletion and debugging capabilities. Additionally, **PyCharm** is a cross-platform tool, allowing developers to work seamlessly across different OS, ensuring smooth collaboration even when developers use different OS environments [1].

Customtkinter

Customtkinter is a framework used for GUI development, and it is not a part of the standard library, so developers need to download and install it separately. **Customtkinter** allows developers to create GUI interfaces with features like Entry and buttons, enabling users to input their information. Developers can also modify the attributes to make the interface look more aesthetically pleasing and user-friendly. The "command" attribute it provides allows buttons to invoke pre-defined functions to respond to user actions [2].

Hashlib

Hashlib is the python library we imported that was used to encrypt the password we wrote to the txt. file. Please refer to the following figure for the specific encryption method and encryption result [3].

```
hashed_password = hashlib.sha256(password.encode()).hexdigest()
users[username] = hashed_password
save_users()
```

Fig 1. hash encode funcion

```
1:6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b
2:d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35
3:4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49fce
5:ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d
6:e7f6c011776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683
```

Fig 2. Numbers on the left are usernames, encode passwords are on the right

Messagebox

Tkinter component is used in the program, but not as the primary framework for building the GUI. Instead, it is used to call the "**messagebox**" module to produce prompt windows that inform the users whether their input is valid and provide guidance on their next steps. This is achieved by using following codes [4].

```
from tkinter import messagebox
```

Serial

Python's **serial** library can detect if there is a device connected to a certain port on the computer, my pacemaker is connected to the COM3. Serial also enables our DCM to transmit and receive data, enabling data interaction between the DCM and the pacemaker. Serial does a good job of verifying that there is data being transferred [5].

Matplotlib

Matplotlib enables us to plot the function images of Atrium and Ventricle on the receiving end.

Unexecuted Plans

Tkinter

The reason we chose **cutsomtkinter** over **Tkinter** as our python UI library is that **customtkinter** serves as an extension library to **Tkinter**, and **customtkinter** has a more modern look and fuller set of features compared to **Tkinter**. However, it is worth noting that **Tkinter** is a library that comes with python and can be used without installation, which is more compatible and portable than **customtkinter** [4].

SQLite

SQLite is an embedded relational database management system (RDBMS) in Python standard library, which is used to store, manage and searching the data. Therefore, most software developers used it to develop a database by embedding it into application instead using an independent data server. However, an efficiency database is not compulsory in assignment 1, so a text file is enough to store 10 groups of personal information [6].

Database

users.txt

Considering that we only need to store 10 usernames and passwords for our users, we plan to use the txt. file as our database for storing usernames and passwords. User passwords are encrypted with the hashlib library to increase the security of the account. The following image shows the contents of the database file where we store user account information. In txt.file, to the left of the colon on each line is the username, and to the right of the colon is the user's password encrypted via hashtable.

```
Bill:6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b
Xinyi:d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35
Kevin:4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49fce
Allen:4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdabf8a
Tony:ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d
Tommy:ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d
Alice:19581e27de7ced00ff1ce50b2047e7a567c76b1cbaebabe5ef03f7c3017bb5b7
Peter:7902699be42c8a8e46fbbb4501726517e86b22c56a189f7625a6da49081b2451
Omen:7902699be42c8a8e46fbbb4501726517e86b22c56a189f7625a6da49081b2451
Jett:ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d
```

Fig 3. Username and encrypted password

In the sava_user () function, we first open users.txt in writing mode, and then use the .write () function to write to the database. Please refer to the following figure for the source code.

```
with open("users.txt", 'w') as file:
    for username, password in users.items():
        file.write(f"{username}:{password}\n")
```

Fig 4. Writing input into a text file

user_inputs.txt

Since there is no requirement in assignment1 to interact with pacemaker yet, we have temporarily saved the mode parameter that was supposed to be transferred into pacemaker into a local txt file. As shown below, in each line of user_inputs.txt, the name of the parameter is to the left of the colon, and the specific parameter is to the right of the colon. In the future, these parameters will be transferred into pacemaker via the serial port.

```
lower limit rate:30.0
upper limit rate:50.0
atrial amplitude:0.5
atrial pulse width:0.05
```

Fig 5. User input parameters

GUI workflow

Overview

This GUI is designed for users with specific cardiac needs. When a user opens the GUI, it displays a login screen, prompting the user to enter a username and password. If the user hasn't registered, they must click the "Register" button to access the registration screen. After completing the registration, they return to the login screen to log in again.

Upon successful login, a "Refresh" button appears. If it displays "Not Connected," the user must establish a device connection. Once the device is successfully connected, it leads to the mode selection screen. Here, four different modes are available for users to choose based on their circumstances and needs.

When a mode is selected, and the parameters are valid, the submission is successful. Otherwise, the user is prompted to re-enter the information.

Flowchart

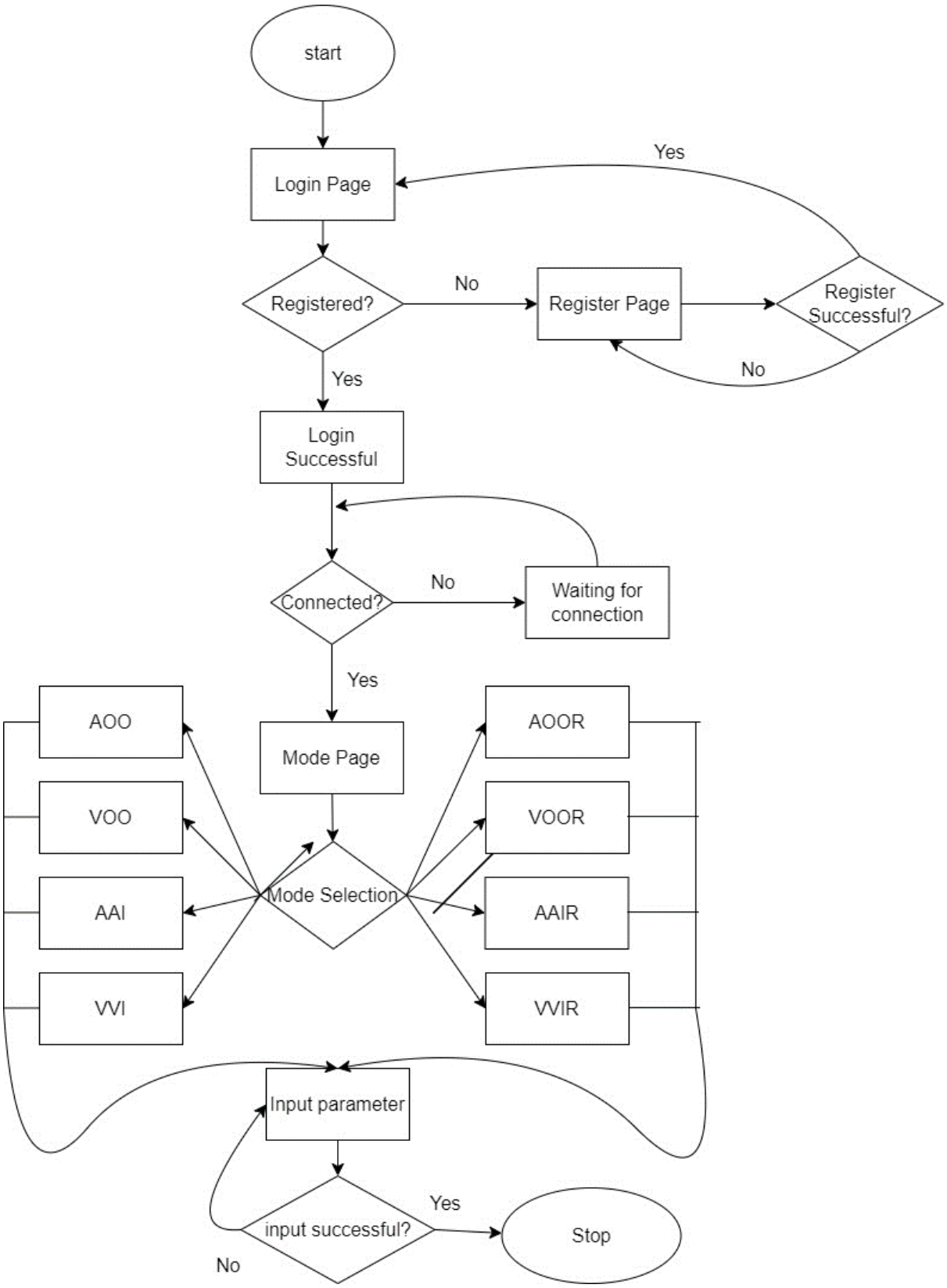

Fig 6. GUI flowchart

Finite State Machine

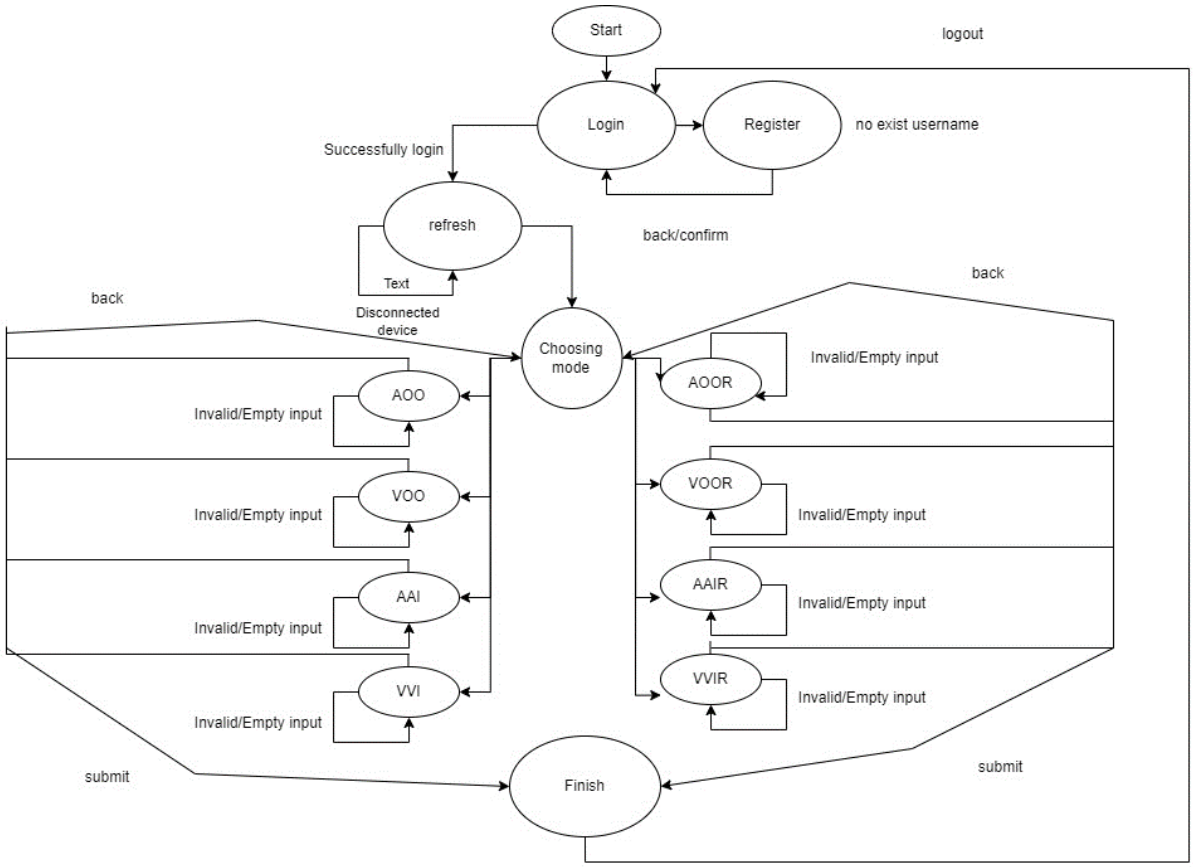

Fig 7. GUI finite state machine

Functions in main.py

def show_login_page():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Remove other pages from view while encapsulating the login page. The purpose is to show only the login page at the beginning.
Description: When a button with this function is pressed, the user will be redirected to the login page, which is implemented by hiding any other page using the syntax:

widget.pack_forget()

The "widget" is the object that developer hopes to hide but not delete, and "pack_forget" is the key word to implement the function.
This function is called by **user_log_out()** function (the user is redirected to the login page when any logout button is pressed), **back_r()** function (the user is redirected to the login page when the back button in register page is pressed), and **main** function (the user only can see the login page when the program starts).
Usage: Directly call the function as follows:

show_login_page()

def save_users():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Write the username and encrypted password to users.txt, the stored data from the last run will be cleared after the new usersname and password has been registered.
Description: After writing the username and password to users.txt, detect whether the username and password stored in the database is more than 10 lines, if it is more than 10 lines, then rewrite the initial 10 lines in the database, and pop-up window to inform the user of the error. If there are no more than 10 lines, a pop-up window will be displayed to inform the user that the registration was successful.
Usage: The function called by the **confirm ()** to save the username and password for users.

def confirm():

Public or Private: Public
Parameters: None
Returns: None
Description: def confirm() is a function that is executed after the confirm button is pressed on the register page. It is used to encrypt the password, save the username and password, and pass the password to the user to see if it was created successfully. We first use .get() to get the values of the username and password entered by the user in the entries of the register page. Next, we check if the length of the username or password is zero, if it is zero, we pop up a window to report an error and return without performing the next operation. If it is not zero,

check whether the username has been registered, if it has been registered, pop-up window will be displayed and return, do not perform the next operation. If both statements match, then we encrypt the password with hashlib and then invoke save_users(), which is the function we wrote earlier. After a successful registration, I invoked the .delete() function to clear the entries to facilitate the next registration. Finally, I close the register and other pages and encapsulate the login page so that the user can go directly to the login page after successful registration. For the source code please refer to the following figure.
Usage: When **command = confirm_button**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

```
def confirm():
    username = new_username.get()
    password = new_password.get()
    if len(username) == 0 or len(password) == 0:
        messagebox.showerror( title: 'Error',  message: 'Username or password cannot be empty.')
        register_page.pack()
        return
    if username in users:
        messagebox.showerror( title: 'Error',  message: 'Username already exists.')
        register_page.pack()
        return

    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    users[username] = hashed_password
    save_users()
    new_username.delete( first_index: 0,  last_index: 'end')
    new_password.delete( first_index: 0,  last_index: 'end')
    register_page.pack_forget()
    mode_page.pack_forget()
    login_page.pack()
```

Fig 8. source code of def confirm()

def registers():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The main function of register() is to retrieve the rest of the page and encapsulate the register page after pressing the "register here" button on the login page.
Description: I'm deleting the contents of the two entries of the previous login page. I have deleted the two entries of the previous login page to ensure the privacy of the user.
Usage: When **command = register_button**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def login():

Public or Private: Public
Parameters: None
Returns: None
Description: def login() is a function that is executed after the login button is pressed on the login page. It decides whether to go to the next page by comparing the user name and encrypted password in the input box with the user name and encrypted password in the database. First it will check whether the length of the username or password entered by the user is zero, if one of them is zero or both of them are zero, then it will pop up an error pop-up window and stay

in the login screen. If the entered username is not in the database, then it will pop up an error message about the username and stay in the current screen. If the password entered does not match the password of the username in the database, the user will be informed that the password is incorrect. After the error is reported, the contents of the entry are cleared to allow the user to enter a new username and password. After no errors have been reported and the password entered matches the password in the database, a login success message is displayed and the user proceeds to the next page. For detailed source code, please refer to the figure below.
Usage: When **command = login_button**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

```
def login():
    username = login_username.get()
    password = login_password.get()
    if len(username) == 0 or len(password) == 0:
        messagebox.showerror( title: 'Error',  message: 'Username or password cannot be empty.')
        login_page.pack()
        return
    if username not in users:
        messagebox.showerror( title: 'Error',  message: 'User does not exist.')
        login_username.delete( first_index: 0,  last_index: 'end')
        login_password.delete( first_index: 0,  last_index: 'end')
        login_page.pack()
        return
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    if users[username] == hashed_password:
        messagebox.showinfo( title: 'Success',  message: 'Login successfully')
        login_page.pack_forget()
        register_page.pack_forget()
        mode_page.pack()
    else:
        messagebox.showerror( title: 'Error',  message: 'Password is not correct.')
        login_password.delete( first_index: 0,  last_index: 'end')
        login_page.pack()
```

Fig 9. login function source code

def clear_entry():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Every time the "**clear_entry()**" is called, the placeholder for user input on the page will be cleared.
Description: This function is used to clear the user's recent input when the user clicks the "back" or "logout" button, preventing confusion that may arise when users revisit the page due to any previous input records. The specific usage example is as follows:

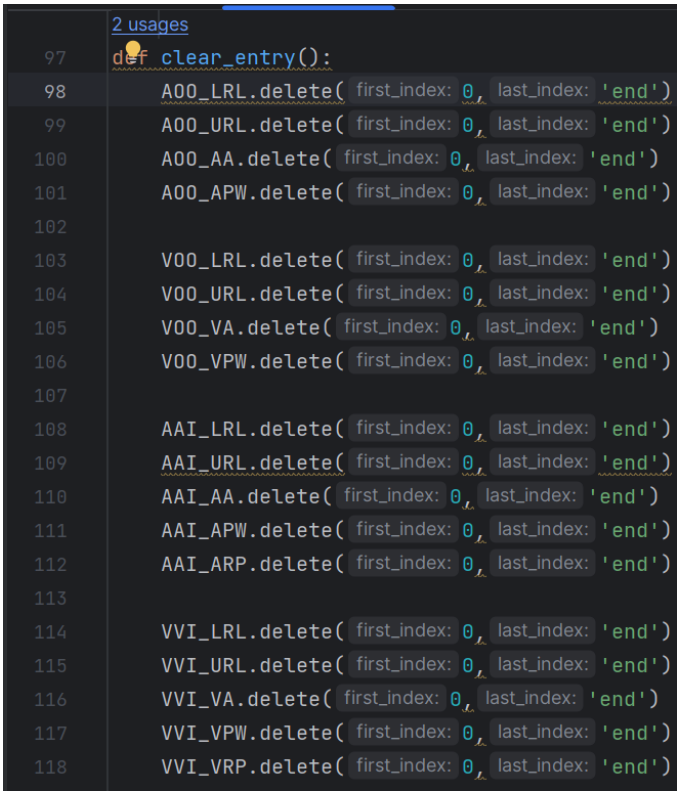

Fig 10. Implementation of clear_entry()

The specific syntax is:

entry_widget.delete(start, end)

In this statement, "entry_widget" is the name of the entry for user input (AOO_LRL, AOO_URL, etc.), "delete" is the key word to use the feature that is included in the standard library, "start" is the first index and "end" is the final index. In this case, all of characters need to be deleted in the placeholder, so the start index is "0" and the final index is "end". This function is called by **user_log_out()** function and **back_button()** function.

Usage: Directly call the function as follows:

clear_entry()

def user_log_out():

Public or Private: Public

Parameters: None

Returns: None

Purpose: Allow the user to return to the login page and clear all historical input records in the placeholder.

Description: Call **show_login_page()** function to back to the login page. Meanwhile, call **clear_entry()** function to delete all input records.

The **user_log_out()** function is called by the **CTkbutton()** command property. Here is a usage example:

```
log_out = customtkinter.CTkButton(master=AOO_page, text="Log out", command=user_log_out)
```

Fig 11. Properties in log_out button

Usage: When **command = user_log_out**, the function is called and executes the contents.

The function is considered as an action, so it is not called by other functions.

def back_r():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Allow users to return to the login page from the register page after pressing “back” button in the register page.
Description: When the “back” button is pressed in the register page, the user would be redirected to the login page by calling the **show_login_page()** function. Furthermore, the context in placeholders would be cleared when the user navigates to the register page again by deleting the function in the standard library.

```
1 usage
def back_r():
    show_login_page()
    new_username.delete( first_index: 0, last_index: 'end')
    new_password.delete( first_index: 0, last_index: 'end')
```

Fig 12. Implementation of back_r()

Usage: When **command = back_r()**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def back_button():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Allow users to return to the previous page from current page. Login page, register page, mode choosing page are not included in the range of “current page”.
Description: When “back” button is pressed in the 4 different mode pages, the user would be redirected to the mode choosing page by hiding any other pages with .pack_forget() function. Then, calling clear_entry() to delete the history records. Finally, illustrate the mode choosing page by **mode_page.pack()**.

```
4 usages
def back_button():
    login_page.pack_forget()
    register_page.pack_forget()
    AOO_page.pack_forget()
    VOO_page.pack_forget()
    AAI_page.pack_forget()
    VVI_page.pack_forget()
    clear_entry()
    mode_page.pack()
```

Fig 13. Implementation of back_button()

Usage: When **command = back_button**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def submit_aoo():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Allow users to submit the input parameters to the pacemaker. Detect if the input parameters are valid. If the input is empty or not a number (except "off" input in the amplitude entry), remind the user to enter valid characters. If the input is not within the expected range of AOO mode, inform the user to input within the specified range that is given in the programmable parameters table. If the input falls within the expected range, inform the user that the submission was successful.
Description: Firstly, the function initialized 3 boolean variables which are "**valid**" (True), "**invalid_input**" (False), and "**out_of_range**" (False).

```
valid = True
invalid_input = False
out_of_range = False
```

Fig 14. Initialization of 3 boolean variables

Then, user inputs are collected for four parameters, "**aoo_lrl**", "**aoo_url**", "**aoo_aa**", "**aoo_apw**". These inputs obtained from the AOO page.
Next, the code proceeds to verify each of these parameters as follows: for the "lower rate limit" ("**aoo_lrl**") and "upper rate limit" ("**aoo_url**"), it attempts to convert the input to a floating-point number. If successful, it checks whether the number is within specific ranges and updates the "**out_of_range**" flag if the input is out of range. If the conversion to a float fails, it updates the "**invalid_input**" flag. Here is an example below which can detect if the input lower rate limit is available:

```
# Verify the lower rate limit parameter
try:
    aoo_lrl = float(aoo_lrl)
    if not (30 <= aoo_lrl <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
# Verify the upper rate limit parameter
try:
    aoo_url = float(aoo_url)
    if not (50 <= aoo_url <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 15. Lower rate limit and upper rate limit check

For "atrial amplitude" ("**aoo_aa**"), it checks if the input is "off" (which is valid). If it's not

"off", it attempts to convert the input to a float and checks if it falls within specific ranges. It updates the "**out_of_range**" flag or "**invalid_input**" flag as appropriate. For "atrial pulse width" ("**aoo_apw**"), it attempts to convert the input to a float and checks whether it falls within specific ranges, updating the "**out_of_range**" flag or "**invalid_input**" flag accordingly. Here is the implementation of this part:

```
# Verify the atrial amplitude
if aoo_aa == 'off':
    # the input can be "off"
    pass
else:
    try:
        aoo_aa = float(aoo_aa)
        if not ((0.5 <= aoo_aa <= 3.2) or (3.5 <= aoo_aa <= 7)):
            # the input is out of range
            out_of_range = True
    except ValueError:
        # the input is neither a number nor "off"
        invalid_input = True
# Verify the atrial pulse width
try:
    aoo_apw = float(aoo_apw)
    if not (aoo_apw == 0.05 or (0.1 <= aoo_apw <= 1.9)):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 16. Atrial amplitude and atrial pulse width check

After verifying all the parameters, the script displays messages to the user based on the flags set: If "**invalid_input**" is "**True**", it shows an error message indicating that the user should enter a valid parameter; If "**out_of_range**" is "**True**", it shows an error message indicating that the input parameter is out of range; If neither of the flags is set (i.e., the inputs are both valid and within the specified ranges), it displays a success message and writes the validated parameters to a file called "**user_inputs.txt**". The implementation of this part is shown below:

```
if invalid_input:
    messagebox.showerror( title: "Error",  message: "Please entry a valid parameter")
elif out_of_range:
    messagebox.showerror( title: "Error",  message: "Input parameter is out of range")
else:
    messagebox.showinfo( title: "Success", message: "Successfully submit!")
    with open('user_inputs.txt','w') as file:
        file.write(f"lower limit rate:{aoo_lrl} \n")
        file.write(f"upper limit rate:{aoo_url} \n")
        file.write(f"atrial amplitude:{aoo_aa} \n")
        file.write(f"atrial pulse width:{aoo_apw} \n")
```

Fig 17. Output of the function

Usage: When **command = submit_aoo**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def submit_voo():

Public or Private: Public

Parameters: None

Returns: None

Purpose: Allow users to submit the input parameters to the pacemaker. Detect if the input parameters is valid. If the input is empty or not a number (except "off" input in the amplitude entry), remind the user to enter valid characters. If the input is not within the expected range of VOO mode, inform the user to input within the specified range that is given in the programmable parameters table. If the input falls within the expected range, inform the user that the submission was successful.

Description: Firstly, the function initialized 3 boolean variables which are "**valid**" (True), "**invalid_input**" (False), and "**out_of_range**" (False).

```
valid = True
invalid_input = False
out_of_range = False
```

Fig 18. Initialization of 3 boolean variables

Then, user inputs are collected for four parameters, "**voo_lrl**", "**voo_url**", "**voo_va**", "**voo_vpw**". These inputs obtained from the VOO page.

Next, the code proceeds to verify each of these parameters as follows: for the "lower rate limit" ("**voo_lrl**") and "upper rate limit" ("**voo_url**"), it attempts to convert the input to a floating-point number. If successful, it checks whether the number is within specific ranges and updates the "**out_of_range**" flag if the input is out of range. If the conversion to a float fails, it updates the "**invalid_input**" flag. Here is the implementation of this part:

```
# Verify the lower rate limit parameter
try:
    voo_lrl = float(voo_lrl)
    if not (30 <= voo_lrl <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
# Verify the upper rate limit parameter
try:
    voo_url = float(voo_url)
    if not (50 <= voo_url <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 19. Lower rate limit and upper rate limit check

For "ventricular amplitude" ("**voo_va**"), it checks if the input is "off" (which is valid). If it's not "off", it attempts to convert the input to a float and checks if it falls within specific ranges. It updates the "**out_of_range**" flag or "**invalid_input**" flag as appropriate. For

"ventricular pulse width" ("**voo_vpw**"), it attempts to convert the input to a float and checks whether it falls within specific ranges, updating the "**out_of_range**" flag or "**invalid_input**" flag accordingly. Here is the implementation of this part:

```
# Verify the ventricular amplitude
if voo_va == 'off':
    # the input can be "off"
    pass
else:
    try:
        voo_va = float(voo_va)
        if not ((0.5 <= voo_va <= 3.2) or (3.5 <= voo_va <= 7)):
            # the input is out of range
            out_of_range = True
    except ValueError:
        # the input is neither a number nor "off"
        invalid_input = True
# Verify the ventricular pulse width
try:
    voo_vpw = float(voo_vpw)
    if not (voo_vpw == 0.05 or (0.1 <= voo_vpw <= 1.9)):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 20. Ventricular amplitude and ventricular pulse width check

After verifying all the parameters, the script displays messages to the user based on the flags set: If "**invalid_input**" is "**True**", it shows an error message indicating that the user should enter a valid parameter; If "**out_of_range**" is "**True**", it shows an error message indicating that the input parameter is out of range; If neither of the flags is set (i.e., the inputs are both valid and within the specified ranges), it displays a success message and writes the validated parameters to a file called "**user_inputs.txt**". The implementation of this part is shown below:

```
if invalid_input:
    messagebox.showerror( title: "Error",  message: "Please entry a valid parameter")
elif out_of_range:
    messagebox.showerror( title: "Error",  message: "Input parameter is out of range")
else:
    messagebox.showinfo( title: "Success", message: "Successfully submit!")
    with open('user_inputs.txt','w') as file:
        file.write(f"lower limit rate:{voo_lrl} \n")
        file.write(f"upper limit rate:{voo_lrl} \n")
        file.write(f"ventricular amplitude:{voo_va} \n")
        file.write(f"ventricular pulse width:{voo_vpw} \n")
```

Fig 21. Output of the function

Usage: When **command = submit_voo**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def submit_aai():
Public or Private: Public
Parameters: None
Returns: None
Purpose: Allow users to submit the input parameters to the pacemaker. Detect that if the input parameters is valid. If the input is empty or not a number (except "off" input in the amplitude entry), remind the user to enter valid characters. If the input is not within the expected range of AAI mode, inform the user to input within the specified range that is given in the programmable parameters table. If the input falls within the expected range, inform the user that the submission was successful.
Description: Firstly, the function initialized 3 boolean variables which are "**valid**" (True), "**invalid_input**" (False), and "**out_of_range**" (False).

```
valid = True
invalid_input = False
out_of_range = False
```

Fig 22. Initialization of 3 boolean variables

Then, user inputs are collected for four parameters, "**aai_lrl**", "**aai_url**", "**aai_aa**", "**aai_apw**", "**aai_arp**". These inputs obtained from the AAI page.
Next, the code proceeds to verify each of these parameters as follows: for the "lower rate limit" ("**aai_lrl**") and "upper rate limit" ("**aai_url**"), it attempts to convert the input to a floating-point number. If successful, it checks whether the number is within specific ranges and updates the "**out_of_range**" flag if the input is out of range. If the conversion to a float fails, it updates the "**invalid_input**" flag. Here is the implementation of this part:

```
# Verify the lower rate limit parameter
try:
    aai_lrl = float(aai_lrl)
    if not (30 <= aai_lrl <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
# Verify the upper rate limit parameter
try:
    aai_url = float(aai_url)
    if not (50 <= aai_url <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 23. Lower rate limit and upper rate limit check

For "atrial amplitude" ("**aai_aa**"), it checks if the input is "off" (which is valid). If it's not "off", it attempts to convert the input to a float and checks if it falls within specific ranges. It updates the "**out_of_range**" flag or "**invalid_input**" flag as appropriate. For "atrial pulse

width" ("**aai_apw**"), it attempts to convert the input to a float and checks whether it falls within specific ranges, updating the "**out_of_range**" flag or "**invalid_input**" flag accordingly. Here is the implementation of this part:

```
# Verify the atrial amplitude
if aai_aa == 'off':
    # the input can be "off"
    pass
else:
    try:
        aai_aa = float(aai_aa)
        if not ((0.5 <= aai_aa <= 3.2) or (3.5 <= aai_aa <= 7)):
            # the input is out of range
            out_of_range = True
    except ValueError:
        # the input is neither a number nor "off"
        invalid_input = True
# Verify the atrial pulse width
try:
    aai_apw = float(aai_apw)
    if not (aai_apw == 0.05 or (0.1 <= aai_apw <= 1.9)):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 24. Atrial amplitude and atrial pulse width check

For "atrial refractory period" ("**aai_arp**"), it attempts to convert the input to a float and checks whether it falls within specific ranges, updating the "**out_of_range**" flag or "**invalid_input**" flag accordingly. Here is the implementation of this part:

```
try:
    aai_arp = float(aai_arp)
    if not(150 <= aai_arp <= 500):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 25. Atrial refractory period check

After verifying all the parameters, the script displays messages to the user based on the flags set: If "**invalid_input**" is "**True**", it shows an error message indicating that the user should enter a valid parameter; If "**out_of_range**" is "**True**", it shows an error message indicating that the input parameter is out of range; If neither of the flags is set (i.e., the inputs are both valid and within the specified ranges), it displays a success message and writes the validated parameters to a file called "**user_inputs.txt**". The implementation of this part is shown

below:

```
if invalid_input:
    messagebox.showerror( title: "Error",  message: "Please entry a valid parameter")
elif out_of_range:
    messagebox.showerror( title: "Error",  message: "Input parameter is out of range")
else:
    messagebox.showinfo( title: "Success", message: "Successfully submit!")
    with open('user_inputs.txt','w') as file:
        file.write(f"lower limit rate:{aai_lrl} \n")
        file.write(f"upper limit rate:{aai_url} \n")
        file.write(f"atrial amplitude:{aai_aa} \n")
        file.write(f"atrial pulse width:{aai_apw} \n")
        file.write(f"atrial refactory period:{aai_arp} \n")
```

Fig 26. Output of the function

Usage: When **command = submit_aai**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def submit_vvi():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Allow users to submit the input parameters to the pacemaker. Detect that if the input parameters is valid. If the input is empty or not a number (except "off" input in the amplitude entry), remind the user to enter valid characters. If the input is not within the expected range of VVI mode, inform the user to input within the specified range that is given in the programmable parameters table. If the input falls within the expected range, inform the user that the submission was successful.
Description: Firstly, the function initialized 3 boolean variables which are "**valid**" (True), "**invalid_input**" (False), and "**out_of_range**" (False).

```
valid = True
invalid_input = False
out_of_range = False
```

Fig 27. Initialization of 3 boolean variables

Then, user inputs are collected for four parameters, "**vvi_lrl**", "**vvi_url**", "**vvi_va**", "**vvi_vpw**", "**vvi_vrp**". These inputs obtained from the AAI page.
Next, the code proceeds to verify each of these parameters as follows: for the "lower rate limit" ("**vvi_lrl**") and "upper rate limit" ("**vvi_url**"), it attempts to convert the input to a floating-point number. If successful, it checks whether the number is within specific ranges and updates the "**out_of_range**" flag if the input is out of range. If the conversion to a float fails, it updates the "**invalid_input**" flag. Here is the implementation of this part:

```
# Verify the lower rate limit parameter
try:
    vvi_lrl = float(vvi_lrl)
    if not (30 <= vvi_lrl <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 28. Lower rate limit and upper rate limit check

For “ventricular amplitude” (“**vvi_va**”), it checks if the input is “off” (which is valid). If it's not “off”, it attempts to convert the input to a float and checks if it falls within specific ranges. It updates the “**out_of_range**” flag or “**invalid_input**” flag as appropriate. For “ventricular pulse width” (“**vvi_vpw**”), it attempts to convert the input to a float and checks whether it falls within specific ranges, updating the “**out_of_range**” flag or “**invalid_input**” flag accordingly. Here is the implementation of this part:

```
# Verify the upper rate limit parameter
try:
    vvi_url = float(vvi_url)
    if not (50 <= vvi_url <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True

# Verify the atrial amplitude
if vvi_va == 'off':
    # the input can be "off"
    pass
else:
    try:
        vvi_va = float(vvi_va)
        if not ((0.5 <= vvi_va <= 3.2) or (3.5 <= vvi_va <= 7)):
            # the input is out of range
            out_of_range = True
    except ValueError:
        # the input is neither a number nor "off"
        invalid_input = True
```

Fig 29. Ventricular amplitude and ventricular pulse width check

For “ventricular refractory period” (“**vvi_vrp**”), it attempts to convert the input to a float and checks whether it falls within specific ranges, updating the “**out_of_range**” flag or “**invalid_input**” flag accordingly. Here is the implementation of this part:

```
try:
    vvi_vrp = float(vvi_vrp)
    if not (150 <= vvi_vrp <= 500):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 30. Ventricular refractory period check

After verifying all the parameters, the script displays messages to the user based on the flags set: If "**invalid_input**" is "**True**", it shows an error message indicating that the user should enter a valid parameter; If "**out_of_range**" is "**True**", it shows an error message indicating that the input parameter is out of range; If neither of the flags is set (i.e., the inputs are both valid and within the specified ranges), it displays a success message and writes the validated parameters to a file called "**user_inputs.txt**". The implementation of this part is shown below:

```
if invalid_input:
    messagebox.showerror( title: "Error",  message: "Please entry a valid parameter")
elif out_of_range:
    messagebox.showerror( title: "Error",  message: "Input parameter is out of range")
else:
    messagebox.showinfo( title: "Success", message: "Successfully submit!")
    with open('user_inputs.txt','w') as file:
        file.write(f"lower limit rate:{vvi_lrl} \n")
        file.write(f"upper limit rate:{vvi_url} \n")
        file.write(f"ventricular amplitude:{vvi_va} \n")
        file.write(f"ventricular pulse width:{vvi_vpw} \n")
        file.write(f"ventricular refactory period:{vvi_vrp} \n")
```

Fig 31. Output of the function

Usage: When **command = submit_vvi**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def submit_aoor():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Allow users to submit the input parameters to the pacemaker. Detect if the input parameters are valid. If the input is empty or not a number (except "off" input in the amplitude entry), remind the user to enter valid characters. If the input is not within the expected range of AOORr mode, inform the user to input within the specified range that is given in the programmable parameters table. If the input falls within the expected range, inform the user that the submission was successful.
Description: Firstly, the function initialized 3 boolean variables which are "**valid**" (True), "**invalid_input**" (False), and "**out_of_range**" (False).

```
valid = True
invalid_input = False
out_of_range = False
```

Fig 32. Initialization of 3 boolean variables

Then, user inputs are collected for following parameters, "**aoor_lrl**", "**aoor_url**", "**aoor_aa**", "**aoor_apw**", "**aoor_msr**", "**aoor_at**", "**aoor_reactionT**", "**aoor_rf**", "**aoor_recoveryT**". These inputs obtained from the AOOR page.
Next, the code proceeds to verify each of these parameters as follows: for the "lower rate

limit" ("**aoor_lrl**"), "upper rate limit" ("**aoor_url**"), "atrial amplitude" ("**aoor_aa**"), "atrial pulse width" ("**aoor_apw**"), "maximum sensor rate" ("**aoor_msr**"), "reaction time" ("**aoor_reactionT**"), "response factor" ("**aoor_rf**"), "recovery time" ("**aoor_recoveryT**"), it attempts to convert the input to a floating-point number. If successful, it checks whether the number is within specific ranges and increment and updates the "**out_of_range**" flag if the input is out of range. If the conversion to a float fails, it updates the "**invalid_input**" flag. Here is the implementation of this part:

```
try:
    aoor_lrl = float(aoor_lrl)
    if (30 <= aoor_lrl <= 50) and (aoor_lrl % 5 != 0):
        # the input is out of range
        out_of_range = True
    elif (50 < aoor_lrl <= 90) and (aoor_lrl %1 != 0):
        out_of_range = True
    elif(90 < aoor_lrl <= 175) and (aoor_lrl %5 !=0):
        out_of_range = True
    elif not(30 <= aoor_lrl <= 175):
        out_of_range =True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 33. aoor_lrl

```
# Verify the Maximum Sensor Rate
try:
    aoor_msr = float(aoor_msr)
    if (50 <= aoor_msr <= 175) and (aoor_msr %5 !=0):
        # the input is out of range
        out_of_range = True
    elif not (50 <= aoor_msr <= 175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 34. aoor_msr

```
try:
    aoor_url = float(aoor_url)
    if (50 <= aoor_url <= 175) and (aoor_url %5 != 0):
        # the input is out of range
        out_of_range = True
    elif not (50 <= aoor_url <= 175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 35. aoor_url

```
# Verify the Reaction Time
try:
    aoor_reactionT = float(aoor_reactionT)
    if (10 <= aoor_reactionT <= 50) and (aoor_reactionT % 10 !=0):
        # the input is out of range
        out_of_range = True
    elif not(10 <= aoor_reactionT <= 50):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 36. aoor_reactionT

```
# Verify the atrial amplitude
try:
    aoor_aa = float(aoor_aa)
    if (0 <= aoor_aa <= 100) and (aoor_aa %2 != 0):
        # the input is out of range
        out_of_range = True
    elif not (0 <=aoor_aa <= 100):
        out_of_range = True
except ValueError:
        # the input is neither a number nor "off"
    invalid_input = True
```

Fig 37. aoor_aa

```
# Verify the Response Factor
try:
    aoor_rf= float(aoor_rf)
    if (1 <= aoor_rf <= 16) and (aoor_rf % 1 != 0):
        # the input is out of range
        out_of_range = True
    elif not(1<= aoor_rf <= 16):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 38. aoor_rf

```
try:
    aoor_apw = float(aoor_apw)
    if (1 <= aoor_apw <= 30) and (aoor_apw % 1 != 0):
        # the input is out of range
        out_of_range = True
    elif not(1 <= aoor_apw <= 30):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 39. aoor_apw

```
# Verify the Recovery Time
try:
    aoor_recoveryT= float(aoor_recoveryT)
    if (2 <= aoor_recoveryT <= 16) and (aoor_recoveryT % 1 !=0):
        # the input is out of range
        out_of_range = True
    elif not (2 <= aoor_recoveryT <= 16):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
# Verify the Recovery Time
if aoor_at not in ["V-Low","Low","Med-Low","Med","Med-High","High","V-high"]:
    invalid_input = True
```

Fig 40. aoor_recoveryT and aoor_at

After verifying all the parameters, the script displays messages to the user based on the flags set: If "**invalid_input**" is "**True**", it shows an error message indicating that the user should

enter a valid parameter; If "**out_of_range**" is "**True**", it shows an error message indicating that the input parameter is out of range; If neither of the flags is set (i.e., the inputs are both valid and within the specified ranges), it displays a success message and writes the validated parameters to a file called "**user_inputs.txt**". The implementation of this part is shown below:

```
else:
    with open('user_inputs.txt','w') as file:
        file.write(f"lower limit rate:{aoor_lrl} \n")
        file.write(f"upper limit rate:{aoor_url} \n")
        file.write(f"atrial amplitude:{aoor_aa} \n")
        file.write(f"atrial pulse width:{aoor_apw} \n")

        file.write(f"maximum sensor rate:{aoor_msr} \n")
        file.write(f"activity threshold:{aoor_at} \n")
        file.write(f"reaction time:{aoor_reactionT} \n")
        file.write(f"response factor:{aoor_rf} \n")
        file.write(f"recovery time:{aoor_recoveryT} \n")
```

Fig 41. Output of the function

Usage: When **command = submit_aoor**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def submit_voor():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Allow users to submit the input parameters to the pacemaker. Detect if the input parameters is valid. If the input is empty or not a number (except "off" input in the amplitude entry), remind the user to enter valid characters. If the input is not within the expected range of VOOR mode, inform the user to input within the specified range that is given in the programmable parameters table. If the input falls within the expected range, inform the user that the submission was successful.
Description: Firstly, the function initialized 3 boolean variables which are "**valid**" (True), "**invalid_input**" (False), and "**out_of_range**" (False).

```
valid = True
invalid_input = False
out_of_range = False
```

Fig 42. Initialization of 3 boolean variables

Then, user inputs are collected for following parameters, "**voor_lrl**", "**voor_url**", "**voor_va**", "**voor_apw**", "**voor_msr**", "**voor_at**", "**voor_reactionT**", "**voor_rf**", "**voor_recoveryT**". These inputs obtained from the VOOR page.
Next, the code proceeds to verify each of these parameters as follows: for the "lower rate limit" ("**voor_lrl**"), "upper rate limit" ("**voor_url**"), "ventricular amplitude" ("**voor_va**"), "ventricular pulse width" ("**voor_vpw**"), "maximum sensor rate" ("**voor_msr**"), "reaction time" ("**voor_reactionT**"), "response factor" ("**voor_rf**"), "recovery time" ("**voor_recoveryT**"), it attempts to convert the input to a floating-point number. If

successful, it checks whether the number is within specific ranges and increment and updates the "**out_of_range**" flag if the input is out of range. If the conversion to a float fails, it updates the "**invalid_input**" flag. Here is the implementation of this part:

```
# Verify the lower rate limit parameter
try:
    voor_lrl = float(voor_lrl)
    if (30 <= voor_lrl <= 50) and (voor_lrl % 5 != 0):
        # the input is out of range
        out_of_range = True
    elif(50 < voor_lrl <= 90) and (voor_lrl %1 != 0):
        out_of_range = True
    elif(90 < voor_lrl <= 175) and (voor_lrl %5 != 0):
        out_of_range = True
    elif not (30 <= voor_lrl <= 175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 43. voor_lrl

```
# Verify the Maximum Sensor Rate
try:
    voor_msr = float(voor_msr)
    if (50 <= voor_msr <= 175) and (voor_msr % 5 !=0):
        # the input is out of range
        out_of_range = True
    elif not (50 <= voor_msr <= 175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 44. voor_msr

```
# Verify the upper rate limit parameter
try:
    voor_url = float(voor_url)
    if (50 <= voor_url <= 175) and (voor_url %5 !=0):
        # the input is out of range
        out_of_range = True
    elif not (50 <= voor_url <= 175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 45. voor_url

```
# Verify the Reaction Time
try:
    voor_reactionT = float(voor_reactionT)
    if (10 <= voor_reactionT <= 50) and (voor_reactionT % 10 != 0):
        # the input is out of range
        out_of_range = True
    elif not (10<=voor_reactionT<=50):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 46. voor_reactionT

```
try:
    voor_va = float(voor_va)
    if  (0 <= voor_va <= 100) and (voor_va % 2 != 0):
        # the input is out of range
        out_of_range = True
    elif not (0 <= voor_va <= 100):
        out_of_range = True
except ValueError:
    # the input is neither a number nor "off"
    invalid_input = True
```

Fig 47. voor_va

```
# Verify the Response Factor
try:
    voor_rf= float(voor_rf)
    if (1 <= voor_rf <= 16) and (voor_rf % 1 != 0):
        # the input is out of range
        out_of_range = True
    elif not (1 <= voor_rf <= 16):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 48. voor_rf

```
# Verify the ventricular pulse width
try:
    voor_vpw = float(voor_vpw)
    if (1 <= voor_vpw <= 30) and  (voor_vpw % 1 != 0):
        # the input is out of range
        out_of_range = True
    elif not (1<=voor_vpw<=30):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 49. voor_vpw

```
# Verify the Recovery Time
try:
    voor_recoveryT= float(voor_recoveryT)
    if (2 <= voor_recoveryT <= 16) and (voor_recoveryT %1 !=0):
        # the input is out of range
        out_of_range = True
    elif not (2<= voor_recoveryT <=16):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
# Verify the Recovery Time
if voor_at not in ["V-Low","Low","Med-Low","Med","Med-High","High","V-high"]:
    invalid_input = True
```

Fig 50. voor_recoveryT and voor_at

After verifying all the parameters, the script displays messages to the user based on the flags set: If "**invalid_input**" is "**True**", it shows an error message indicating that the user should enter a valid parameter; If "**out_of_range**" is "**True**", it shows an error message indicating that the input parameter is out of range; If neither of the flags is set (i.e., the inputs are both

valid and within the specified ranges), it displays a success message and writes the validated parameters to a file called "**user_inputs.txt**". The implementation of this part is shown below:

```
else:
    with open('user_inputs.txt','w') as file:
        file.write(f"lower limit rate:{voor_lrl} \n")
        file.write(f"upper limit rate:{voor_lrl} \n")
        file.write(f"ventricular amplitude:{voor_va} \n")
        file.write(f"ventricular pulse width:{voor_vpw} \n")

        file.write(f"maximum sensor rate:{voor_msr} \n")
        file.write(f"activity threshold:{voor_at} \n")
        file.write(f"reaction time:{voor_reactionT} \n")
        file.write(f"response factor:{voor_rf} \n")
        file.write(f"recovery time:{voor_recoveryT} \n")
```

Fig 51. Output of the function

Usage: When **command = submit_voor**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def submit_aair():

Public or Private: Public
Parameters: None
Returns: None
Purpose: Allow users to submit the input parameters to the pacemaker. Detect that if the input parameters is valid. If the input is empty or not a number (except "off" input in the amplitude entry), remind the user to enter valid characters. If the input is not within the expected range of AAIR mode, inform the user to input within the specified range that is given in the programmable parameters table. If the input falls within the expected range, inform the user that the submission was successful.
Description: Firstly, the function initialized 3 boolean variables which are "**valid**" (True), "**invalid_input**" (False), and "**out_of_range**" (False).

```
valid = True
invalid_input = False
out_of_range = False
```

Fig 52. Initialization of 3 boolean variables

Then, user inputs are collected for following parameters, "**aair_lrl**", "**aair_url**", "**aair_aa**", "**aair_apw**", "**aair_arp**", "**aair_as**", "**aair_pvarp**", "**aair_h**", "**aair_rs**", "**aair_msr**", "**aair_at**", "**aair_reactionT**", "**aair_rf**", "**aair_recoveryT**". These inputs are obtained from the VOOR page. Next, the code proceeds to verify each of these which attempts to convert the input to a floating-point number. If successful, it checks whether the number is within specific ranges and increment and updates the "**out_of_range**" flag if the input is out of range. If the conversion to a float fails, it updates the "**invalid_input**" flag. Here is the implementation of this part:

```
# Verify the lower rate limit parameter
try:
    aair_lrl = float(aair_lrl)
    if (30 <= aair_lrl <= 50) and (aair_lrl %5 !=0):
        # the input is out of range
        out_of_range = True
    elif (50 < aair_lrl <= 90) and (aair_lrl % 1 !=0):
        out_of_range = True
    elif(90 < aair_lrl <= 175) and(aair_lrl % 5 !=0):
        out_of_range = True
    elif not (30<= aair_lrl <=175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 53. aair_lrl

```
#Verify the atrial sensitivity
try:
    aair_as = float(aair_as)
    if (0 <=aair_as<= 100) and (aair_as %2 !=0):
        out_of_range = True
    elif (0 <=aair_as<= 100):
        out_of_range = True
except ValueError:
    invalid_input = True
```

Fig 54. aair_as

```
# Verify the upper rate limit parameter
try:
    aair_url = float(aair_url)
    if (50 <= aair_url <= 175) and (aair_url %5 !=0):
        # the input is out of range
        out_of_range = True
    elif not (50 <= aair_url <= 175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 55. aair_url

```
#Verify the PVARP
try:
    aair_pvarp = float(aair_pvarp)
    if (150<= aair_pvarp <=500 ) and (aair_pvarp %10 !=0):
        out_of_range = True
    elif not (150<= aair_pvarp <=500):
        out_of_range = True
except ValueError:
    invalid_input = True
```

Fig 56. aair_pvarp

```
# Verify the atrial amplitude
try:
    aair_aa = float(aair_aa)
    if (0 <= aair_aa <= 100) and (aair_aa % 2 != 0):
            # the input is out of range
        out_of_range = True
    elif not (0 <= aair_aa <= 100):
        out_of_range = True
except ValueError:
        # the input is neither a number nor "off"
    invalid_input = True
```

Fig 57. aair_aa

```
else:
    try:
        aair_h = float(aair_h)
        if not (((30<=aair_lrl<=50)and(30<=aair_h<=50)) or ((50<=aair_lrl<=90)and(50<=aair_h<=90)) or ((90<=aair_lrl<=175)and(90<=aair_
            # the input is out of range
            out_of_range = True
    except ValueError:
        # the input is neither a number nor "off"
        invalid_input = True
#Verify the Rate Smoothing
if aair_rs == 'off':
    pass
else:
    try:
        aair_rs = float(aair_rs)
        if not(aair_rs == 3.0 or aair_rs == 6.0 or aair_rs == 9.0 or aair_rs == 12.0 or aair_rs == 15.0 or aair_rs == 18.0 or aair_rs
            out_of_range = True
    except ValueError:
        invalid_input = True
```

Fig 58. aair_h and aair_rs

```
# Verify the atrial pulse width
try:
    aair_apw = float(aair_apw)
    if (1 <= aair_apw <= 30) and (aair_apw %1 != 0):
        # the input is out of range
        out_of_range = True
    elif not (1 <= aair_apw <= 30):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 59. aair_apw

```
# Verify the Maximum Sensor Rate
try:
    aair_msr = float(aair_msr)
    if (50 <= aair_msr <= 175) and (aair_msr %5 !=0):
        # the input is out of range
        out_of_range = True
    elif not (50 <= aair_msr <= 175):
        out_of_range =True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 60. aair_msr

```
try:
    aair_arp = float(aair_arp)
    if (150 <= aair_arp <= 500) and (aair_arp %10 != 0):
        # the input is out of range
        out_of_range = True
    elif not (150 <= aair_arp <= 500):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 61. aair_arp

```
# Verify the Reaction Time
try:
    aair_reactionT = float(aair_reactionT)
    if  (10 <= aair_reactionT <= 50) and (aair_reactionT %10 !=0):
        # the input is out of range
        out_of_range = True
    elif not (10 <= aair_reactionT <= 50):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 62. aair_reactionT

```
# Verify the Response Factor
try:
    aair_rf= float(aair_rf)
    if (1 <= aair_rf <= 16) and (aair_rf %1 != 0):
        # the input is out of range
        out_of_range = True
    elif not (1<=aair_rf<=16):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 63. aair_rf

```
# Verify the Recovery Time
try:
    aair_recoveryT= float(aair_recoveryT)
    if  (2 <= aair_recoveryT <= 16) and (aair_recoveryT %1 !=0):
        # the input is out of range
        out_of_range = True
    elif not (2 <= aair_recoveryT <= 16):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
# Verify the Recovery Time
if aair_at not in ["V-Low","Low","Med-Low","Med","Med-High","High","V-high"]:
    invalid_input = True
```

Fig 64. aair_recoveryT and aair_at

After verifying all the parameters, the script displays messages to the user based on the flags set: If "**invalid_input**" is "**True**", it shows an error message indicating that the user should enter a valid parameter; If "**out_of_range**" is "**True**", it shows an error message indicating that the input parameter is out of range; If neither of the flags is set (i.e., the inputs are both valid and within the specified ranges), it displays a success message and writes the validated parameters to a file called "**user_inputs.txt**". The implementation of this part is shown below:

```
else:
    with open('user_inputs.txt','w') as file:
        file.write(f"lower limit rate:{aair_lrl} \n")
        file.write(f"upper limit rate:{aair_url} \n")
        file.write(f"atrial amplitude:{aair_aa} \n")
        file.write(f"atrial pulse width:{aair_apw} \n")
        file.write(f"atrial refactory period:{aair_arp} \n")
        file.write(f"atrial sensitivity:{aair_as} \n")
        file.write(f"PVARP:{aair_pvarp} \n")
        file.write(f"Hysteresis:{aair_h} \n")
        file.write(f"Rate smoothing:{aair_rs} \n")

        file.write(f"maximum sensor rate:{aair_msr} \n")
        file.write(f"activity threshold:{aair_at} \n")
        file.write(f"reaction time:{aair_reactionT} \n")
        file.write(f"response factor:{aair_rf} \n")
        file.write(f"recovery time:{aair_recoveryT} \n")
```

Fig 65. Output of the function

Usage: When **command = submit_aair**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def submit_vvir():

Public or Private: Public

Parameters: None

Returns: None

Purpose: Allow users to submit the input parameters to the pacemaker. Detect if the input parameters is valid. If the input is empty or not a number (except "off" input in the amplitude entry), remind the user to enter valid characters. If the input is not within the expected range of VVIR mode, inform the user to input within the specified range that is given in the programmable parameters table. If the input falls within the expected range, inform the user that the submission was successful.

Description: Firstly, the function initialized 3 boolean variables which are "**valid**" (True), "**invalid_input**" (False), and "**out_of_range**" (False).

```
valid = True
invalid_input = False
out_of_range = False
```

Fig 66. Initialization of 3 boolean variables

Then, user inputs are collected for following parameters, "**vvir_lrl**", "**vvir_url**", "**vvir_va**", "**vvir_vpw**", "**vvir_vrp**", "**vvir_vs**", "**vvir_h**", "**vvir_rs**", "**vvir_msr**", "**vvir_at**", "**vvir_reactionT**", "**vvir_rf**", "**vvir_recoveryT**". These inputs are obtained from the VOOR page. Next, the code proceeds to verify each of these which attempts to convert the input to a floating-point number. If successful, it checks whether the number is within specific ranges and increment and updates the "**out_of_range**" flag if the input is out of range. If the conversion to a float fails, it updates the "**invalid_input**" flag. Here is the implementation of this part:

```
# Verify the lower rate limit parameter
try:
    vvir_lrl = float(vvir_lrl)
    if (30 <= vvir_lrl <= 50) and (vvir_lrl %5 != 0):
        # the input is out of range
        out_of_range = True
    elif (50< vvir_lrl <= 90) and (vvir_lrl %1!=0):
        out_of_range = True
    elif (90<vvir_lrl<= 175) and (vvir_lrl %5 !=0):
        out_of_range = True
    elif not (30 <= vvir_lrl <= 175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 67. vvir_lrl

```
# Verify the atrial pulse width
try:
    vvir_vpw = float(vvir_vpw)
    if (1 <= vvir_vpw <= 30) and (vvir_vpw %1 != 0):
        # the input is out of range
        out_of_range = True
    elif not (1<= vvir_vpw <= 30):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 68. vvir_vpw

```
# Verify the upper rate limit parameter
try:
    vvir_url = float(vvir_url)
    if (50 <= vvir_url <= 175) and (vvir_url %5 !=0):
        # the input is out of range
        out_of_range = True
    elif not (50 <= vvir_url <= 175):
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 69. vvir_url

```
try:
    vvir_vrp = float(vvir_vrp)
    if (150 <= vvir_vrp <= 500) and (vvir_vrp%10 != 0):
        # the input is out of range
        out_of_range = True
    elif not (150 <= vvir_vrp <= 500):
        out_of_range =True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 70. vvir_vrp

```
# Verify the atrial amplitude
try:
    vvir_va = float(vvir_va)
    if (0 <= vvir_va <= 100) and (vvir_va %2 !=0):
            # the input is out of range
        out_of_range = True
    elif not (0 <= vvir_va <= 100):
        out_of_range = True
except ValueError:
        # the input is neither a number nor "off"
    invalid_input = True
```

Fig 71. vvir_va

```
#Verify the ventricular sensitivity
try:
    vvir_vs = float(vvir_vs)
    if (0<=vvir_vs<=100) and (vvir_vs %2 !=0):
        out_of_range = True
    elif not (0<= vvir_vs <= 100):
        out_of_range = True
except ValueError:
    invalid_input = True
```

Fig 72. vvir_vs

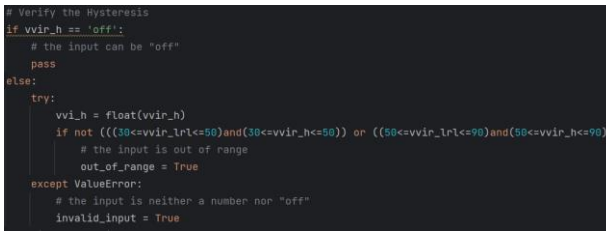

Fig 73. vvir_h

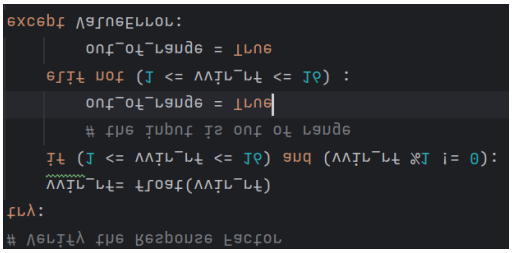

Fig 74. vvir_rf

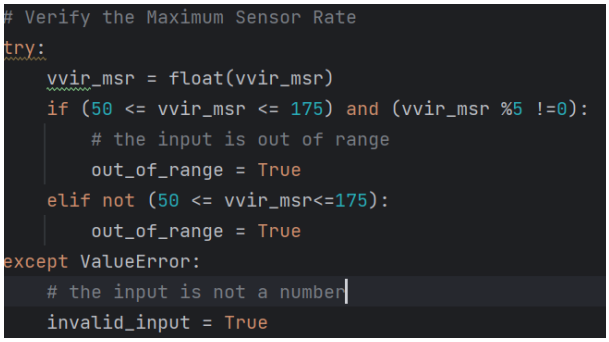

Fig 75. vvir_msr

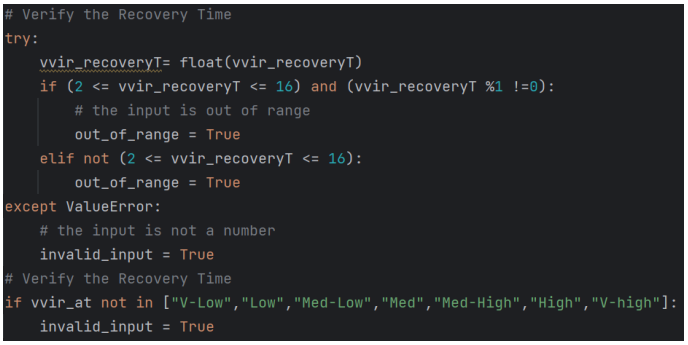

Fig 76. vvir_recoveryT

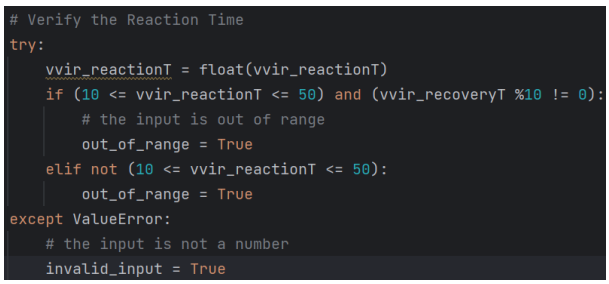

Fig 77. vvir_reactionT

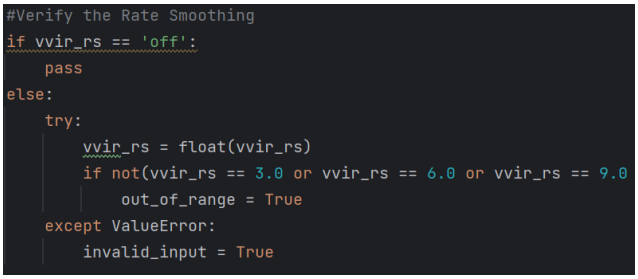

Fig 78. vvir_rs

After verifying all the parameters, the script displays messages to the user based on the flags set: If "**invalid_input**" is "**True**", it shows an error message indicating that the user should enter a valid parameter; If "**out_of_range**" is "**True**", it shows an error message indicating that the input parameter is out of range; If neither of the flags is set (i.e., the inputs are both valid and within the specified ranges), it displays a success message and writes the validated parameters to a file called "**user_inputs.txt**". The implementation of this part is shown below:

```
else:
    with open('user_inputs.txt','w') as file:
        file.write(f"lower limit rate:{vvir_lrl} \n")
        file.write(f"upper limit rate:{vvir_url} \n")
        file.write(f"ventricular amplitude:{vvir_va} \n")
        file.write(f"ventricular pulse width:{vvir_vpw} \n")
        file.write(f"ventricular refactory period:{vvir_vrp} \n")
        file.write(f"ventricular sensitivity:{vvir_vs} \n")
        file.write(f"hysteresis:{vvir_h} \n")
        file.write(f"rate smoothing:{vvvir_rs} \n")

        file.write(f"maximum sensor rate:{vvir_msr} \n")
        file.write(f"activity threshold:{vvir_at} \n")
        file.write(f"reaction time:{vvir_reactionT} \n")
        file.write(f"response factor:{vvir_rf} \n")
        file.write(f"recovery time:{vvir_recoveryT} \n")
```

Fig 79. Output of the function

Usage: When **command = submit_vvir**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def AOO():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The webpage navigates to the AOO mode page from mode choosing page after pressing the "AOO" button.
Description: Close previous pages (login page, mode choosing page, register page) by using the syntax as follows:

widget.pack_forget()

The prefix "widget" means the component (label, button, entry, etc.) in **customtkinter**, and the component in this case means a packed webpage. Therefore, the **.pack_forget** can hide the webpage but not delete them. To illustrate the AOO mode page, the following pages must be hidden: login page, register page, mode choosing page. The details of the function are presented below:

```
def AOO():
    login_page.pack_forget()
    mode_page.pack_forget()
    register_page.forget()
    AOO_page.pack()
```

Fig 80. Implementation of AOO()

Usage: When **command = AOO**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def VOO():

Public or Private: Public
Parameters: None
Returns: None

Purpose: The webpage navigates to the VOO mode page from mode choosing page after pressing the “VOO” button.
Description: Close previous pages (login page, mode choosing page, register page) by using the syntax as follows:

widget.pack_forget()

The prefix “widget” means the component (label, button, entry, etc.) in **customtkinter**, and the component in this case means a packed webpage. Therefore, the **.pack_forget** can hide the webpage but not delete them. To illustrate the VOO mode page, the following pages must be hidden: login page, register page, mode choosing page. The details of the function are presented below:

```
def VOO():
    login_page.pack_forget()
    mode_page.pack_forget()
    register_page.forget()
    VOO_page.pack()
```

Fig 81. Implementation of VOO()

Usage: When **command = VOO**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def AAI():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The webpage navigates to the AAI mode page from mode choosing page after pressing the “AAI” button.
Description: Close previous pages (login page, mode choosing page, register page) by using the syntax as follows:

widget.pack_forget()

The prefix “widget” means the component (label, button, entry, etc.) in **customtkinter**, and the component in this case means a packed webpage. Therefore, the **.pack_forget** can hide the webpage but not delete them. To illustrate the AAI mode page, the following pages must be hidden: login page, register page, mode choosing page. The details of the function are presented below:

```
def AAI():
    login_page.pack_forget()
    mode_page.pack_forget()
    register_page.forget()
    AAI_page.pack()
```

Fig 82. Implementation of AAI()

Usage: When **command = AAI**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def VVI():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The webpage navigates to the VVI mode page from mode choosing page after pressing the "VVI" button.
Description: Close previous pages (login page, mode choosing page, register page) by using the syntax as follows:

widget.pack_forget()

The prefix "widget" means the component (label, button, entry, etc.) in **customtkinter**, and the component in this case means a packed webpage. Therefore, the **.pack_forget** can hide the webpage but not delete them. To illustrate the VVI mode page, the following pages must be hidden: login page, register page, mode choosing page. The details of the function are presented below:

```
def VVI():
    login_page.pack_forget()
    mode_page.pack_forget()
    register_page.forget()
    VVI_page.pack()
```

Fig 83. Implementation of VVI()

Usage: When **command = VVI**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def AOOR():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The webpage navigates to the AOOR mode page from mode choosing page after pressing the "AOOR" button.
Description: Close previous pages (login page, mode choosing page, register page) by using the syntax as follows:

widget.pack_forget()

The prefix "widget" means the component (label, button, entry, etc.) in **customtkinter**, and the component in this case means a packed webpage. Therefore, the **.pack_forget** can hide the webpage but not delete them. To illustrate the AOOR mode page, the following pages must be hidden: login page, register page, mode choosing page. The details of the function are presented below:

```
1198    def AOOR():
1199        login_page.pack_forget()
1200        mode_page.pack_forget()
1201        register_page.forget()
1202        AOOR_page.pack()
```

Fig 84. Implementation of AOOR()

Usage: When **command = AOOR**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def VOOR():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The webpage navigates to the VOOR mode page from mode choosing page after pressing the "VOOR" button.
Description: Close previous pages (login page, mode choosing page, register page) by using the syntax as follows:

widget.pack_forget()

The prefix "widget" means the component (label, button, entry, etc.) in **customtkinter**, and the component in this case means a packed webpage. Therefore, the **.pack_forget** can hide the webpage but not delete them. To illustrate the VOOR mode page, the following pages must be hidden: login page, register page, mode choosing page. The details of the function are presented below:

```
def VOOR():
    login_page.pack_forget()
    mode_page.pack_forget()
    register_page.forget()
    VOOR_page.pack()
```

Fig 85. Implementation of VOOR()

Usage: When **command = VOOR**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def AAIR():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The webpage navigates to the AAIR mode page from mode choosing page after pressing the "AAIR" button.
Description: Close previous pages (login page, mode choosing page, register page) by using

the syntax as follows:

widget.pack_forget()

The prefix “widget” means the component (label, button, entry, etc.) in **customtkinter**, and the component in this case means a packed webpage. Therefore, the **.pack_forget** can hide the webpage but not delete them. To illustrate the AAIR mode page, the following pages must be hidden: login page, register page, mode choosing page. The details of the function are presented below:

```
def AAIR():
    login_page.pack_forget()
    mode_page.pack_forget()
    register_page.forget()
    AAIR_page.pack()
```

Fig 86. Implementation of AAIR()

Usage: When **command = AAIR**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def VVIR():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The webpage navigates to the VVIR mode page from mode choosing page after pressing the “VVIR” button.
Description: Close previous pages (login page, mode choosing page, register page) by using the syntax as follows:

widget.pack_forget()

The prefix “widget” means the component (label, button, entry, etc.) in **customtkinter**, and the component in this case means a packed webpage. Therefore, the **.pack_forget** can hide the webpage but not delete them. To illustrate the VVIR mode page, the following pages must be hidden: login page, register page, mode choosing page. The details of the function are presented below:

```
def VVIR():
    login_page.pack_forget()
    mode_page.pack_forget()
    register_page.forget()
    VVIR_page.pack()
```

Fig 87. Implementation of VVIR()

Usage: When **command = VVIR**, the function is called and executes the contents. The

function is considered as an action, so it is not called by other functions.

def refresh ():

Public or Private: Public
Parameters: None
Returns: Boolean
Purpose: The purpose of this function is to detect whether the pacemaker is connected to our DCM or not.
Description: We put this function into the submit function of each mode. When the user clicks the submit button and no error is reported, the refresh function in the submit function will detect if there is a device accessing the DCM, if so, it will prompt the user that the two devices are transmitting data after the data is written. If there is no pacemaker connected to the DCM, it will tell the user that the port is closed. The refresh function uses python's serial library, which can detect if there is a device connected to a certain port on the computer, my pacemaker is connected to the COM3.
Usage: Called by function submit_aoo (), submit_voo (), submit_aai (), submit_vvi () to check if the pacemaker is connected to DCM.

def check ():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The purpose of this function is to detect whether the pacemaker has changed
Description: We put this on the mode page to check if the pacemaker has been changed. Due to the reason we can interact with the pacemaker, we can't get the value of usb devices' vendor and product IDs. In the future, we plan to use the pyusb library to detect the specific devices' IDs.
Usage: When **command = check**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def plot_1 ():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The purpose of this function is to build an interface for the Atrium Plot.
Description: We put this on the mode page to form an interface for the atrium plot if the Atrial Button is pressed.
Usage: When **command = plot_1**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

def plot_2 ():

Public or Private: Public
Parameters: None
Returns: None

Purpose: The purpose of this function is to build an interface for the Ventricle Plot.
Description: We put this on the mode page to form an interface for the ventricle plot if the Ventricular Button is pressed.
Usage: When **command = plot_2**, the function is called and executes the contents. The function is considered as an action, so it is not called by other functions.

Functions in DCM_serial.py

def input ():

Public or Private: Public
Parameters: lrl, url, amplitude, pw, rp, factor_input, threshold, mode
Returns: None
Purpose: The purpose of this function is to transfer the user inputs to the board.
Description: The function takes the user inputs, constructs them in a byte signal, and transfer the data to the board through serial.
Usage: When **command = submit**, the submit function is called. If the parameter inputs fulfill the range requirement and are all valid the input () function will be called and transmit the parameter in byte sequence.

def receive ():

Public or Private: Public
Parameters: None
Returns: ATR_signal, VENT_signal
Purpose: The purpose of this function is to receive the outputs from the board.
Description: This function first defines some starting sequence of bytes and then constructs a signal for sending to the connected serial device. Next, it reads 16 bytes of data through the serial port and unpacks it into two values of type double, denoting ATR_signal and VENT_signal, and finally returns these two values as the result.
Usage: When **command = plot_1/plot_2**, the plot_1/plot_2 function is called. Then the create_real_time_dual_graphs_1/2 function is called. After that, the class RealTimeDualGraphs will be called which will call the function update_plot. Finally, the function receive is called.

Functions in Display_graph.py

class RealTimeDualGraphs ():

Public or Private: Public
Parameters: None
Returns: None
Purpose: The purpose of this class is to create a matplotlib-based real-time dynamic chart for displaying a graphical interface of atrial signal amplitude over time.

Description: Sets the window title to "Atrium Graph". Initializes the data list self.x (time axis) and self.y1 (atrial signal amplitude). Creates a graph using matplotlib and sets its horizontal axis to time (in milliseconds). Creates a line in the chart to represent the atrial signal amplitude. Plots the chart onto tkinter's canvas. Uses the master.after function to set a timer to call the update_plot method after every 100 milliseconds.
Usage: When **command = plot_1/plot_2**, the plot_1/plot_2 function is called. Then the create_real_time_dual_graphs_1/2 function is called.

Tests and Results

Login Page

Successful Login

Overview: Login using a username and password which have been successfully registered and stored in the username database.
Expected behavior: Pop up a prompt window displaying the text “Successfully login”. Then, redirect to the mode choosing page after clicking the button “Confirm”.
Result: Success

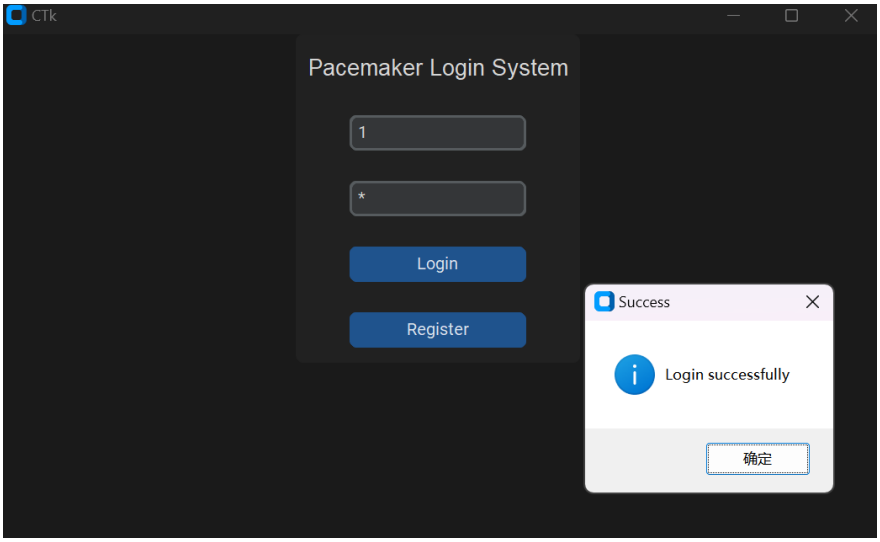

Fig 88. Login successfully test

Fig 89. Mode choosing page

Unsuccessful Login

Overview: Login using a username and password, but the personal information does not exist in the database

Expected behavior: Pop up a prompt window displaying the text “User does not exist”. Click “confirm” to close the prompt window.

Result: Success.

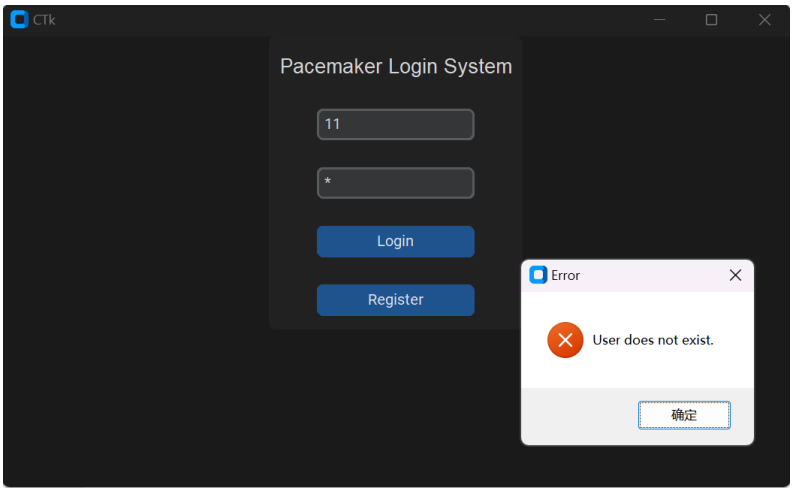

Fig 90. Login unsuccessfully test

Empty Input

Overview: Keep the placeholders empty.

Expected behavior: Pop up a prompt window displaying the text “Username or password cannot be empty”. Click “confirm” to close the prompt window.

Result: Success.

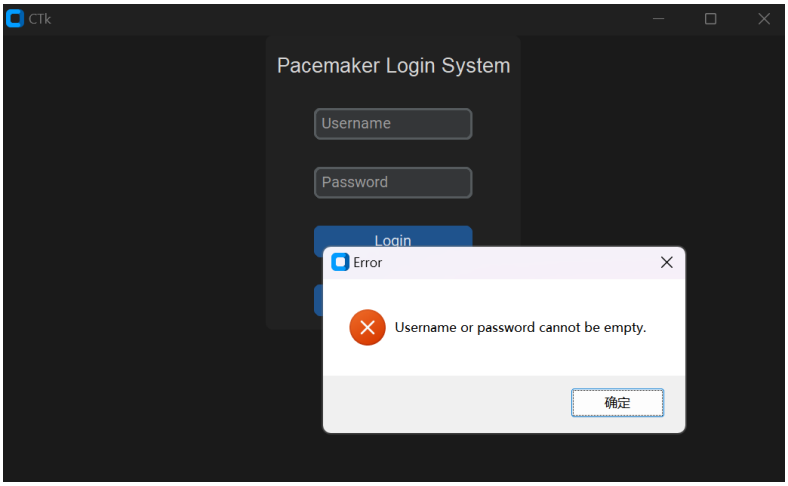

Fig 91. Empty input test

Input contains space

Overview: The user entered username or password contains space.

Expected behavior: Pop up a prompt window displaying the text “Username or password cannot contain space”. Click “ok” to close the prompt window.

Result: Success.

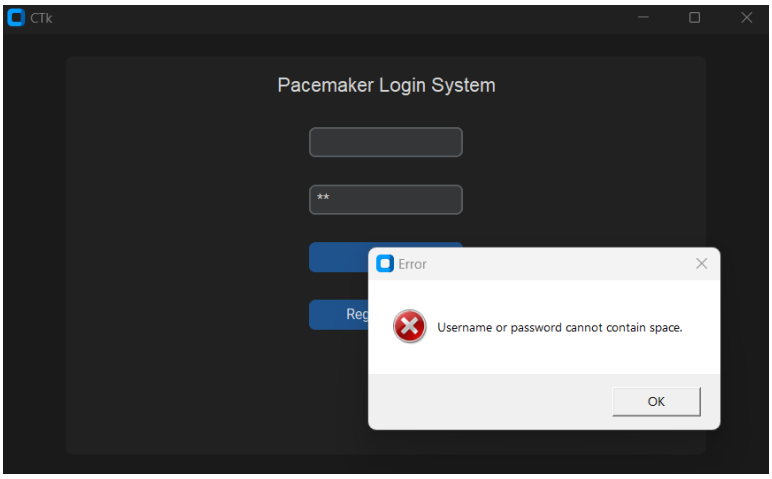

Fig 92. input contains space test

Register Page

Successful Register

Overview: Register by creating a new username and password to store in the username database.

Expected behavior: Pop up a prompt window displaying the text “Account has been created successfully”. Then, redirect to the *login page* after clicking the button “Confirm”.

Result: Success

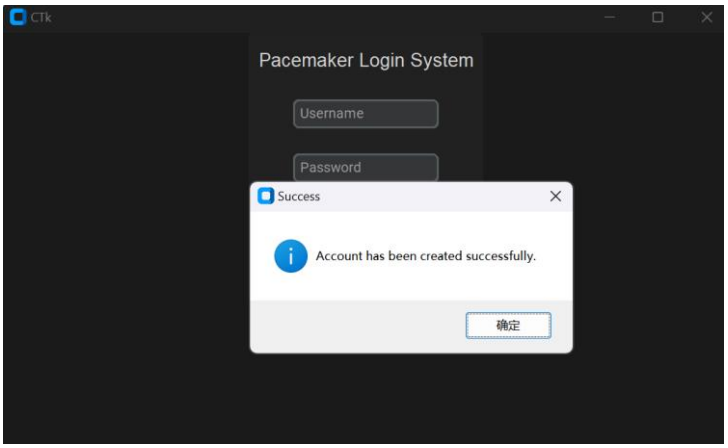

Fig 93. Register successfully test

Unsuccessful Register

Overview: Register by creating an existing username and password.

Expected behavior: Pop up a prompt window displaying the text “Username already exists”. Click “confirm” to close the prompt window.

Result: Success

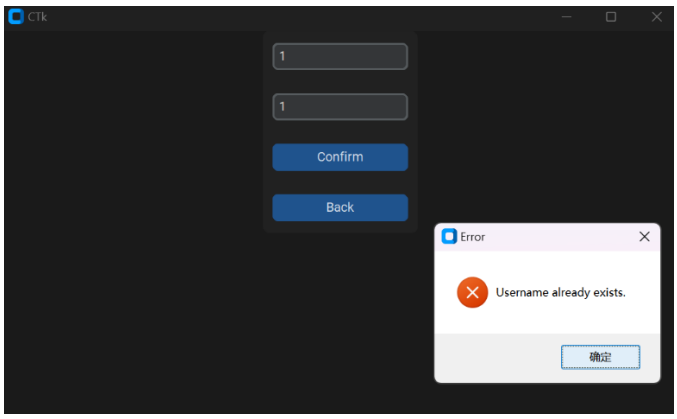

Fig 94. Register unsuccessfully test

Empty Input

Overview: Keep the placeholders empty.

Expected behavior: Pop up a prompt window displaying the text “Username or password cannot be empty”. Click “confirm” to close the prompt window.

Result: Success.

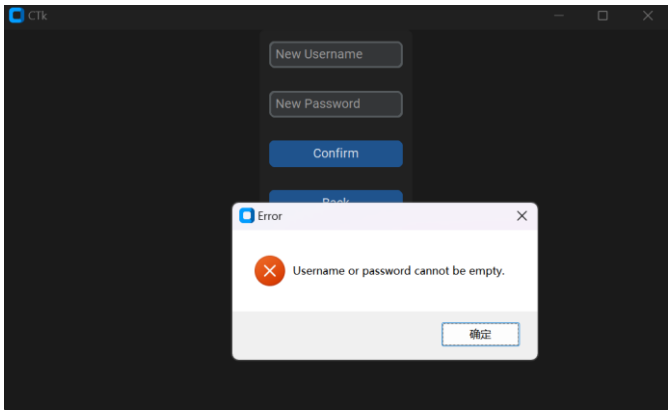

Fig 95. Empty input test

Input contains space

Overview: The user entered username or password contains space.

Expected behavior: Pop up a prompt window displaying the text “Username or password cannot contain space”. Click “ok” to close the prompt window.

Result: Success.

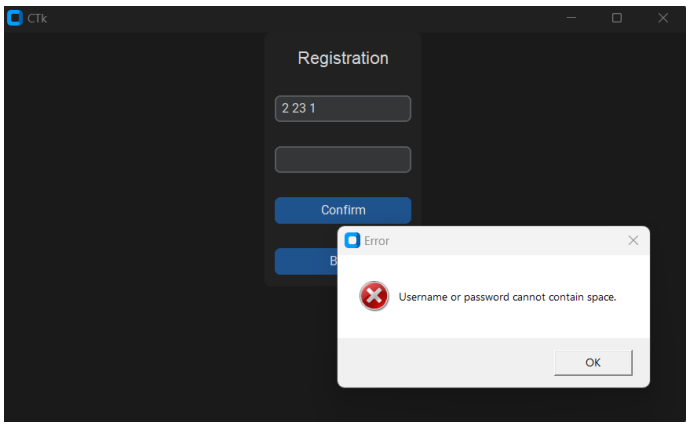

Fig 96. Register inputs contains space

Mode Page

Overview: The page allows users to select the different modes that are appropriate to the personal symptom.

Fig 97. Mode choosing page

AOO page

Overview: Allow users to go to AOO mode.
Expected behavior: Click the button with text “AOO” and redirect to the *AOO page*.
Result: Success

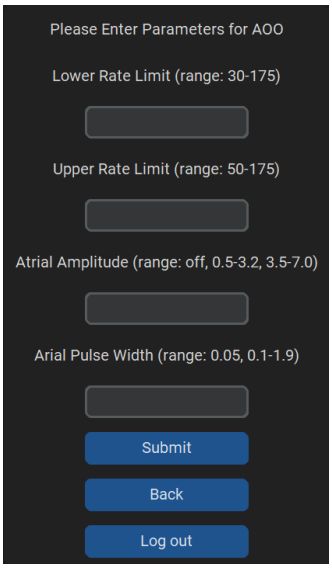

Fig 98. AOO page

VOO page

Overview: Allow users to go to VOO mode.
Expected behavior: Click the button with text “VOO” and redirect to the *VOO page*.
Result: Success

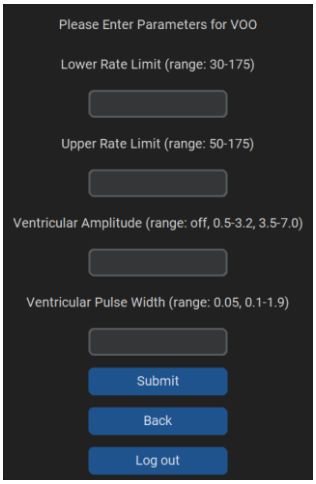

Fig 99. VOO page

AAI page

Overview: Allow users to go to AAI mode.
Expected behavior: Click the button with text “AAI” and redirect to the *AAI page*.
Result: Success

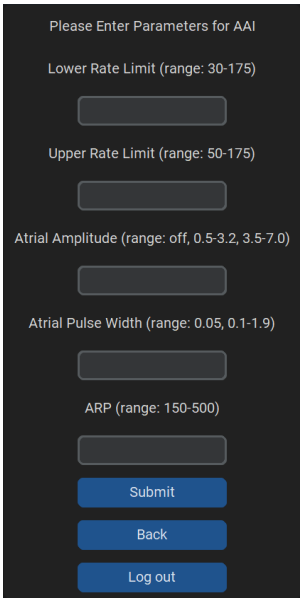

Fig 100. AAI page

VVI page

Overview: Allow users to go to VVI mode.
Expected behavior: Click the button with text “VVI” and redirect to the *VVI page*.
Result: Success

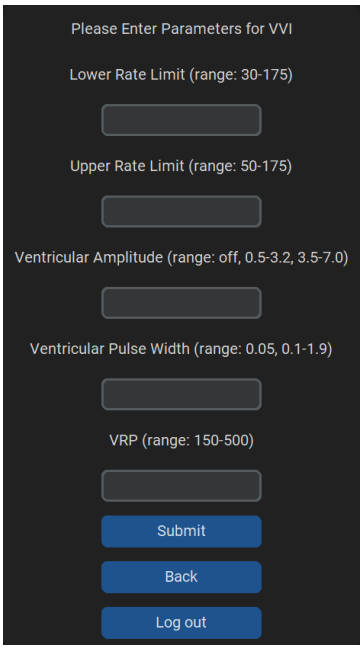

Fig 101. VVI page

AOOR page

Overview: Allow users to go to AOOR mode.
Expected behavior: Click the button with text "AOOR" and redirect to the *AOOR page*.
Result: Success

Fig 102. AOOR page

VOOR page

Overview: Allow users to go to VOOR mode.
Expected behavior: Click the button with text "VOOR" and redirect to the *VOOR page*.
Result: Success

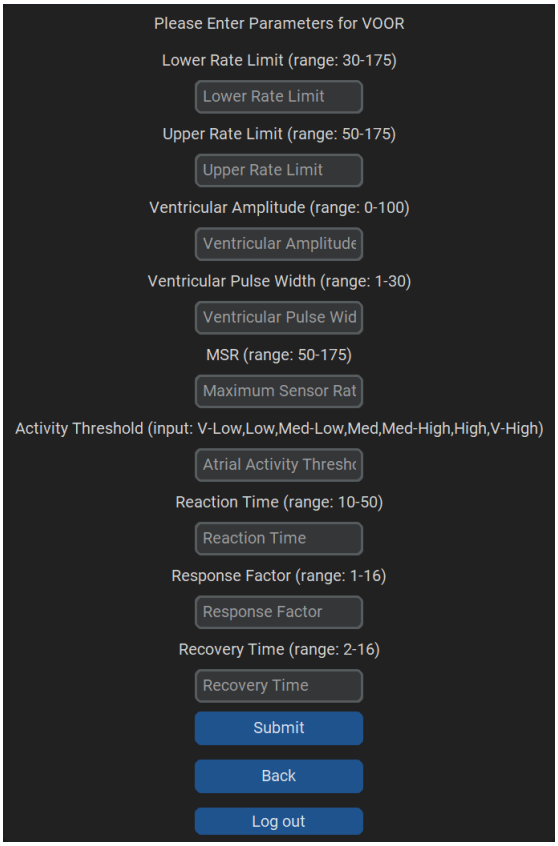

Fig 103. AOOR page

AAIR page

Overview: Allow users to go to AAIR mode.
Expected behavior: Click the button with text "AAIR" and redirect to the *AAIR page*.
Result: Success

Fig 104. AAIR page

VVIR page

Overview: Allow users to go to VVIR mode.

Expected behavior: Click the button with text “VVIR” and redirect to the *VVIR page*.

Result: Success

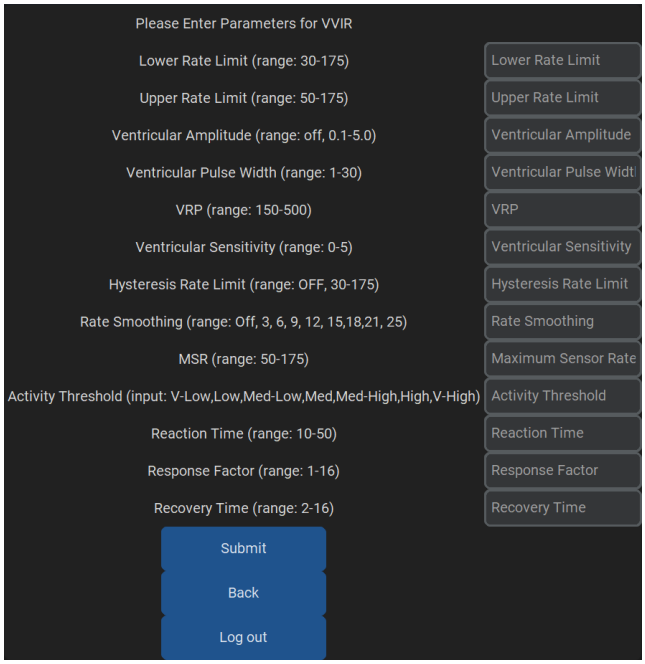

Fig 105. VVIR page

Check Device

Overview: Checking if the device is changed or not.

Expected behavior: Click the button with text “Check Device” and prompt a window to show the result.

Result: Success

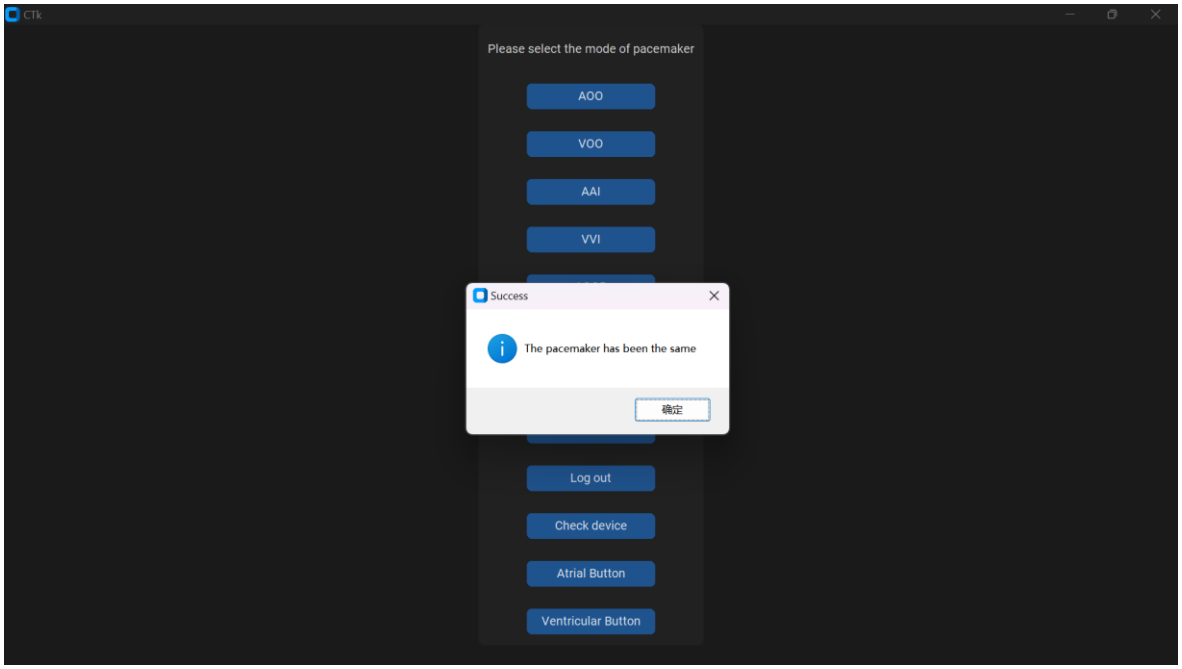

Fig 106. Checking Device

Atrial Button

Overview: Illustrate the egram which corresponds to the submitted mode and parameters.
Expected behavior: Click the button with text “Atrial Button” and prompt a window to show the graph. Can be turned on and off independently from the Ventricular Button.
Result: Success

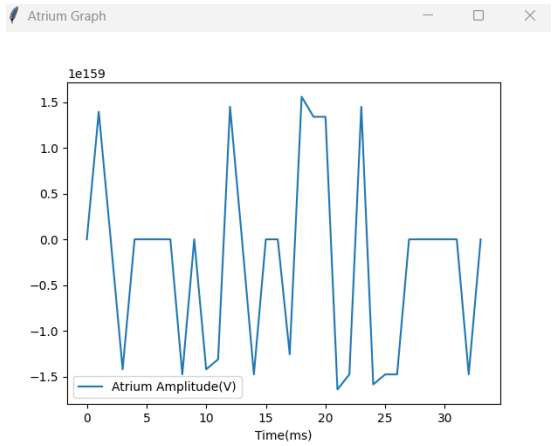

Fig 107. AOOR Atrium Graph

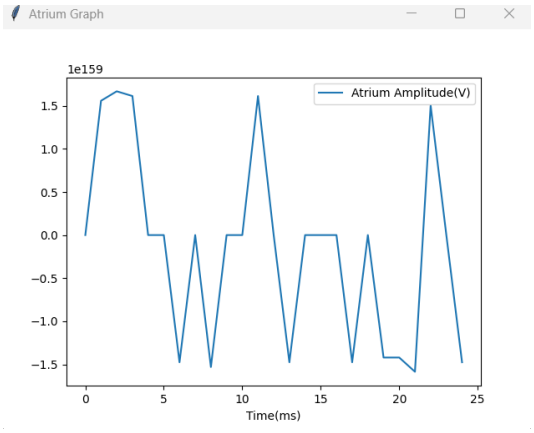

Fig 108. AAIR Atrium Graph

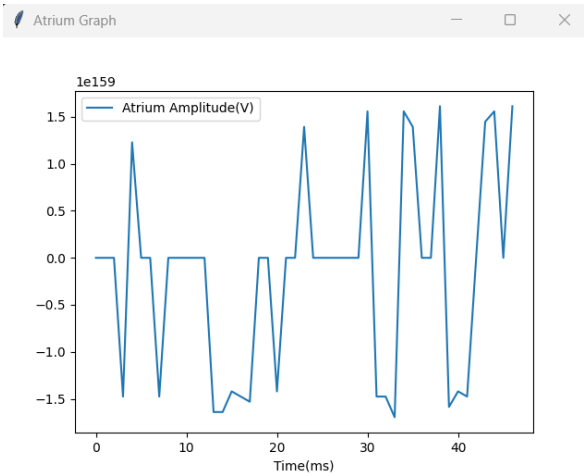

Fig 109. VOOR Atrium Graph

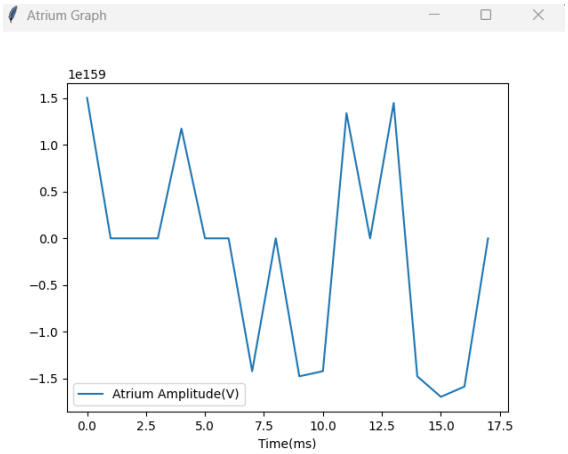

Fig 110. VVIR Atrium Graph

Ventricular Button

Overview: Illustrate the egram which corresponds to the submitted mode and parameters.
Expected behavior: Click the button with text “Ventricular Button” and prompt a window to show the graph. Can be turned on and off independently from the Atrial Button.
Result: Success

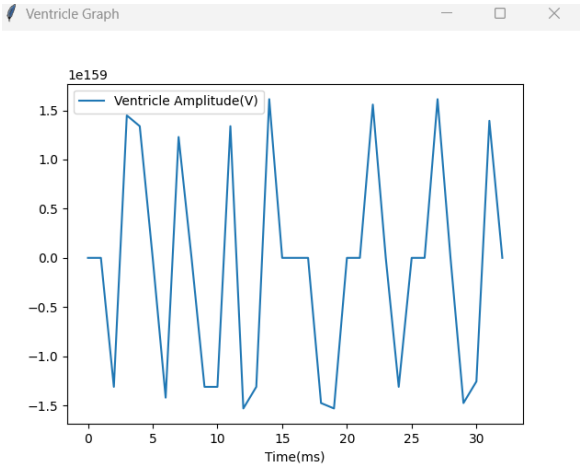

Fig 111. AOOR Ventricle Graph

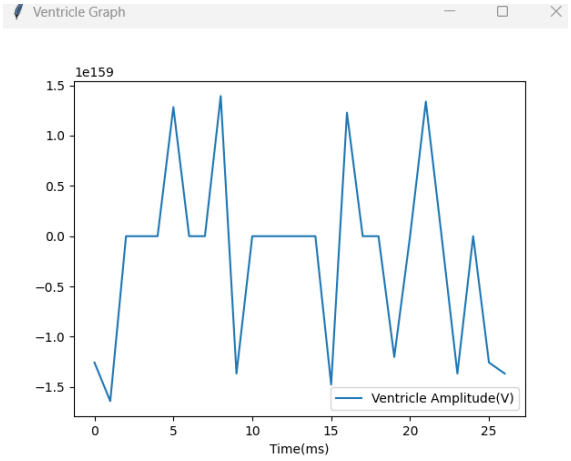

Fig 112. AAIR Ventricle Graph

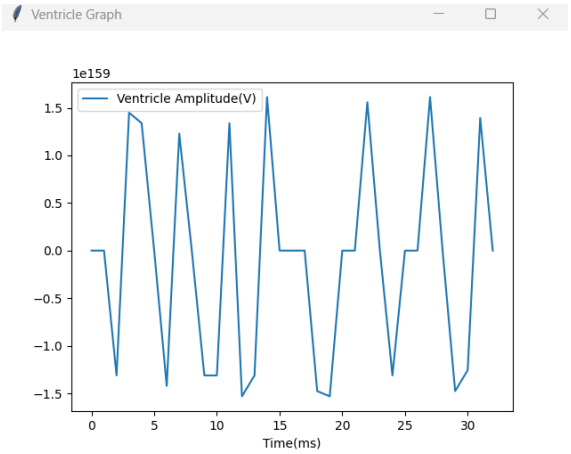

Fig 113. VOOR Ventricle Graph

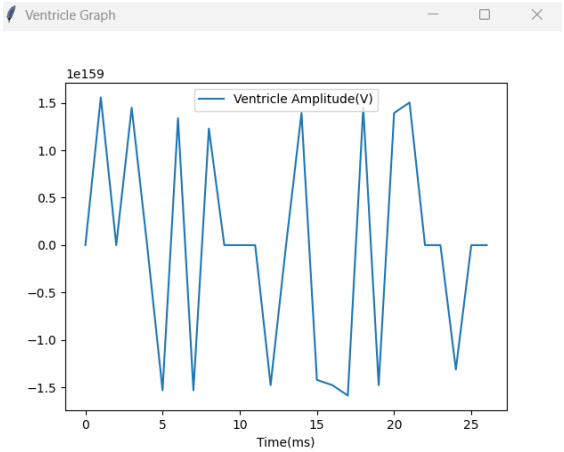

Fig 114. VVIR Ventricle Graph

Serial Communication

File name: DCM_Serial.py

Purpose: Used to build communication with the Pacemaker board. Sending and receiving data through UART that is preset on the Simulink side. The data transmitted can be verified by local database and def refresh () function.

Expected behavior: Imported to main.py for transferring parameters to the board by **input ()** and open engram graph with **receive ().**

Write the input parameter into user_inputs.txt if the parameters are all valid and within the range. Popup window shows successfully submitted.

Detect if the pacemaker connects to the correct serial port, Popup window show communicating if connected to the correct port.

Writing the input parameters into local database and detecting if the pacemaker has connected to the right port can ensure the parameters are sent from the DCM, and the user can check what's their latest input in the local txt file.

Result: Success

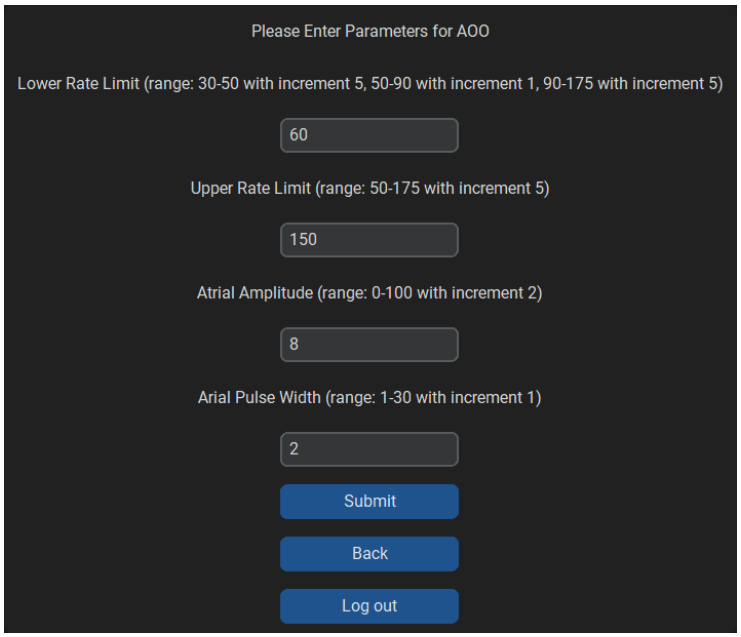

Fig 115. Valid user input test

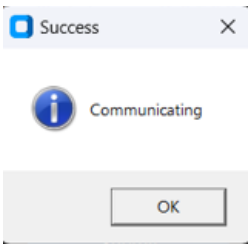

Fig 116. Pacemaker connected correctly

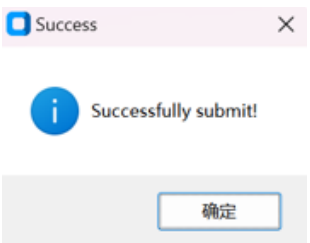

Fig 117. Feedback of parameters

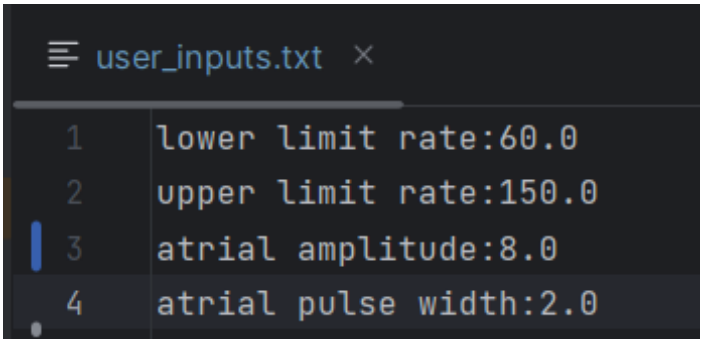

Fig 118. local database

Functions

def input():

Public or Private: Public
Parameters: lrl, url, amplitude, pw, rp, factor_input, threshold, mode
Returns: None
Purpose: Pack the input variables entered from one of the modes and send to the pacemaker through serial.
Description: Configure the communication port. After filling in all the required parameters listed and activate the submit button, the input() function will be called and convert all the data to bytes for transformation. The first two bytes transferred have to be the same as set in the Simulink system to start the transition.

```
def input(lrl, url, amplitude, pw, rp, factor_input, threshold, mode):
    Start = b'\x16'
    SYNC = b'\x22'\
    Fn_set = b'\x55'

    mode = struct.pack("B", int(mode))
    # rate = struct.pack("BB", 0, 1)
    rate = int(lrl*2).to_bytes(2, byteorder='little')
    max = int(url).to_bytes(2, byteorder='little')
    apl = int(amplitude).to_bytes(2, byteorder='little')
    pw = int(pw).to_bytes(2, byteorder='little')
    rp = int(rp).to_bytes(2, byteorder='little') #ref period
    factor = int(factor_input).to_bytes(2, byteorder='little')
    thre = int(threshold).to_bytes(2, byteorder='little') #comp_pwm threshold

    signal_echo = Start + Fn_set + mode + rate + pw + apl + rp+ thre + max + factor
    # print("test: ", signal_echo)
    with serial.Serial(frdm_port, 115200) as pacemaker:
        print("Connect")
        pacemaker.write(signal_echo)
```

Fig 119. def input() reference code

def receive():

Public or Private: Public
Parameters: None
Returns: ATR_Signal, VENT_Signal
Purpose: Receive data from the pacemaker and compute egram plots

Description: Configure the communication port. Once the pacemaker is pacing under one of the modes. Click corresponding buttons to open egram diagrams, the function will be activate while clicking the button and send the same code in pacemaker, like what input() done to enable the board send signals back to DCM. The receive data has a length of 8 each as its type is double. The data are unpacked and output to the graph functions for generating real time plots.

```
def receive():
    Start = b'\x16'
    SYNC = b'\x22'
    Fn_set = b'\x55'
    Signal_echo = Start + SYNC
    i=0
    while(i<15):
        Signal_echo = Signal_echo + struct.pack("B", 0)
        i = i+1

    with serial.Serial(frdm_port, 115200, timeout=1) as pacemaker:
        pacemaker.reset_input_buffer()
        pacemaker.reset_output_buffer()
        pacemaker.write(Signal_echo)
        data = pacemaker.read(16)
        ATR_signal = struct.unpack(">d", data[0:8])[0]
        # print(ATR_signal)
        VENT_signal = struct.unpack(">d", data[8:16])[0]

        print("finish reading")
        # return [sig1,sig2]
        return ATR_signal,VENT_signal
```

Fig 120. def receive() reference code

Submit Button

Overview: The button is provided to users to submit the input parameters and transfer the data into the pacemaker in the future assignment. The available input parameters depend on the “Programmable Parameters” table.

A Programmable Parameters

Parameter	Programmable Values	Increment	Nominal	Tolerance
Mode	Off DDD VDD DDI DOO AOO AAI VOO VVI AAT VVT DDDR VDDR DDIR DOOR AOOR AAIR VOOR VVIR	—	DDD	—
Lower Rate Limit	30-50 ppm 50-90 ppm 90-175 ppm	5 ppm 1 ppm 5 ppm	60 ppm	±8 ms
Upper Rate Limit	50-175 ppm	5 ppm	120 ppm	±8 ms
Maximum Sensor Rate	50-175 ppm	5 ppm	120 ppm	±4ms
Fixed AV Delay	70-300 ms	10 ms	150 ms	±8 ms
Dynamic AV Delay	Off, On	—	Off	—
Minimum Dynamic AV Delay	30-100 ms	10 ms	50 ms	
Sensed AV Delay Offset	Off, -10 to -100 ms	-10 ms	Off	±1 ms
A or V Pulse Amplitude Regulated	Off, 0.5-3.2V 3.5-7.0 V	0.1V 0.5V	3.5V	±12%
A or V Pulse Amplitude Unregulated	Off, 1.25, 2.5, 3.75, 5.0V	—	3.75V	—
A or V Pulse Width	0.05 ms 0.1-1.9 ms	— 0.1 ms	0.4 ms	0.2 ms
A or V Sensitivity	0.25, 0.5, 0.75 1.0-10 mV	— 0.5 mV	A-0.75 mV V-2.5 mV	±20%
Ventricular Refractory Period	150-500 ms	10 ms	320 ms	±8 ms
Atrial Refractory Period	150-500 ms	10 ms	250 ms	±8 ms
PVARP	150-500 ms	10 ms	250 ms	±8 ms
PVARP Extension	Off, 50-400 ms	50 ms	Off	±8 ms
Hysteresis Rate Limit	Off or same choices as LRL	—	Off	±8 ms
Rate Smoothing	Off, 3, 6, 9, 12, 15, 18, 21, 25%	—	Off	±1%
ATR Mode	On, Off	—	Off	—
ATR Duration	10 cardiac cycles 20-80 cc 100-2000 cc	— 20 cc 100 cc	20 cc	±1 cc
ATR Fallback Time	1-5 min	1 min	1 min	±1 cc
Ventricular Blanking	30-60 ms	10 ms	40 ms	—
Activity Threshold	V-Low, Low, Med-Low, Med, Med-High, High, V-High	—	Med	—
Reaction Time	10-50 sec	10 sec	30 sec	±3 sec
Response Factor	1-16	1	8	—
Recovery Time	2-16 min	1 min	5 min	±30 sec

Table 1. Programmable Parameters

Successful Submit & Communicating (AOO)

Overview: Submitting the input parameters under AOO mode which are in an appropriate range that is given in the “Programmable Parameters” table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text “Communicating”. Click “confirm” to close the prompt window. Next, pop a prompt window displaying the text “Successfully submit”. Click “confirm” to close the prompt window.

Result: Success

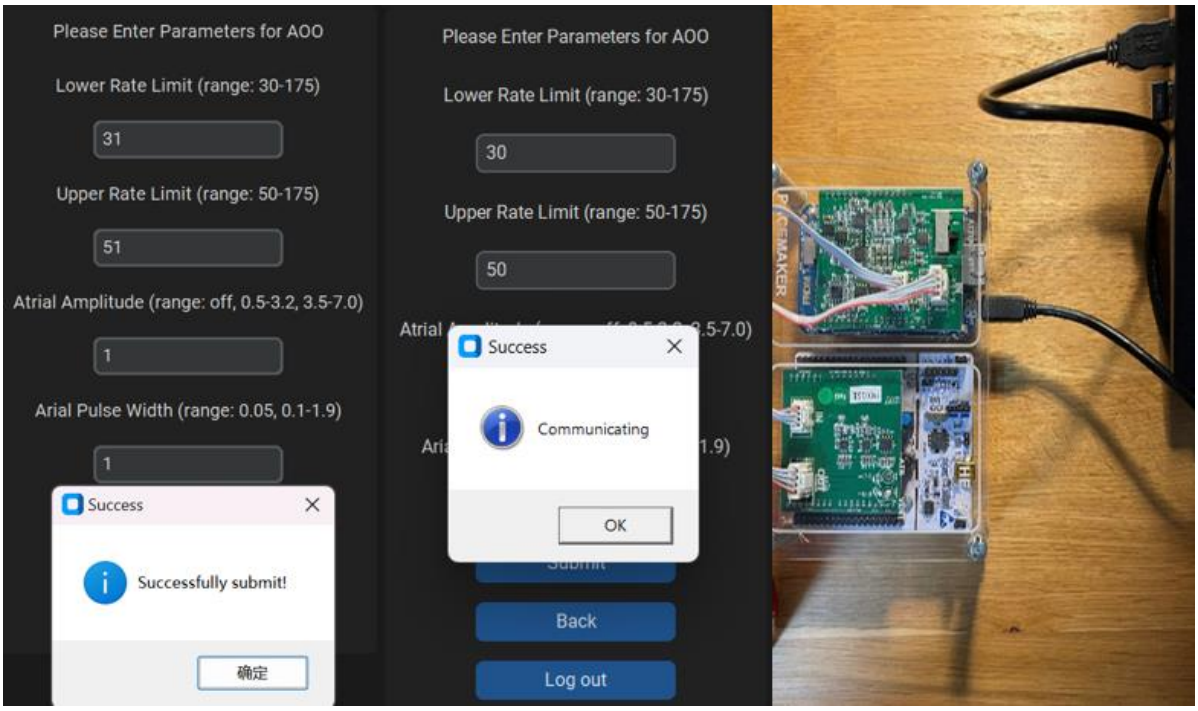

Fig 121. Successfully submitting the AOO input parameters under communicating status

Successful Submit & Communicating (VOO)

Overview: Submitting the input parameters under VOO mode which are in an appropriate range that is given in the "Programmable Parameters" table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text "Communicating". Click "confirm" to close the prompt window. Next, pop a prompt window displaying the text "Successfully submit". Click "confirm" to close the prompt window.

Result: Success

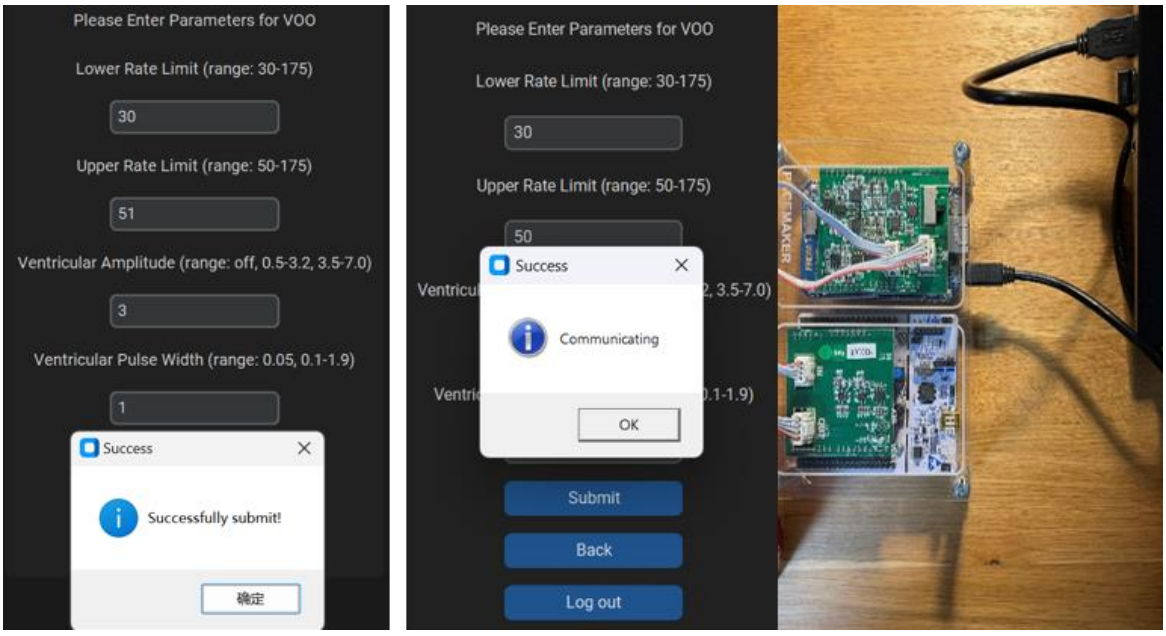

Fig 122. Successfully submitting the VOO input parameters under communicating status

Successful Submit & Communicating (AAI)

Overview: Submitting the input parameters under AAI mode which are in an appropriate range that is given in the “Programmable Parameters” table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text “Communicating”. Click “confirm” to close the prompt window. Next, pop a prompt window displaying the text “Successfully submit”. Click “confirm” to close the prompt window.

Result: Success

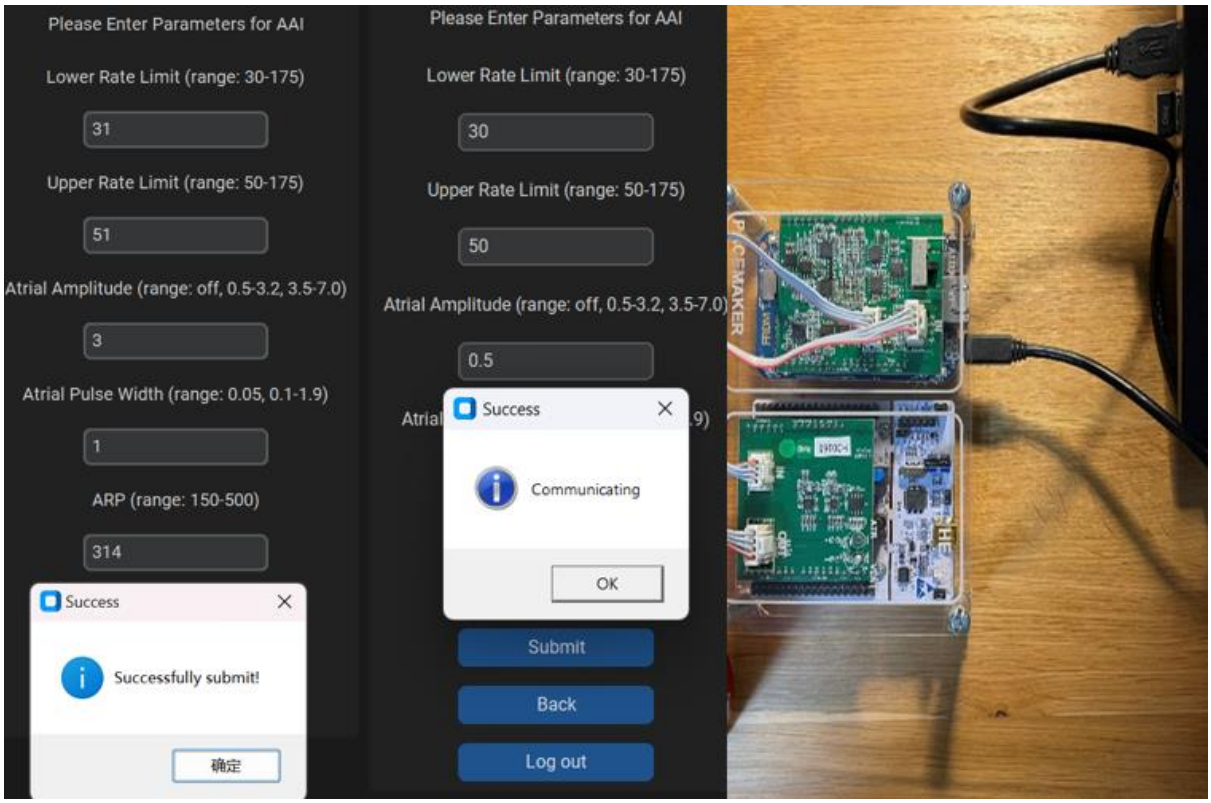

Fig 123. Successfully submitting the AAI input parameters under communicating status

Successful Submit & Communicating (VVI)

Overview: Submitting the input parameters under VVI mode which are in an appropriate range that is given in the “Programmable Parameters” table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text “Communicating”. Click “confirm” to close the prompt window. Next, pop a prompt window displaying the text “Successfully submit”. Click “confirm” to close the prompt window.

Result: Success

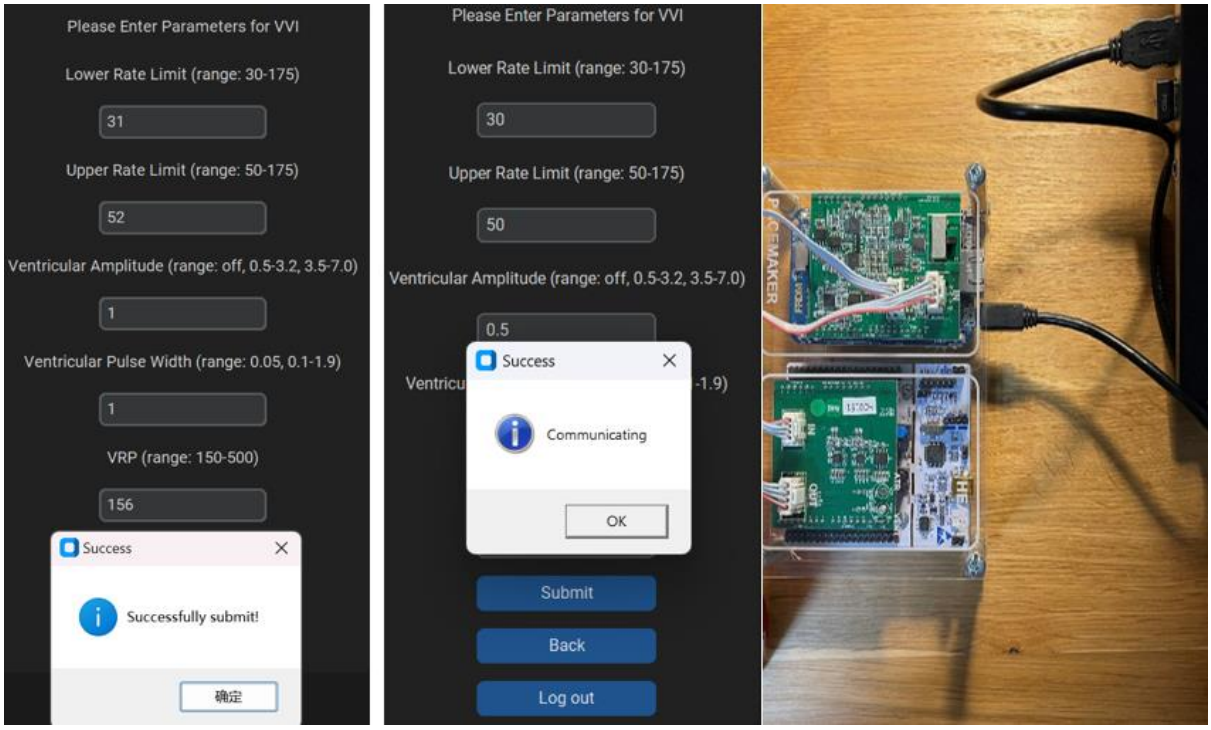

Fig 124. Successfully submitting the VVI input parameters under communicating status

Successful Submit & Communicating (AOOR)

Overview: Submitting the input parameters under AOOR mode which are in an appropriate range that is given in the "Programmable Parameters" table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text "Communicating". Click "confirm" to close the prompt window. Next, pop a prompt window displaying the text "Successfully submit". Click "confirm" to close the prompt window.

Result: Success

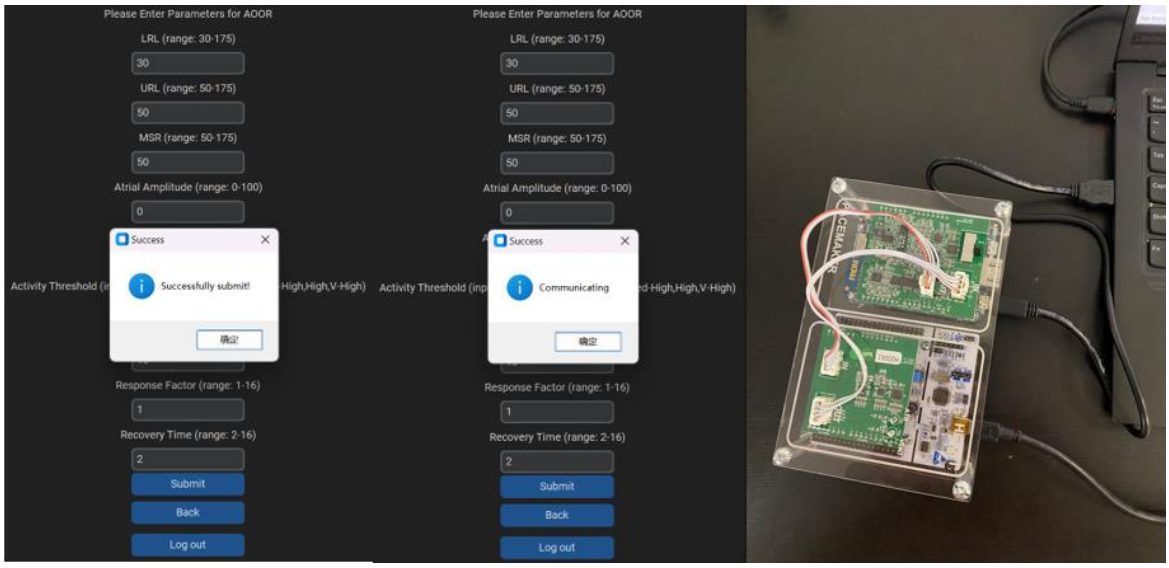

Fig 125. Successfully submitting the AOOR input parameters under communicating status

Successful Submit & Communicating (VOOR)

Overview: Submitting the input parameters under VOOR mode which are in an appropriate range that is given in the “Programmable Parameters” table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text “Communicating”. Click “confirm” to close the prompt window. Next, pop a prompt window displaying the text “Successfully submit”. Click “confirm” to close the prompt window.

Result: Success

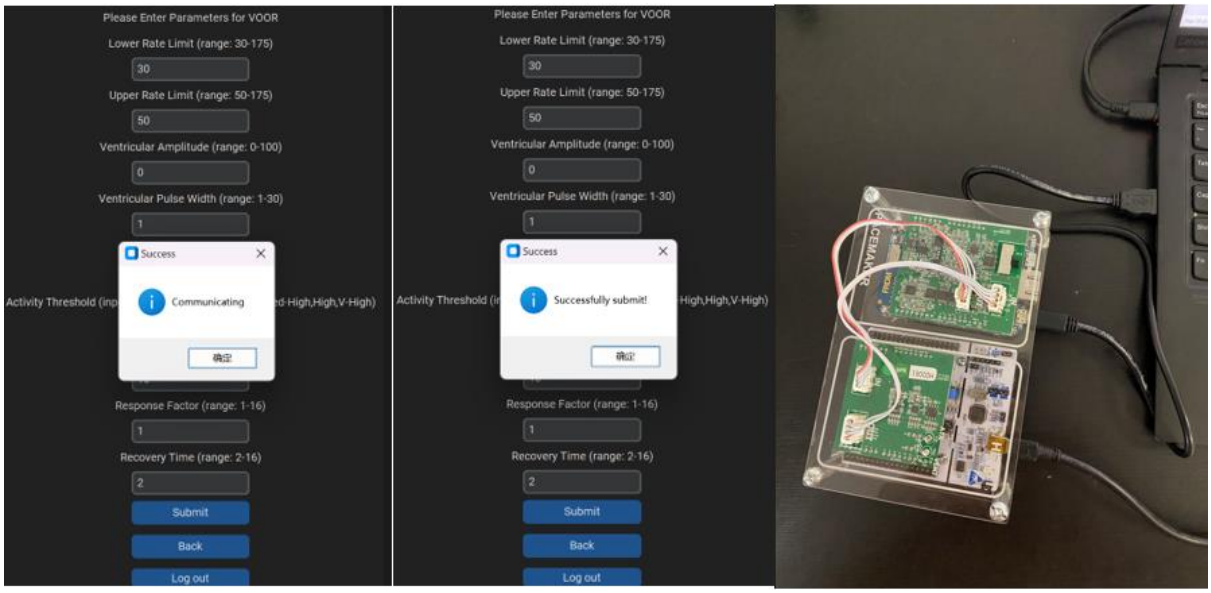

Fig 126. Successfully submitting the VOOR input parameters under communicating status

Successful Submit & Communicating (AAIR)

Overview: Submitting the input parameters under AAIR mode which are in an appropriate range that is given in the “Programmable Parameters” table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text “Communicating”. Click “confirm” to close the prompt window. Next, pop a prompt window displaying the text “Successfully submit”. Click “confirm” to close the prompt window.

Result: Success

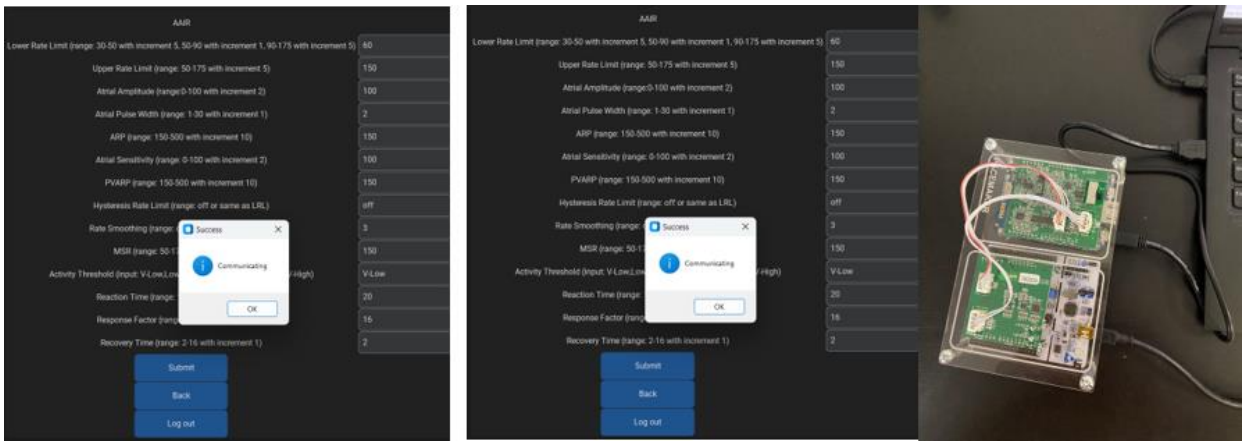

Fig 127. Successfully submitting the AAIR input parameters under communicating status

Successful Submit & Communicating (VVIR)

Overview: Submitting the input parameters under VVIR mode which are in an appropriate range that is given in the "Programmable Parameters" table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text "Communicating". Click "confirm" to close the prompt window. Next, pop a prompt window displaying the text "Successfully submit". Click "confirm" to close the prompt window.

Result: Success

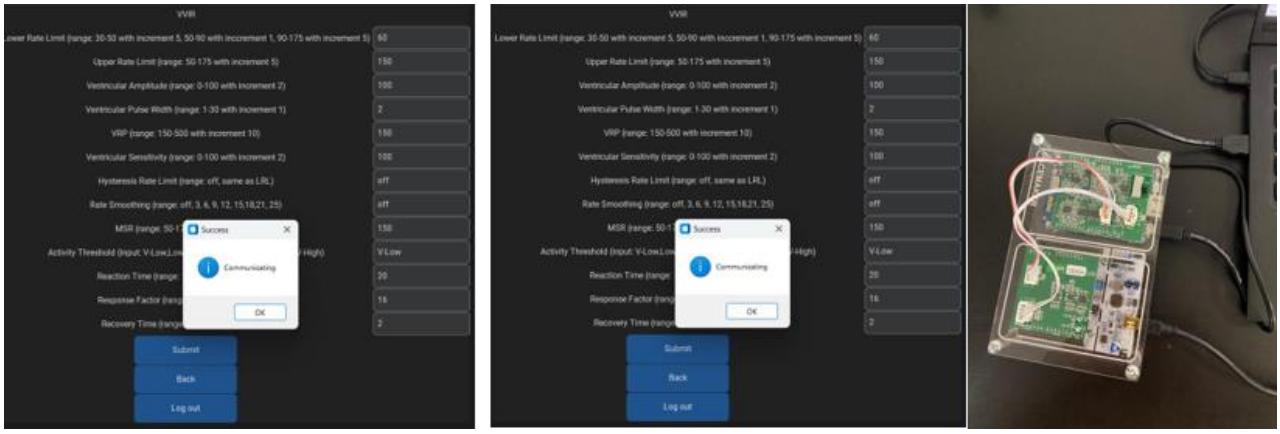

Fig 128. Successfully submitting the VVIR input parameters under communicating status

Out of Range Input

Overview: Submitting the input parameters which are not in an appropriate range that is given in the "Programmable Parameters" table. Meanwhile, the pacemaker is connected to the DCM.

Expected behavior: Pop up a prompt window displaying the text "Input parameter is out of range". Click "confirm" to close the prompt window. Next, pop a prompt window displaying the text "Communicating". Click "confirm" to close the prompt window.

Result: Success

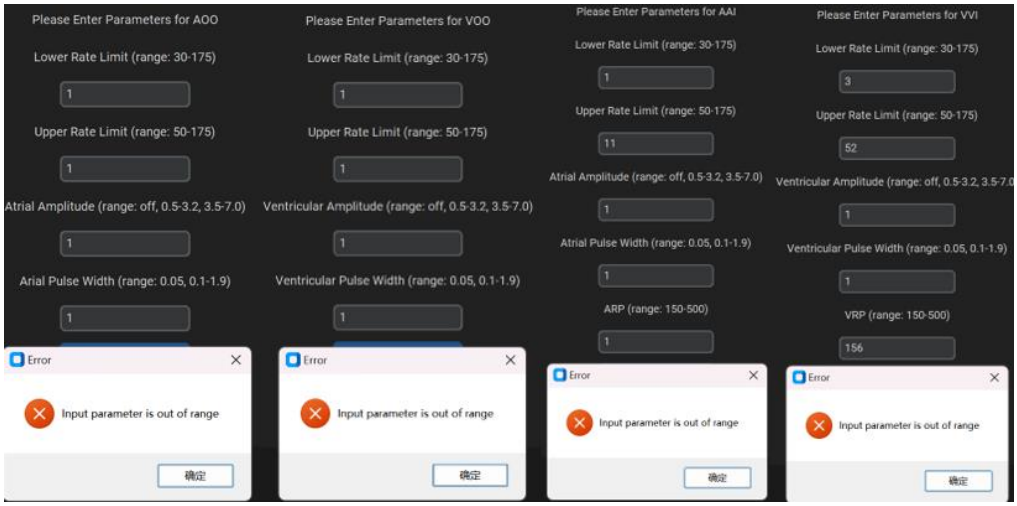

Fig 129. Unsuccessfully submitting due to any out-of-range input

Unexpected Input

Overview: Submitting the input parameters which is not a number, except the “off” input in atrial and ventricular amplitude placeholders.

Expected behavior: Pop up a prompt window displaying the text “Please entry a valid parameter”. Click “confirm” to close the prompt window.

Result: Success

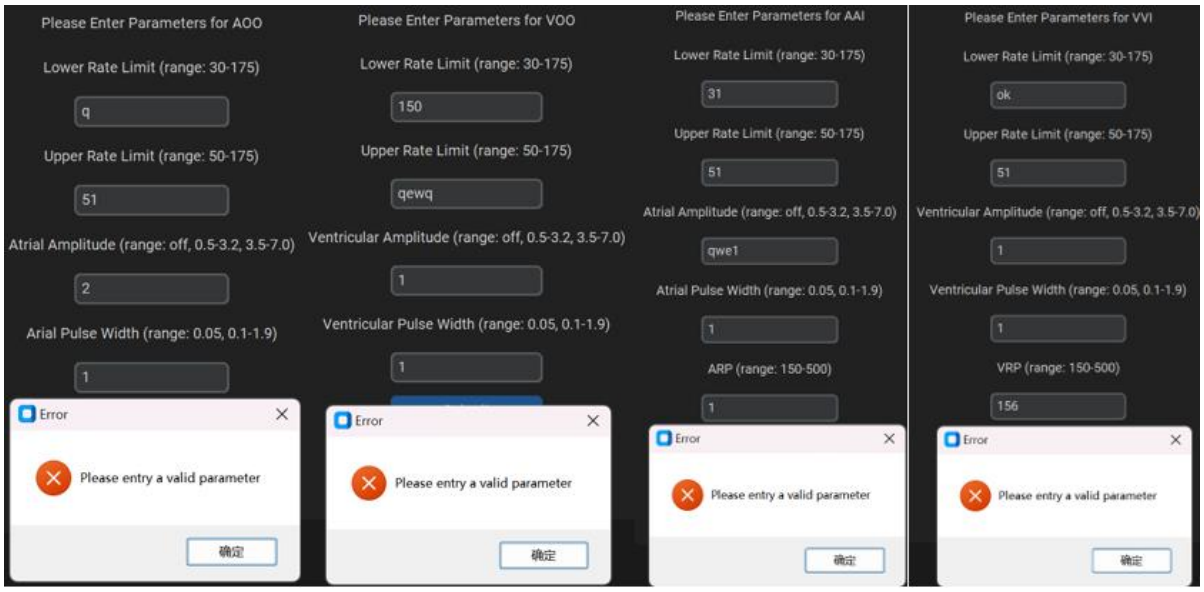

Fig 130. Unsuccessfully submitting due to any unexpected input

Empty Input

Overview: Submitting the input parameters with one or more empty entries.

Expected behavior: Pop up a prompt window displaying the text “Please entry a valid parameter”. Click “confirm” to close the prompt window.

Result: Success

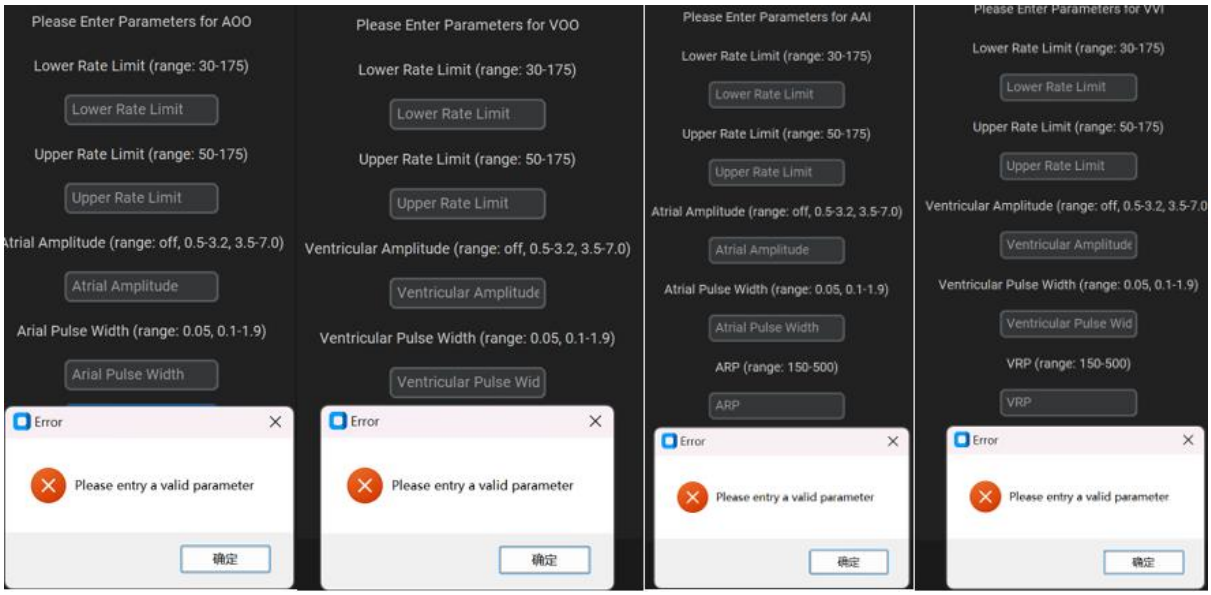

Fig 131. Unsuccessfully submitting due to any empty input

Device disconnected

Overview: The pacemaker is not connected to our DCM

Expected behavior: Pop up a prompt window displaying the text “Serial port COM3 is not open”. Click “confirm” to close the prompt window.

Result: Success

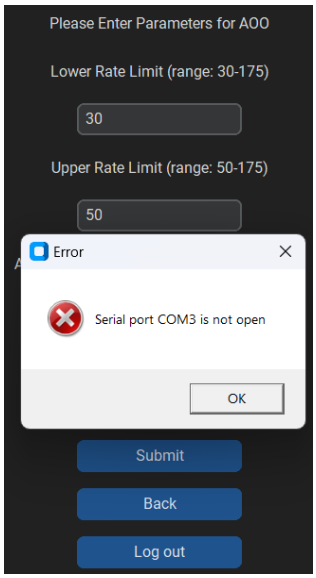

Fig 132. Device disconnected

Back Button

Overview: The button allows user to back to the previous page.

Back to Login Page

Overview: Allow users back to the *login page* by clicking the “Back” button.

Expected behavior: Going to the *login page*.

Result: Success

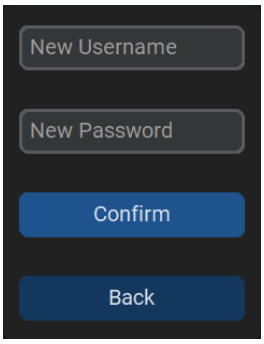

Fig 133. Back button in register page is clicked

Back to Mode Page

Overview: Allow users back to the *mode page* by clicking the “Back” button.

Expected behavior: Going to the *mode page*.

Result: Success

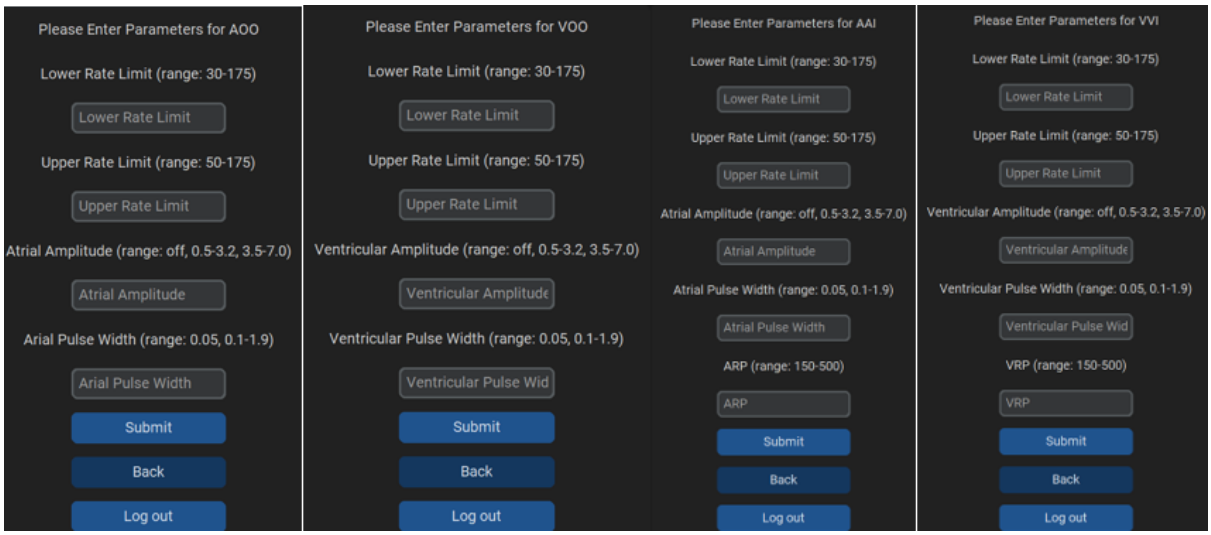

Fig 134. Back button in 4 mode pages are clicked

Logout Button

Overview: Allow users back to the *login page* by clicking the "Log out" button.
Expected behavior: Going to the *login page*.
Result: Success

Fig 135. Log out button in mode choosing page is clicked

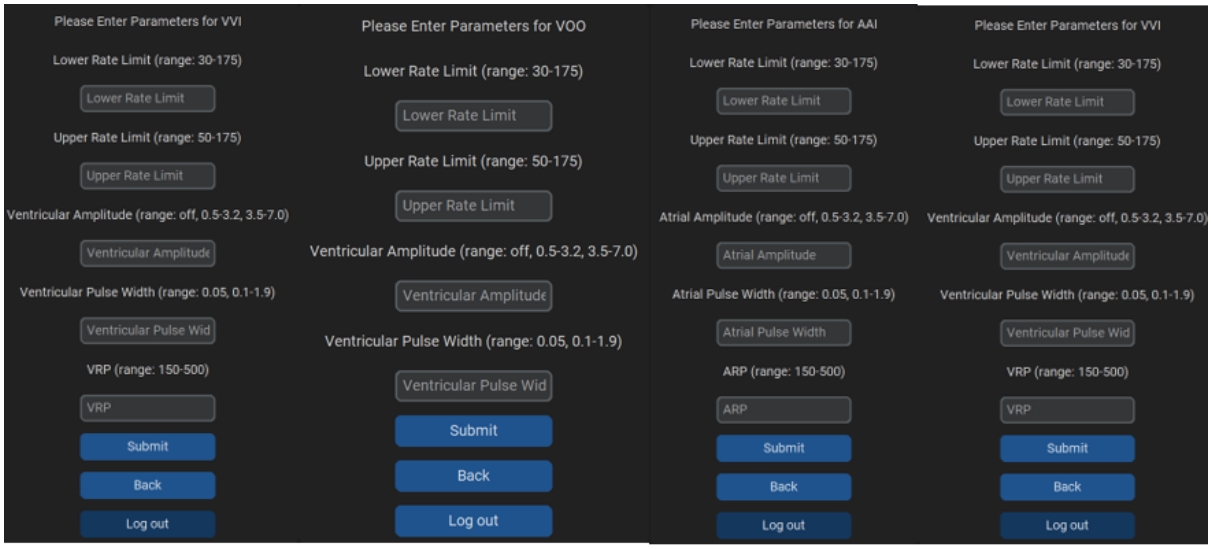

Fig 136. Log out button in 4 mode pages are clicked

Future Development

Different Device Connection

We cannot implement prompts such as "Device Replaced" when different devices are connected because we have not yet covered the topics of pacemaker serial numbers and attributes. Therefore, this will require further learning and development in the future to be implemented.

OOP

Adopting object-oriented programming principles will enhance the efficiency and development timeline of the software. While rapid development allows a team to implement most of the features quickly, it results in cluttered and difficult-to-read code, severely affecting subsequent development. Moreover, the current code's efficiency could be higher because it lacks encapsulation and polymorphism through features like classes, leading to overlapping and confusing functionality.

Translation

The PACEMAKER file mentioned the need for a GUI to be presented in eight languages for various users. We can only achieve an English version of the GUI at this stage, and we attempted to import a translation library into the code. However, the results could have been better and easier to use. Our team also does not believe that creating eight identical but language-specific pages is feasible, as we cannot handle such a significant development workload for the additional seven languages. Therefore, we must explore alternative libraries and integrate them for the translation work.

Scrollbar and Drop-down menu

After adding some buttons and textboxes our current DCM is starting to get a little cramped for space on each page. Some of our pages have features that can't be displayed on a small screen. And to display all the functions we have sacrificed the layout of some mode pages. We plan to add some scroll wheels and drop-down menus in the future to expand and organize our DCM, to optimize our interface and to meet the needs of different users.

Safety

Try Except Statement

In the implementation of the code, we extensively use try-except statements within functions rather than relying solely on if-else statements. The advantage of try-except lies in its ability to prevent system crashes effectively when there is a potential risk of code execution errors, thereby avoiding irreparable losses. This construct is capable of handling exceptional situations, whereas if-else merely directs the program flow into different branches.

Here is an example in AOO mode:

```
# Verify the lower rate limit parameter
try:
    aoo_lrl = float(aoo_lrl)
    if not (30 <= aoo_lrl <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
# Verify the upper rate limit parameter
try:
    aoo_url = float(aoo_url)
    if not (50 <= aoo_url <= 175):
        # the input is out of range
        out_of_range = True
except ValueError:
    # the input is not a number
    invalid_input = True
```

Fig 137. example of try-except statement in AOO mode

Modularity

Throughout the entire system, we have enhanced modularity in response to the shortcomings of the assignment. This involves breaking down the entire system into several loosely interconnected modules, such that when a module encounters issues, it minimally affects other parts. This accomplishment aligns with the goal of achieving high cohesion and low coupling in the system design.

Output Data Visualization

We have stored user-inputted data in a text file for easy verification of data accuracy. This practice helps us inspect the data at any time, mitigating the risk of system crashes or other potential hazards that may arise from erroneous data.

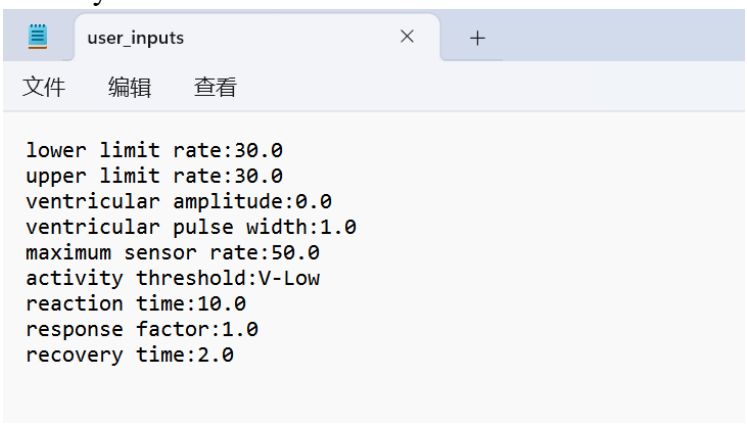

Fig 138. example of storing data in user_inputs.txt file

Assurance Case

FTA

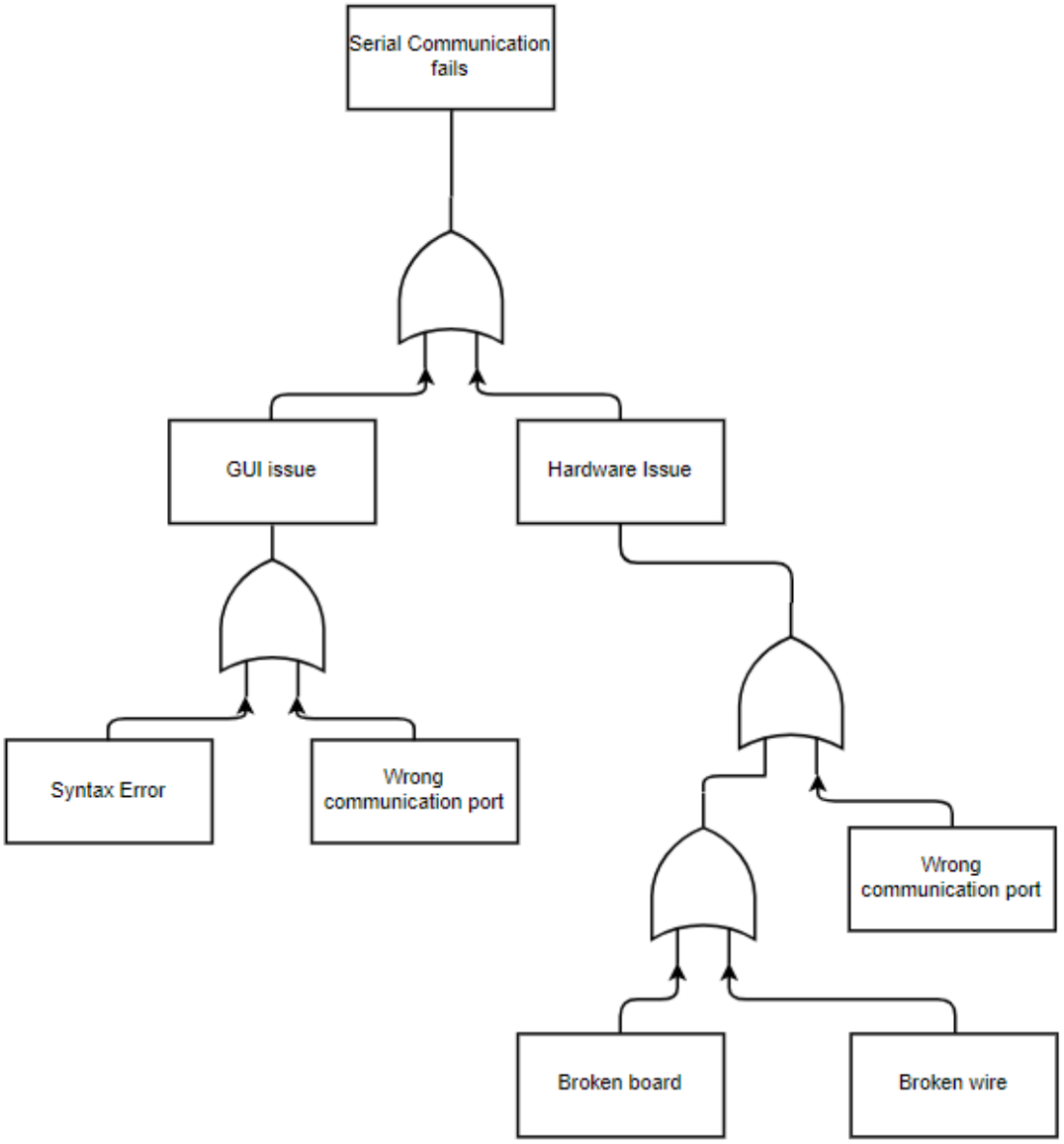

Fig 139. FTA

FMEA

Component	Failure Mode	Causal Factors	Immediate Effect	System Effect	Safety Requirement
Board	Disconnection	1. Wrong port 2. Borad is broken	1. Cannot connect to a laptop.	System does not work	A good quality board connect with

			2. Cannot connect with a laptop		laptop with correct connection,
Wire	Disconnection	Wire is broken	Cannot connect board and laptop	System does not work	A good quality wire which can keep connection
GUI	1. Syntax Error 2. Disconnection	1. Typo or wrong logic when coding 2. Wrong connecting port	Disconnection	System does not work	Correct syntax that can connect with board to transmit data

Table 2. FMEA

References

[1] Guido. Van Rossum and F. L. Drake, *An introduction to Python : release 2.2.2*. Network Theory Ltd, 2003.

[2] H. Seetha, V. Tiwari, K. R. Anugu, D. S. Makka, and D. R. Karnati, "A GUI Based Application for PDF Processing Tools Using Python & CustomTkinter," *Int J Res Appl Sci Eng Technol*, vol. 11, no. 1, pp. 1613–1618, Jan. 2023, doi: 10.22214/ijraset.2023.48848.

[3] A. Kak, "Lecture 15: Hashing for Message Authentication Lecture Notes on 'Computer and Network Security.'"

[4] J. W. Shipman, "Tkinter 8.5 reference: a GUI for Python." [Online]. Available: http://www.nmt.edu/tcc/help/pubs/tkinter/tkinter.pdf

[5] M. Sharma, N. Agarwal, and S. R. N. Reddy, "Design and development of daughter board for USB-UART communication between Raspberry Pi and PC," in *International Conference on Computing, Communication & Automation*, IEEE, May 2015, pp. 944–948. doi: 10.1109/CCAA.2015.7148532.

[6] S. Tanaji Bhosale VPIMSR, S. Bhosale, M. Tejaswini Patil, and M. Pooja Patil, "SQLite International Journal of Computer Science and Mobile Computing SQLite: Light Database System," 2015. [Online]. Available: https://www.researchgate.net/publication/279621848