SFWRENG 3K04: Software Development

Assignment 2

DCM

Group 25

December 1st, 2021

Contents

1 DCM

1.1 DCM Overview

The purpose of the Device Controller-Monitor (DCM) is to act as a simple and functional graphical interface for the user to interact with. Designed with the intent of ease of use and reliability, the system has been programmed into modular parts, separating back-end database flow as well as front-end user interface that allows. The separation of respective parts allows for safe data transfer in the event of modular updates for each system. The DCM is divided into five main modules:

- Main
- Windows
- Users
- Pacemakers
- Egram

Both the main and window modules control the front-end design in which the user interacts directly through the DCM to the pacemaker. These modules contain only graphical output and input for the user to view and modify information regarding the pacemaker settings. The graphical interface is designed to be as simplistic and intuitive as possible in the event that the user (in the most likely cases, a doctor) is not technologically proficient.

The users and pacemakers modules contain only the back-end design to edit and modify the direct databases that store up to 10 users as well as the edit history of the sensing mode parameters for each user's respective pacemaker. No graphical design is included in these sections for increased modularity. The use of get functions allow for efficient information hiding and thus, the program is easily modifiable with no effect on the user interface.

1.2 DCM Requirements

Current requirements for the DCM that are likely to change include:

1. Implementing more pacing modes
2. Switching among multiple pacemakers

1.3 DCM Design Decisions

The first design decision encountered was choosing a language to utilize for the DCM. Upon consideration, Python was selected due to its versatility and ease of use. The graphical user interface (GUI) does not require a high degree of control over specific hardware, thus a high level programming language is acceptable. Python also has the benefit of a wide selection of libraries and packages which increases options for the design of the GUI.

For the visual and interactive development of the GUI, PySimpleGUI was selected also due to its versatility and ease of use. PyQt5 was also considered, however, was not selected due to its more complicated package use and requirement of an additional program.

Modules were separated based on functionality and interactivity. The main module is used solely for the purpose of handling data flow and activation/deactivation of respective windows in regards to user input. The windows module is used to modify the visual appearance of the respective windows. The users and pacemaker modules are for storing and modifying the information of their respective functions. Part of pacemakers and egram modules handle the communication between DCM and pacemaker. This separation of front-end and back-end modules allows for high cohesion and low coupling, giving the DCM effective modularity.

Current design decisions that are likely to change include:

1. Wrap the windows module with a class to increase the modularity and information hiding
2. Show the parameters review and modification on the same tab to increase readability
3. Add date and time in the modification history
4. Provide an interface to check the history of the parameter's modification in the application instead of opening the .json file manually
5. Use the slider input for parameters to enforce the valid input
6. Add choice for resetting the parameters
7. Make better appearance for egram

1.4 Modules

1.4.1 Main

This module combines all the modules and coordinates them to achieve the requirements of the DCM. This module contains only one function "main()" using the structural programming paradigm to manipulate windows, users, and pacemakers.

Control Flow

1. The module (or application) starts with a welcome window to indicate either logging in or registering to the system. This window is both the entry and exit of the whole system.

    a. If the user chooses logging in, then a new window will show up to require username and password, while hiding (deactivating) the welcome window. The user validating will be accomplished by the users module.

    b. If the user chooses to register, then a new window will show up asking the user to input the username and password twice. Again, the user information updating and validating will be handled by the users module.

2. After entering the main system, the user will need to connect to the pacemaker first to connect the pacemaker through UART and see the parameters of it. The three modes of egram in the status tab bring up the real-time egram made from the data transferred from the pacemaker. At any time, the "get parameters" will show the current parameters of the pacemaker in the status tab.

3. The user can switch to the modify tab to set the parameters. First save then send to pacemaker. The modification of the parameters will also need the pacemakers module to validate and commit.

4. After working with the parameters, the exit button will bring the user back to the welcome window and either log in as another user or exit from the welcome window which is also exiting from the application.

1.4.2 Windows

The windows.py module contains the parameters for the GUI. Here, the visual windows can be modified directly with no effect on the decision-making process for the transitions between interfaces. Only the visual portion is contained within this module. The windows are generated using the Python package PySimpleGUI, which was chosen for its visual simplicity and ease of use. Each window is stored as a separate function and can be individually edited. The functions are as follows:

Public Functions:

1. create_welcome_window()

This function uses PySimpleGUI to format the visual appearance of the GUI of the welcome window where the user selects to login or register. It also accepts user inputted information and passes it through to the main module for data processing and allows for the decision flow to the next window. Note that there are no input parameters as the function is purely visual.

2. create_login_window()

This function formats the visual appearance of the login window. The user is prompted to enter their username and password. No input parameters are needed.

3. create_register_window()

This function formats the visual appearance of the register window. The user is prompted to create a username and password. No input parameters are needed.

4. create_main_window()

This function formats the visual appearance of the main window. The pacing mode, lower rate limit, atrial amplitude, atrial pulse width, ARP, upper rate limit, ventricular amplitude, ventricular pulse width, VRP, AVDelay, activity threshold, reaction time, response factor, recovery time are all listed. A separate tab is included to allow the user to modify the values. The buttons for communicating with the pacemaker are also included. No input parameters are needed.

5. create_egram_window()

This function formats the window showing the egram.

1.4.3 Users

The users.py module only contains the back-end logic design that stores and modifies the user information including the usernames and passwords of up to ten unique users, as well as the function for logging in and registering new users. There is no modification of the GUI (visual or interactive) in this module. The functions in this module are as follows, all belonging to a public class Users:

Public Functions:

1. get_active(self)

This function returns the latest list of active users

2. login(self, values)

This function attempts to check the user inputted login information with an existing user already stored. It will output true on success, and false otherwise.

3. register(self, values)

This function takes the user inputted registration information and attempts to register a new user. If the inputted username is already in use, the inputted entries are empty, or the maximum number of users have been registered, it will return false.

Global Variables:

1. USERS_FILE

This variable stores the directory path of the local file storing the user information

2. DEFAULT_USER

This variable is the default record of a single user. The default username and password are both “admin”.

3. USER_KEYS_TO_ELEMENT_KEYS

This variable uses a **dictionary** mapping user keys to the window’s element keys. It maps the username to “-USERNAME-” and password to “-PASSWORD-”

4. self.users_active

This variable is a blank array that is used to store the current active users.

5. self.info

This variable holds all the users records in a list of dictionaries. The runtime equivalent of local file.

Private Functions:

1. __new__(cls)

This is the constructor for the Users object. It outputs an initialized instance of the Users class.

2. load_users(self, users_file, default_user)

This function takes in the file path where the user records are stored and loads the history from the file to self.info. In the event that no user information is found, a blank user file will be created using the default user admin.

3. save_users(self, users_file)

This function saves the current user information into the local file.

Internal Behavior:

1. __new__(cls):

This function first checks if there exists an instance of the Users class. If not, it then creates and initializes the instance by calling the load_users function to get access to the stored user information

2. load_users(self, users_file, default_user)

This function tries to load the local .json file storing the usernames and passwords in order to check the user inputted values with the previously registered users. If no local file is found, it will create a blank file and automatically store a default user “admin”

3. save_users(self, users_file)

This function writes the user inputted information into the local .json file.

4. get_active(self)

This function returns the array of active users.

5. login(self, values)

This function checks the user inputted information to the previously loaded record of usernames and passwords. If they match, it will output true to bring the user to the next window through the main module. Otherwise, it will output false and prompt the user to try again.

6. register(self, values)

This function attempts to register a new user using the user inputted registration information. This will output logic false if no more users may be stored, if the entries are empty, or if the desired username is already in use.

1.4.4 Pacemakers

This module is used to access and manipulate the history of modification of parameters. This module is implemented using the singleton pattern to ensure there is only one instance of the whole history. The history records are stored in a json file locally in the form of a ***list of dictionaries*** (in Python's terminology). It also handles the communication between DCM and the pacemaker to get and set parameters on the pacemaker in real-time.

Public Functions:

1. get_param(self)

This function returns the latest version of parameters.

2. set_param(self, values)

This function takes the input values of pacemaker parameters and updates the state of the history (both runtime state and local file).

3. get_param_com(self)

This function requests current parameters of the pacemaker and returns it in a dictionary for the main module to update the status tab.

4. set_param_com(self)

This function sends the latest saved parameters to the connected pacemaker.

Global Variables:

1. MODE

This variable stores the implemented modes of the pacemakers in a tuple. At this time, there are ten modes: “AOO”, “AOOR”,“VOO”,”VOOR”,”DOO”, “DOOR”, “AAI”,”AAIR”, “VVI”,”VVIR”.

2. DEFAULT_PARAM

This variable is the default record of the pacemaker parameters in a **dictionary**. It maps each parameter to None.

3. PARAM_KEYS_TO_ELE_KEYS

This variable is a **dictionary** mapping the parameters to elements in the ***status tab*** of the main window of the application. The key is a parameter, and the value is the *element key* (identifier for elements on the window) corresponding to specific parameters.

4. PARAM_KEYS_TO_ELE_KEYS_IN

This variable is a **dictionary** mapping the parameters to elements in the ***modify tab*** of the main window of the application. The key is a parameter, and the value is the *element key* (identifier for elements on the window) corresponding to specific parameters.

5. MODE _TO_ELE _KEYS

This variable is a dictionary mapping the modes of the pacemaker to the elements in the review tab corresponding to the parameters needed in each specific mode. The key is mode, and the value is a tuple of element keys.

6. MODE_TO_PARAM _KEYS

This variable is a dictionary mapping the modes of the pacemaker to the parameters needed in each specific mode. The key is mode, and the value is a tuple of parameters.

7. PACEMAKERS_FILE

This is the general path for the json file storing the history records.

8. PORT

This is the port name for serial communication, UART in this case.

9. RATE

This is the baud rate for serial communication, UART in this case.

10. COM_ORDER

This is the parameters order transferred between the DCM and the pacemaker.

11. COM_FORMAT

This is the format string used by struct to unpack the received byte stream.

12. COM_PARAM_SIZE

This is the byte number that needs to be read from the serial port when receiving data.

13. COM_GET_H

This is the header to initiate the process on the pacemaker to send the current parameters on it back to DCM.

14. COM_SET_H

This is the header to initiate the process on the pacemaker to set the parameters on it using the parameters sent by DCM.

15. self.pacemakers_history

This variable stores the history of modification in a list of dictionaries.

16. self.pacemakers_current

This variable stores the latest record of modification in a dictionary.

Private Functions:

1. validate_keys(self, key, value)

This function is used to validate the input for each parameter according to pacemaker specification. It takes in the key (parameter) and the value (user input), and returns True or False for validation.

2. type_converter(self, key, value)

This function converts the parameters type from string to specified type complying with that of pacemaker.

3. __new__(cls)

This is the constructor for the Pacemakers object. It outputs an initialized instance of the Pacemakers class.

4. load_pacemakers(self, pacemakers_file)

This function takes in the file path where the modification history records are stored and loads the history from the file to self.pacemaker_history. In addition, update the self.pacemaker_current to the latest state of the pacemaker.

4. save_pacemakers(self, pacemakers_file)

This function saves the modified history into the local file.

Internal Behavior:

1. __new__(cls):

This function first checks if there exists an instance of the Pacemakers. If not, it then creates and initializes the instance by calling the load_pacemakers function to get access to the history.

2. load_pacemakers(self, pacemakers_file)

This function tries to open the file storing the history and uses the json module to get the history into an object variable called pacemaker_history which is the runtime equivalent of the local file. And then it updates the variable storing the latest history record. If the file does not exist yet, it will create a file and add the DEFAULT_PARAM into it as well as updating the history and latest record variables.

3. save_pacemakers(self)

This function stores the content in the pacemakers_history back to the local file using the json module.

4. get_param(self)

This function constructs a dictionary mapping the element keys on the status tab to the latest parameters.

5. set_param(self, values)

This function constructs a dictionary mapping the parameters to the user inputs and updates the history and latest record. It first extracts the pacing mode parameter and uses the mode to determine which parameters are needed and then add them to the new dictionary after validating the inputs. The parameters not needed in specified pacing mode will get None as the value.

6. validate_keys(self, key, value)

This function uses multiple if statements to check the input for each parameter. The standard for validating an input value is from the pacemaker specification.

7. type_converter(self, key, value)

This function is similar to validate_key, but it assumes the keys have been validated which means it just needs to convert keys directly to desired types.

8. get_param_com(self)

This function sends header code packed by struct module to serial port using pyserial module which requests the pacemaker to send back the current parameters. Then it parses the bytes using the struct module to correct values. At last, it maps the parameter keys to window element keys for updating the status tab.

9. set_param_com(self)

This function does opposite operations to the get_param_com. It wraps the current parameters to bytes following the specified order in COM_ORDER and sends them to the serial port.

Note: When saving, loading, or sending parameters, those parameters not needed by the corresponding pacing mode will be set None or 0.

1.4.5 Egram

Public Functions:

1. draw_graph(self)

This function draws the egram on the area specified by the egram window.

Global Variables:

1. self.atr

This is a list holding the data point for signals from the atrium.

2. self.vent

This is a list holding the data point for signals from the ventricles.

3. self.fig

A figure object for egram to show.

4. self.ax

An ax object for detailed formatting of the egram.

5. self.canvas_agg

A handle from matplotlib to connect the figure object with the canvas object in the window.

6. self.mode

Mode of egram (atrial, ventricle, both).

Private Functions:

1. update_data(self)

This function updates the data point for generating a real-time egram.

2. clip(slef)

This function clips the data point to desired length for drawing.

Internal Behavior:

1. Update_data(self)

This function requests pacemaker in the same way as communication in pacemakers module to send back the signals from atrium and ventricles and add them to the self.atr and self.vent to update the graph in real-time.

2. clip(self)

This function handles the start of the egram where data points are not enough to fill the graph. It fills the extra empty point with 0.

3. drawe_graph(self)

This function first updates the data points then draws the graph according to the self.mode, either drawing one of the atrium and ventricles or both. It calls the handle from the matplotlib.

4. __init__(self, canvas, mode)

This function is not a trivial initializer as it combines the canvas object of the window, the mode of the egram, together with the figure object.

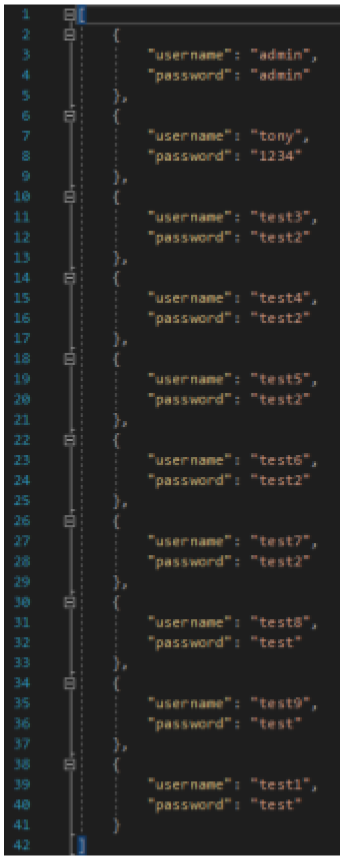
Note: Every time a new egram is requested, a new egram object will be created.

1.5 DCM Testing

Functionality:

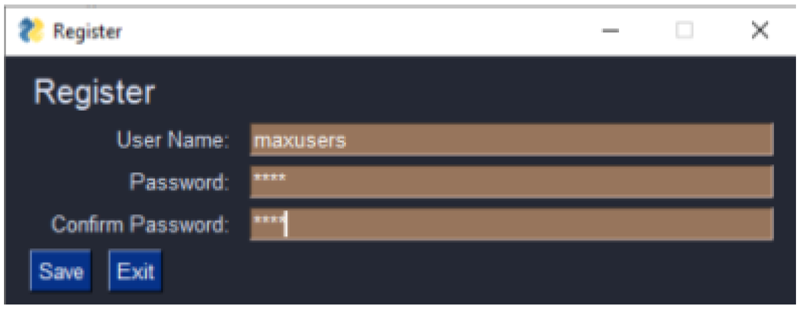
1. Registering the max number of users

The purpose of this test is to check if the DCM prevents the user from registering more than 10 users. To begin, 10 users were created and stored in the users.json file as indicated below (Figure 1.1). Then, there was an attempt to register a new user (Figure 1.2). The expected output is that an error message appears alerting that the max number of users have already been registered (Figure 1.3), and was matched by the actual output. The test result is a pass.



```
[
    {
        "username": "admin",
        "password": "admin"
    },
    {
        "username": "tony",
        "password": "1234"
    },
    {
        "username": "test3",
        "password": "test2"
    },
    {
        "username": "test4",
        "password": "test2"
    },
    {
        "username": "test5",
        "password": "test2"
    },
    {
        "username": "test6",
        "password": "test2"
    },
    {
        "username": "test7",
        "password": "test2"
    },
    {
        "username": "test8",
        "password": "test"
    },
    {
        "username": "test9",
        "password": "test"
    },
    {
        "username": "test1",
        "password": "test"
    }
]
```

Figure 1.1: 10 stored users



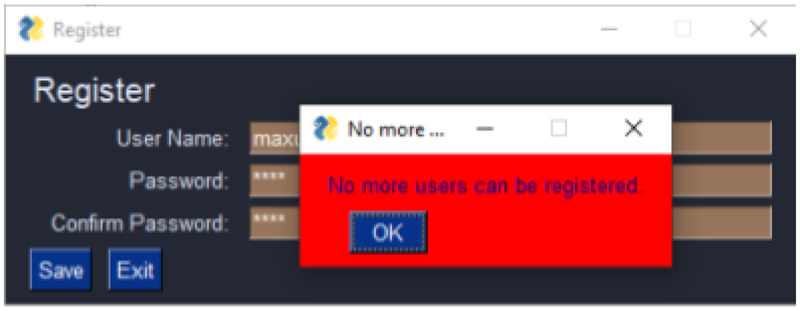
Register

User Name: maxusers

Password: ****

Confirm Password: ****

Figure 1.2: Attempting to register a new user

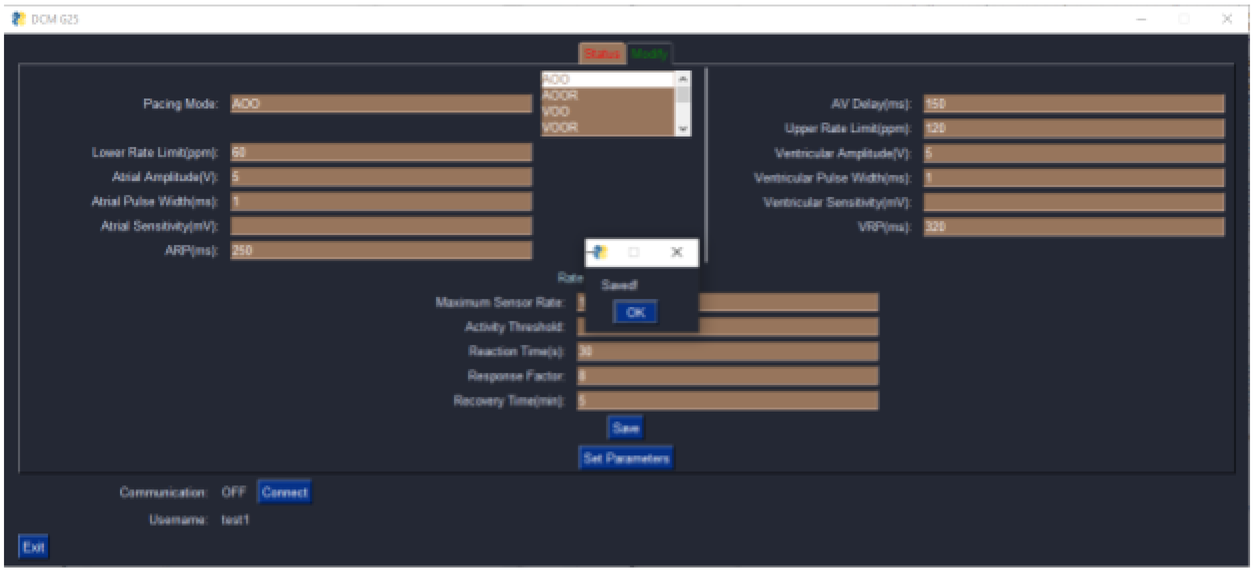


No more users can be registered.

Figure 1.3: Error message showing that max users are stored

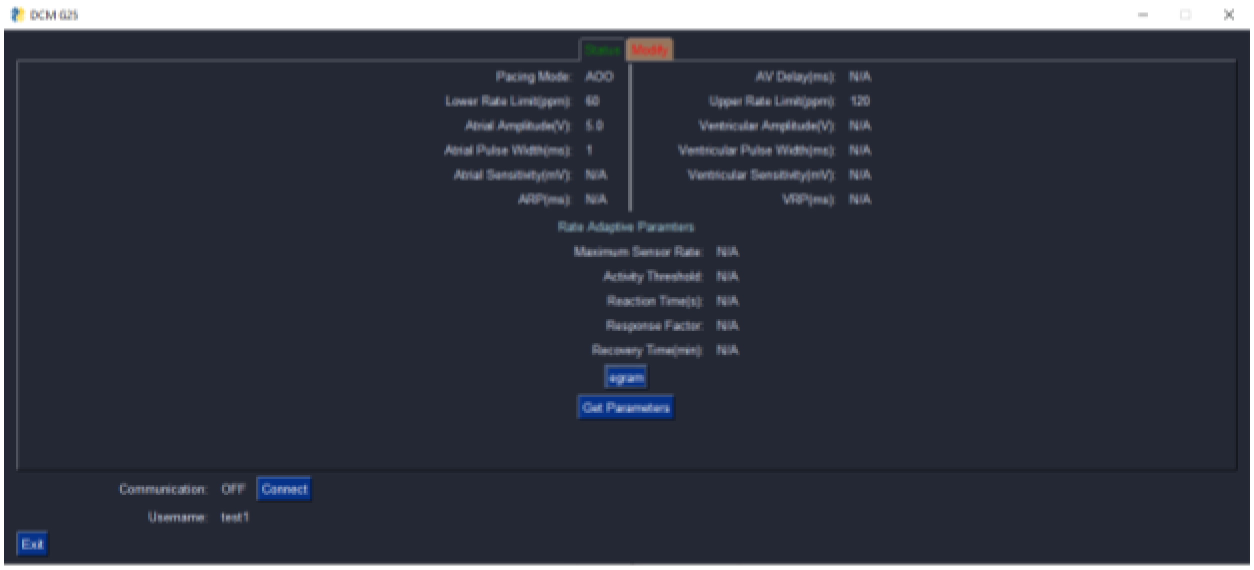
2. Active parameters

The active parameters should change according to the selected pacing mode. In this test, AOO was selected as the pacing mode and all required parameters were filled in the ‘Modify’ menu (Figure 2.1). The expected output is that in ‘Status’ menu, only the parameters as specified in Table 6: Programmable Parameters for Bradycardia Therapy Modes of the PACEMAKERS.pdf are displayed. All other values were displayed with N/A. The actual output matched, and the test was repeated for all required pacing modes. The test result is a pass.



Pacing Mode: AOO
Lower Rate Limit(ppm): 60
Atrial Amplitude(V): 5
Atrial Pulse Width(ms): 1
Atrial Sensitivity(mV):
ARP(ms): 250
AV Delay(ms): 150
Upper Rate Limit(ppm): 120
Ventricular Amplitude(V): 5
Ventricular Pulse Width(ms): 1
Ventricular Sensitivity(mV):
VRP(ms): 320
Saved!
Maximum Sensor Rate: 1
Activity Threshold:
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5
Communication: OFF
Username: test1

Figure 2.1: A pacing mode is selected and saved

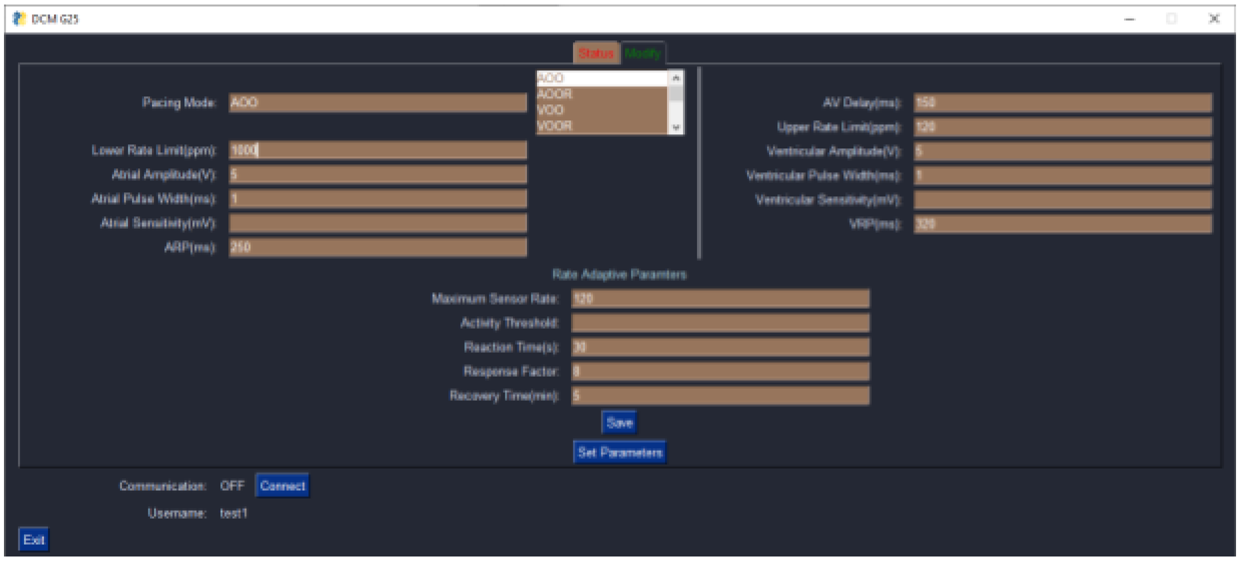


Pacing Mode: AOO
Lower Rate Limit(ppm): 60
Atrial Amplitude(V): 5.0
Atrial Pulse Width(ms): 1
Atrial Sensitivity(mV): N/A
ARP(ms): N/A
AV Delay(ms): N/A
Upper Rate Limit(ppm): 120
Ventricular Amplitude(V): N/A
Ventricular Pulse Width(ms): N/A
Ventricular Sensitivity(mV): N/A
VRP(ms): N/A
Rate Adaptive Paramters
Maximum Sensor Rate: N/A
Activity Threshold: N/A
Reaction Time(s): N/A
Response Factor: N/A
Recovery Time(min): N/A
Communication: OFF
Username: test1

Figure 2.2: Only the parameters outlined in Table 6 of the specification sheet PACEMAKER.pdf are displayed

3. Range limitations

The purpose of this test is to ensure that the user inputted value is limited to a certain range for each parameter. The range limitations of the lower rate limit was tested using the exceedingly high value of 1000 (Figure 3.1). Upon selecting 'Save', an error message should appear notifying that the input value is out of range and the change is not saved as expected (Figure 3.2). The actual result is as expected, and the test was repeated for all parameters using both high and low values. The test result is a pass.


DCM G25

Status Modify

AOO
AOOR
VOO
VOOR

Pacing Mode: AOO
Lower Rate Limit(ppm): 1000
Atrial Amplitude(V): 5
Atrial Pulse Width(ms): 1
Atrial Sensitivity(mV):
ARP(ms): 250

AV Delay(ms): 150
Upper Rate Limit(ppm): 120
Ventricular Amplitude(V): 5
Ventricular Pulse Width(ms): 1
Ventricular Sensitivity(mV):
VRP(ms): 320

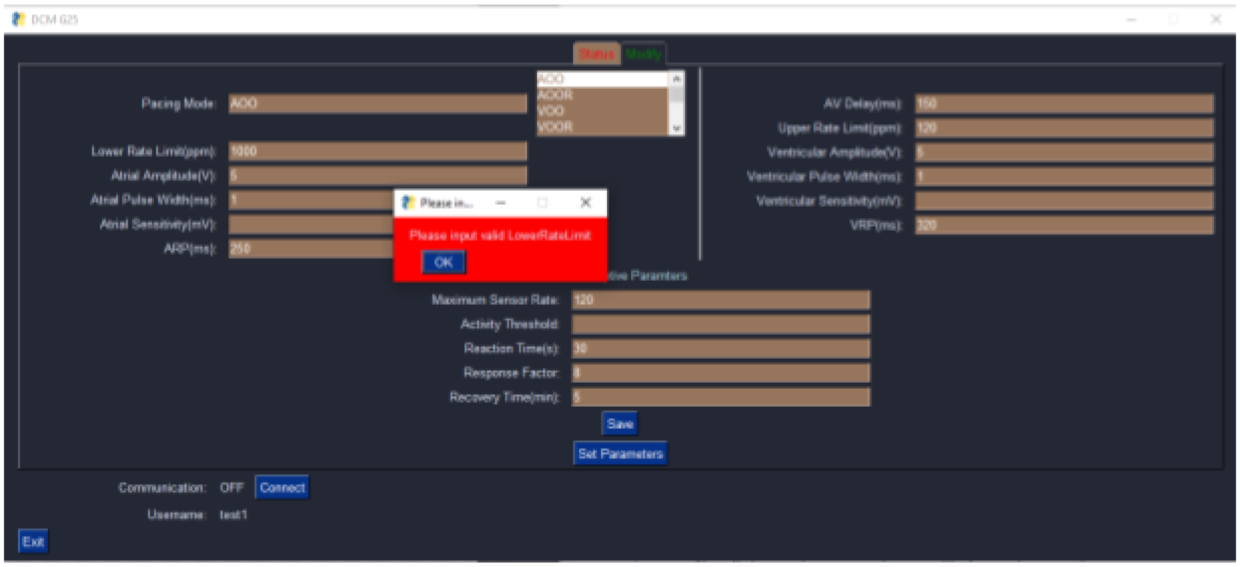
Rate Adaptive Paramters
Maximum Sensor Rate: 120
Activity Threshold:
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5

Save
Set Parameters

Communication: OFF Connect
Username: test1

Exit

Figure 3.1: Entering a value outside the expected range for a parameter


DCM G25

Status Modify

AOO
AOOR
VOO
VOOR

Pacing Mode: AOO
Lower Rate Limit(ppm): 1000
Atrial Amplitude(V): 5
Atrial Pulse Width(ms): 1
Atrial Sensitivity(mV):
ARP(ms): 250

AV Delay(ms): 150
Upper Rate Limit(ppm): 120
Ventricular Amplitude(V): 5
Ventricular Pulse Width(ms): 1
Ventricular Sensitivity(mV):
VRP(ms): 320

Please in...
Please input valid LowerRateLimit
OK

Maximum Sensor Rate: 120
Activity Threshold:
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5

Save
Set Parameters

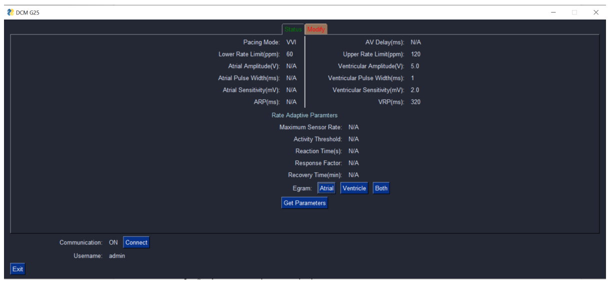
Communication: OFF Connect
Username: test1

Exit

Figure 3.2: Error message appears and the change is not saved

4. Connect with the pacemaker / Get the parameters in communication

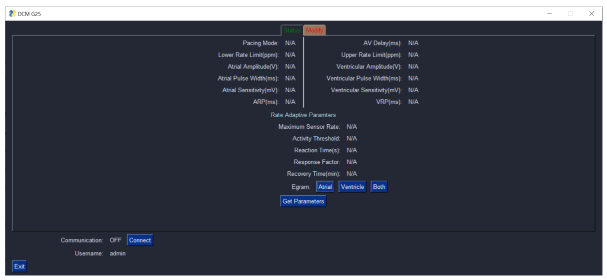
The purpose is to test whether DCM can get parameters from the pacemaker through UART. Input is the click on the connect button or get parameters button, both of which call the get_param_com function. The expected output is the current parameters showing on the status tab and the irrelevant parameters will be N/A. The actual output meets the expected one. The tests are passed for multiple trials.



Pacing Mode: VVI | AV Delay(ms): N/A
Lower Rate Limit(ppm): 60 | Upper Rate Limit(ppm): 120
Atrial Amplitude(V): N/A | Ventricular Amplitude(V): 5.0
Atrial Pulse Width(ms): N/A | Ventricular Pulse Width(ms): 1
Atrial Sensitivity(mV): N/A | Ventricular Sensitivity(mV): 2.0
ARP(ms): N/A | VRP(ms): 320

Rate Adaptive Paramters
Maximum Sensor Rate: N/A
Activity Threshold: N/A
Reaction Time(s): N/A
Response Factor: N/A
Recovery Time(min): N/A
Egram:

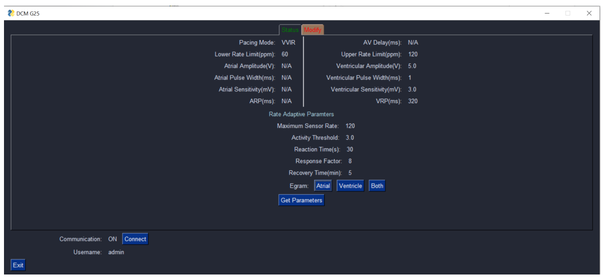
Communication: ON
Username: admin



Pacing Mode: N/A | AV Delay(ms): N/A
Lower Rate Limit(ppm): N/A | Upper Rate Limit(ppm): N/A
Atrial Amplitude(V): N/A | Ventricular Amplitude(V): N/A
Atrial Pulse Width(ms): N/A | Ventricular Pulse Width(ms): N/A
Atrial Sensitivity(mV): N/A | Ventricular Sensitivity(mV): N/A
ARP(ms): N/A | VRP(ms): N/A

Rate Adaptive Paramters
Maximum Sensor Rate: N/A
Activity Threshold: N/A
Reaction Time(s): N/A
Response Factor: N/A
Recovery Time(min): N/A
Egram:

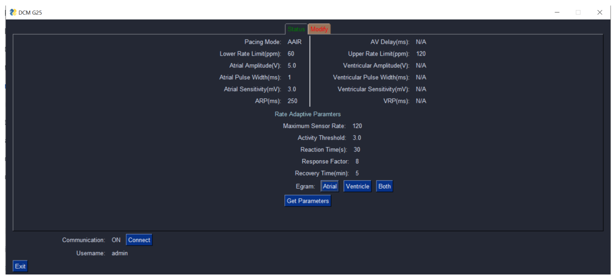
Communication: OFF
Username: admin



Pacing Mode: VVIR | AV Delay(ms): N/A
Lower Rate Limit(ppm): 60 | Upper Rate Limit(ppm): 120
Atrial Amplitude(V): N/A | Ventricular Amplitude(V): 5.0
Atrial Pulse Width(ms): N/A | Ventricular Pulse Width(ms): 1
Atrial Sensitivity(mV): N/A | Ventricular Sensitivity(mV): 3.0
ARP(ms): N/A | VRP(ms): 320

Rate Adaptive Paramters
Maximum Sensor Rate: 120
Activity Threshold: 3.0
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5
Egram:

Communication: ON
Username: admin



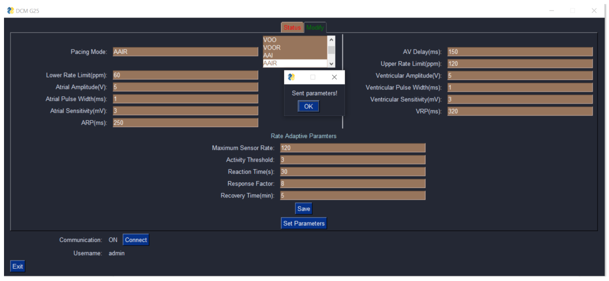
Pacing Mode: AAIR | AV Delay(ms): N/A
Lower Rate Limit(ppm): 60 | Upper Rate Limit(ppm): 120
Atrial Amplitude(V): 5.0 | Ventricular Amplitude(V): N/A
Atrial Pulse Width(ms): 1 | Ventricular Pulse Width(ms): N/A
Atrial Sensitivity(mV): 3.0 | Ventricular Sensitivity(mV): N/A
ARP(ms): 250 | VRP(ms): N/A

Rate Adaptive Paramters
Maximum Sensor Rate: 120
Activity Threshold: 3.0
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5
Egram:

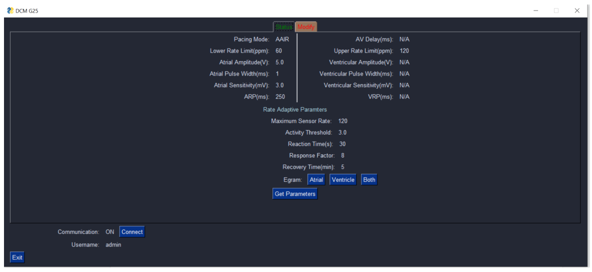
Communication: ON
Username: admin

5. Set the parameters in communication

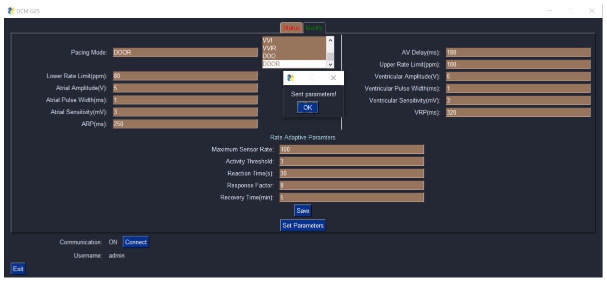
The purpose of this test is to make sure the parameters saved in DCM can be sent to the pacemaker successfully. The input is the click in the "Set Parameters" button in the modify tab. The expected output is a pop up window indicating that it was sent successfully. The actual output meets the expected one. All tests passed. It can be checked by clicking the "get parameters" button in the status tab after sending the parameters.



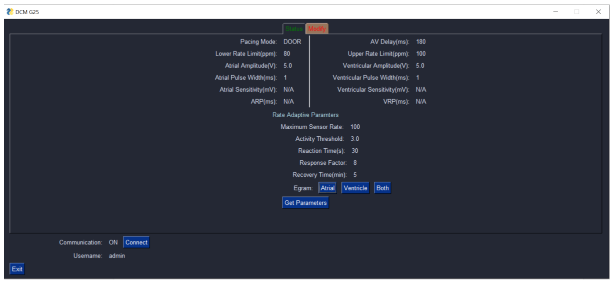
Pacing Mode: AAIR
Lower Rate Limit(ppm): 60
Atrial Amplitude(V): 5
Atrial Pulse Width(ms): 1
Atrial Sensitivity(mV): 3
ARP(ms): 250
AV Delay(ms): 150
Upper Rate Limit(ppm): 120
Ventricular Amplitude(V): 5
Ventricular Pulse Width(ms): 1
Ventricular Sensitivity(mV): 3
VRP(ms): 320
Sent parameters!
Rate Adaptive Paramters
Maximum Sensor Rate: 120
Activity Threshold: 3
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5
Communication: ON
Username: admin



Pacing Mode: AAIR
Lower Rate Limit(ppm): 60
Atrial Amplitude(V): 5.0
Atrial Pulse Width(ms): 1
Atrial Sensitivity(mV): 3.0
ARP(ms): 250
AV Delay(ms): N/A
Upper Rate Limit(ppm): 120
Ventricular Amplitude(V): N/A
Ventricular Pulse Width(ms): N/A
Ventricular Sensitivity(mV): N/A
VRP(ms): N/A
Rate Adaptive Paramters
Maximum Sensor Rate: 120
Activity Threshold: 3.0
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5
Egram:
Communication: ON
Username: admin



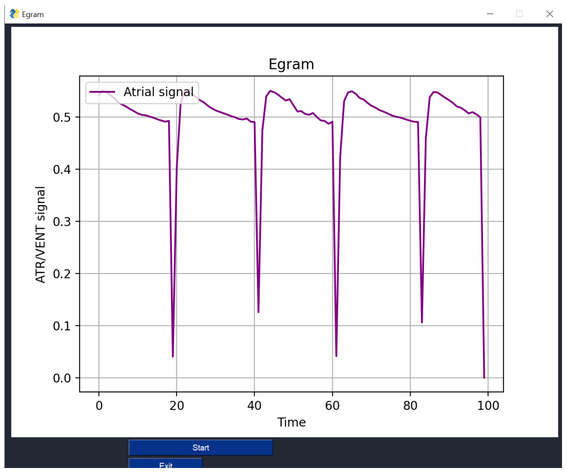
Pacing Mode: DOOR
Lower Rate Limit(ppm): 80
Atrial Amplitude(V): 5
Atrial Pulse Width(ms): 1
Atrial Sensitivity(mV): 3
ARP(ms): 250
AV Delay(ms): 180
Upper Rate Limit(ppm): 100
Ventricular Amplitude(V): 5
Ventricular Pulse Width(ms): 1
Ventricular Sensitivity(mV): 3
VRP(ms): 320
Sent parameters!
Rate Adaptive Paramters
Maximum Sensor Rate: 100
Activity Threshold: 3
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5
Communication: ON
Username: admin

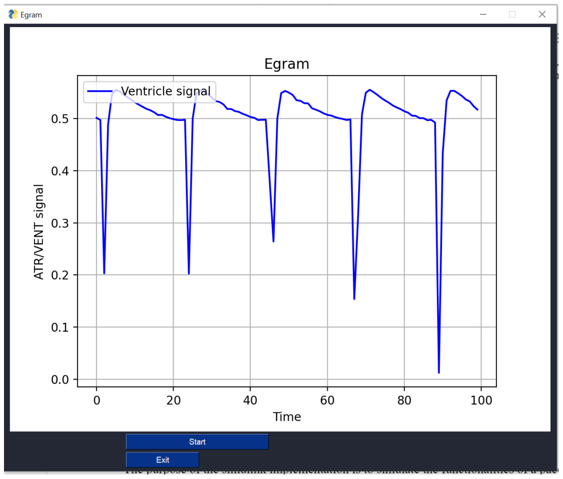


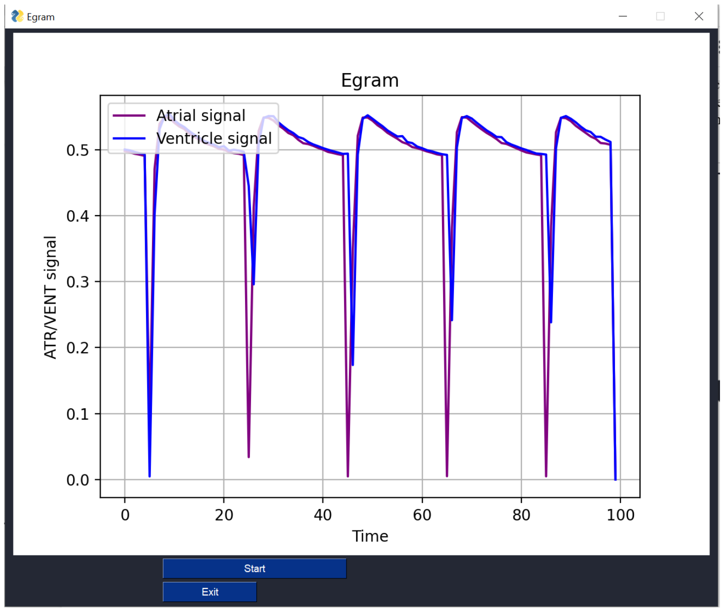
Pacing Mode: DOOR
Lower Rate Limit(ppm): 80
Atrial Amplitude(V): 5.0
Atrial Pulse Width(ms): 1
Atrial Sensitivity(mV): N/A
ARP(ms): N/A
AV Delay(ms): 180
Upper Rate Limit(ppm): 100
Ventricular Amplitude(V): 5.0
Ventricular Pulse Width(ms): 1
Ventricular Sensitivity(mV): N/A
VRP(ms): N/A
Rate Adaptive Paramters
Maximum Sensor Rate: 100
Activity Threshold: 3.0
Reaction Time(s): 30
Response Factor: 8
Recovery Time(min): 5
Egram:
Communication: ON
Username: admin

6. Egram display

This test is aimed to verify the egram of the pacemaker can be shown correctly. The input is clicking either atrial, ventricle or both, then clicking the start button in the egram window. The expected output is the animated graph in the egram window showing the signals from atrium and/or ventricles in real time. The actual output meets the expected one. The tests are all passed.







1.6 References

1. Lab tutorials
2. PACEMAKER System Specification
3. 3K04 Requirements Specification of the Pacemaker
4. PySimpleGUI official documentation
5. Matplotlib official documentation