SFWRENG 3K04

Project Documentation

Group 1: Tioluwalayomi Babayeju, Gowri Gowda, Alanjim Halabia, Nicholas Levantis, Yazdan Rahman

Updated: November 29 2020

Table of Contents

0.0 - Summary

The purpose of this project is to develop a functional working pacemaker with the knowledge of Simulink to create a real-time program based in model-driven design that will run on selected hardware. In this particular project, the goal is to create a working model of a pacemaker with multiple pacing modes and a user interface for access to settings.

A pacemaker is an electrical device that is surgically implanted in the chest or abdomen of a patient to control irregular heart rhythms known as arrhythmia. Pacemakers are used to treat and control both tachycardia (heart beating too fast) and bradycardia (heart beating too slow). Arrhythmia is a serious and frequently life-threatening condition that can cause symptoms from fatigue to shortness of breath to loss of consciousness or death. A functional pacemaker not only prolongs the life of the patient but allows them to resume a more active lifestyle closer to that of an able-bodied person. As such, pacemakers are a safety-critical system with a low margin of error.

Pacemakers use low-level electrical pulses to stimulate the tissues of the heart into contracting and relaxing as a healthy heart would. There are several standardized pacing modes based on what form of arrhythmia the patient has. These include but are not limited to: AOO, VOO, AAI, VVI, etc. Each mode will be elaborated on as necessary further in the document.

In this project, Simulink will be used to develop a variety of pacing modes. These modes will be set and controlled using a proprietary user interface developed in tandem by the same team. The software will then be flashed onto standardized hardware for model testing purposes. Hardware details can be found in *pacemaker_shield_explained* [2].

1.0 - Pacemaker Design

The objective was to implement stateflows a total of ten pacemaker modes using Simulink.[1] Previously, only AOO, VOO, AAI, and VVI were implemented. The six additional pacing modes are DOO, AOOR, VOOR, AAIR, VVIR, and DOOR.

It should be noted that all states ending in the letter R address issues of chronotropic incompetence. Chronotropic incompetence (CI) is loosely defined as the inability of the heart to increase the heart rate based on activity or demand. This impairs quality of life as it causes exercise intolerance and increases risk of major adverse cardiac events. To combat this serious issue, rate-adaptive features are added to the pacing mode.

1.1 - Requirements

The requirements can be found in *PACEMAKER System Specification* [3] and *srsVVI rev 2* [6] as cited in *4.1 - Works Cited*. The requirements are expanded on and clarified in the following sections.

The following tables of variables apply to *1.1.0 - AOO Requirements*, *1.1.1 - VOO Requirements*, *1.1.2 - AAI Requirements,* and *1.1.3 - VVI Requirements*.

Table 1.0 - Monitored Variables:

Name	**Units/Type**	**Device**	**Range**	**Description**
Heartrate	int32	-	1-10000	Changes the pacing of the pacemaker
Amp	int32	-	0-100	Changes the PWM duty cycle

[1] Mistakes have been edited and replaced within *1.0 - Pacemaker Design (Assignment 1 - Part 1)*; refer to the previous version of *Project Documentation* to compare and contrast.

VENT_CMP_ DETECT	boolean	PIN D1	{true, false}	Sensing Circuit for heart ventricle
ATR_CMP_ DETECT	boolean	PIN D0	{true, false}	Sensing Circuit for heart atrium

Table 1.1 - Controlled Variables:

Name	**Units/Type**	**Device**	**Range**	**Description**
PACE_CHARGE_CTRL	boolean	PIN D2	{true, false}	Allows charge to go through the C22 capacitor
VENT_CMP_REF_PWM	boolean	PIN D3	{true, false}	Establishes a threshold for the Ventricle PWM
Z_ATR_CTRL	boolean	PIN D4	{true, false}	Impedes the flow of electrons electrode of the atrium
PACING_REF_PWM	int32	PIN D5	0-100.0	Used to charge the primary capacitor of pacing circuit
ATR_CMP_REF_PWM	boolean	PIN D6	{true, false}	Establishes a threshold for the Atrial PWM
Z_VENT_CTRL	boolean	PIN D7	{true, false}	Impedes the flow of electrons ventricle of the atrium
ATR_PACE_CTRL	boolean	PIN D8	{true, false}	Discharges primary capacitor into atrium
VENT_PACE_CTRL	boolean	PIN D9	{true, false}	Discharges primary capacitor into ventricle
PACE_GND_CTRL	boolean	PIN D10	{true, false}	Allows current to flow through either atrium or ventricle from the ring
ATR_GND_CTRL	boolean	PIN D11	{true, false}	Used to connect ATR_RING_OUT to ground

VENT_GND_CTRL	boolean	PIN D12	{true, false}	Used to connect VENT_RING_OUT to ground
FRONTEND_CTRL	boolean	PIN D13	{true, false}	Connects the pacemaker to the sensing circuitry
LEDR	boolean	PIN RED_LED	{true, false}	Light that turns red
LEDB	boolean	PIN BLUE_LED	{true, false}	Light that turn blue

1.1.0 - AOO Requirements

AOO is an atrial pacing mode. It has no sensor input. AOO is atrial synchronous pacing at a lower programmed pacing rate.

Table 1.1.0.0 - Tabular Expression of AOO Requirements:

Source State	**Event**	**Condition**	**During Actions**	**Condition Actions**	**Exit Actions**	**Transition Actions**	**Destination State**	**Entry Actions**
State == START	NONE	NONE	NC	NC	NC	NC	Charging_C22	PACE_CHARGE_CTRL=true; Z_ATR_CTRL=false; PACING_REF_PWM=80; Z_VENT_CTRL=true; ATR_PACE_CTRL=false; VENT_PACE_CTRL=false; PACE_GND_CTRL=true; ATR_GND_CTRL=true; VENT_GND_CTRL=false; LEDR=true; LEDB=false;
State == Charging_C22	NONE	NONE	NC	NC	NC	after(Heartrate -10, msec)	Discharging_C22	PACE_CHANGE_CTRL=true; ATR_PACE_CTRL=true; ATR_GND_CTRL=false; LEDR=true; LEDG=false;
State == Discharging_C22	NONE	NONE	NC	NC	NC	after(10, msec)	Charging_C22	PACE_CHARGE_CTRL=true; Z_ATR_CTRL=false; PACING_REF_PWM=80; Z_VENT_CTRL=true; ATR_PACE_CTRL=false;

								VENT_PACE_CTRL=false;
								PACE_GND_CTRL=true;
								ATR_GND_CTRL=true;
								VENT_GND_CTRL=false;
								LEDR=true;
								LEDB=false;

Note that underlined values are subject to change. See 1.2.0.2 - Planned Future Changes for further details.

1.1.1 - VOO Requirements

VOO is a ventricular pacing mode. It has no sensor input. VOO is ventricular asynchronous pacing at a lower programmed pacing rate.

Table 1.1.1.0 - Tabular Expression of VOO Requirements:

Source State	**Event**	**Condition**	**During Actions**	**Condition Actions**	**Exit Actions**	**Transition Actions**	**Destination State**	**Entry Actions**

State == START	NONE	NONE	NC	NC	NC	NC	Charging_ C22	PACE_CHARGE_CTRL=true;
								Z_ATR_CTRL=false;
								PACING_REF_PWM=<u>60</u>
								Z_VENT_CTRL=false;
								ATR_PACE_CTRL=false;
								PACE_GND_CTRL=true;
								ATR_GND_CTRL=false;
								VENT_GND_CTRL=true;
								LEDR=false;
								LEDB=true;

State == Charging_ C22	NONE	NONE	NC	NC	NC	after(Heartrate-1, msec)	Discharging_C22	PACE_CHARGE_CTRL=false;
								VENT_PACE_CTRL=true;
								VENT_GND_CTRL=false;
								LEDR=true;
								LEDB=false;

State == Discharging_C22	NONE	NONE	NC	NC	NC	after(1, msec)	Charging_ C22	PACE_CHARGE_CTRL=true;
								Z_ATR_CTRL=false;
								PACING_REF_PWM=<u>60</u>
								Z_VENT_CTRL=false;
								ATR_PACE_CTRL=false;
								PACE_GND_CTRL=true;
								ATR_GND_CTRL=false;
								VENT_GND_CTRL=true;
								LEDR=false;

								LEDB=true;

Note that underlined values are subject to change. See 1.2.0.2 - Planned Future Changes for further details.

1.1.2 - AAI Requirements

AAI is an atrial pacing mode. It has a sensor input. In AAI, the intrinsic P wave inhibits atrial pacing.

Table 1.1.2.0 - Tabular Expression of AAI Requirement:

Source State	**Event**	**Condition**	**During Actions**	**Condition Actions**	**Exit Actions**	**Transition Actions**	**Destination State**	**Entry Actions**
State == START	TRUE	ATR_CMP_DETECT==false	NC	NC	NC	NC	Charging_C22	PACE_CHARGE_CTRL=true; VENT_CMP_REF_PWM=false; Z_ATR_CTRL=false; PACING_REF_PWM=Amp; ATR_CMP_REF_PWM=<u>70</u>; ATR_PACE_CTRL=false; VENT_PACE_CTRL=false; PACE_GND_CTRL=true; ATR_GND_CTRL=true; VENT_GND_CTRL=false; FRONTEND_CTRL=true; LEDR=true; LEDB=false;
		ATR_CMP_DETECT==TRUE	NC	NC	NC	NC	Discharging_C22	PACE_CHANGE_CTRL=false; Z_ATR_CTRL=true; Z_VENT_CTRL=false; ATR_PACE_CTRL=true; VENT_PACE_CTRL=false; ATR_GND_CTRL=false; FRONTEND_CTRL=true; LEDR=false; LEDB=true;
State == Charging_C22	NONE	NONE	NC	NC	NC	after(1, msec)	START	NC
State == Dischargin	TRUE	NONE	NC	NC	NC	after(1, msec)	Charging_C22	PACE_CHARGE_CTRL=true; VENT_CMP_REF_PWM=false;

g_C22								Z_ATR_CTRL=false;
								PACING_REF_PWM=Amp;
								ATR_CMP_REF_PWM=<u>70</u>;
								ATR_PACE_CTRL=false;
								VENT_PACE_CTRL=false;
								PACE_GND_CTRL=true;
								ATR_GND_CTRL=true;
								VENT_GND_CTRL=false;
								FRONTEND_CTRL=true;
								LEDR=true;
								LEDB=false;

Note that underlined values are subject to change. See 1.2.0.2 - Planned Future Changes for further details.

1.1.3 - VVI Requirements

VVI is a ventricular pacing mode. It has a sensor input. In VVI, the sensed intrinsic QRS inhibits ventricular pacing.

Table 1.1.3.0 - Tabular Expression of VVI Requirements:

Source State	**Event**	**Condition**	**During Actions**	**Condition Actions**	**Exit Actions**	**Transition Actions**	**Destination State**	**Entry Actions**

State == START	TRUE	VENT_CMP_DETECT == true	NC	NC	NC	NC	Charging_C22	PACE_CHARGE_CTRL = true;
								VENT_CMP_REF_PWM = <u>60</u>;
								Z_ATR_CNTRL=false;
								PACING_REF_PWM=<u>60</u>;
								ATR_CMP_REF_PWM=false;
								Z_VENT_CTRL=false;
								ATR_PACE_CTRL=false;
								VENT_PACE_CTRL=false;
								PACE_GND_CTRL=true;
								ATR_GND_CTRL=false;
								VENT_GND_CTRL=true;
								FRONTEND_CTRL=true;
								LEDR=true;
								LEDB=false;

		VENT_CMP_DETECT == false	NC	NC	NC	after(999, msec)	Discharging_C22	PACE_CHANGE_CTRL=false;
								Z_ATR_CTRL=false;
								Z_VENT_CTRL=false;
								ATR_PACE_CTRL=false;
								VENT_PACE_CTRL=true;
								FRONTEND_CTRL=false;

								PACE_GND_CTRL=true;
								LEDR=true;
								LEDB=false;

State == Charging _C22	NONE	NONE	NC	NC	NC	NC	START	NC

State == Discharging_C22	TRUE	NONE	NC	NC	NC	after(1, msec)	Charging_C 22	PACE_CHARGE_CTRL = true;
								VENT_CMP_REF_PWM = 60;
								Z_ATR_CNTRL=false;
								PACING_REF_PWM=60;
								ATR_CMP_REF_PWM=false;
								Z_VENT_CTRL=false;
								ATR_PACE_CTRL=false;
								VENT_PACE_CTRL=false;
								PACE_GND_CTRL=true;
								ATR_GND_CTRL=false;
								VENT_GND_CTRL=true;
								FRONTEND_CTRL=true;
								LEDR=true;
								LEDB=false;

Note that underlined values are subject to change. See 1.2.0.2 - Planned Future Changes for further details.

1.1.4 - DOO Requirements

DOO is a dual chamber non-tracking pacing mode. In DOO, both the atrium and the ventricle are paced. The intrinsic P wave and QRS do not affect the pacing of the heart. It is an asynchronous pacing mode; it always paces at a lower pacing rate.

Table 1.1.4.0 - Tabular Expression of DOO Requirements:

Source State	**Event**	**Condition**	**During Actions**	**Condition Actions**	**Exit Actions**	**Transition Actions**	**Destination State**	**Entry Actions**
State ==	**True**	**ATR_PACING=T**	**NC**	**NC**	**NC**	**NC**	DUAL_CHARGING_	**VENT_GND_CTRL = false;**

START		**RUE**					AND_DISCHARGE	**ATR_GND_CTRL = true;**

	None	**ATR_PACING=FALSE**	**NC**	**NC**	**NC**	**after(1, msec)**	VENT_CHARGING_AND_DISCHARGE ==	**entry:
								% pace atrium
								PACE_CHARGE_CTRL=false;
								PACE_GND_CTRL = true;
								VENT_PACE_CTRL = false;
								VENT_GND_CTRL = false;
								Z_VENT_CTRL = false;
								Z_ATR_CTRL = false;
								ATR_GND_CTRL = false;
								ATR_PACE_CTRL = true;
								FRONTEND_CTRL = sensing;**

State == DUAL_CHARGING_AND_DISCHARGE	**True**	**VENT_PACING=TRUE**	**NC**	**NC**	**NC**	**after (30, msec)**	VENT_CHARGING_AND_DISCHARGE ==	
State == VENT_CHARGING_AND_DISCHARGE	**False**	**VENT_PACING=TRUE**	**NC**	**NC**	**NC**	**after (1, msec)**	DUAL_CHARGING_AND_DISCHARGE ==	

1.1.5 - Rate-Adaptive States

1.1.5.0 - AOOR Requirements

The requirements for pacing are the same as *1.1.0 - AOO Requirements* except for the addition of the rate-adaptive threshold for setting the pace.

1.1.5.1 - VOOR Requirements

The requirements for pacing are the same as *1.1.1 - VOO Requirements* except for the addition of the rate-adaptive threshold for setting the pace.

1.1.5.2 - AAIR Requirements

The requirements for pacing are the same as *1.1.2 - AAI Requirements* except for the addition of the rate-adaptive threshold for setting the pace.

1.1.5.3 - VVIR Requirements

The requirements for pacing are the same as *1.1.3 - VVI Requirements* except for the addition of the rate-adaptive threshold for setting the pace.

1.1.5.4 - DOOR Requirements

The requirements for pacing are the same as *1.1.4 - DOO Requirements* except for the addition of the rate-adaptive threshold for setting the pace.

Table 1.1.5.0 - Tabular Expression of Rate Adaptive Mode:

Source State	**Event**	**Condition**	**During Actions**	**Condition Actions**	**Exit Actions**	**Transition Actions**	**Destination State**	**Entry Actions**
State ==	NONE	NONE	NC	NC	NC	NC	Rate_initial	LRL1=LRL;

START								
State == Rate_initial	TRUE	rateAdap == 0	NC	NC	NC	NC	initial	LRL1=LRL;
	TRUE	avg_acc>threshold && [avg_acc>threshold &&avg_acc<1.4]	NC	NC	NC	NC	low	new_rate = 90;
		avg_acc>threshold && [avg_acc>=1.4&&avg_acc<2.0]	NC	NC	NC	NC	medium	new_rate = 120;
		avg_acc>threshold && [avg_acc>=2.0]	NC	NC	NC	NC	high	New_rate = MSR;
		avg_acc<=threshold && [Old_LRL >LRL]	NC	NC	NC	NC	low_rate	Recovery_time1=single(Recovery_time)/single(rateDec);
		avg_acc<=threshold && [Old_LRL <=LRL]	NC	NC	NC	NC	Name1	Change_value = 1;
Start == low	TRUE	new_rate>Old_LRL	NC	NC	NC	NC	same	
		new_rate<=Old_LRL	NC	NC	NC	NC	avg_rate	rateInc=new_rate-LRL; LRL1=Old_LRL + rateInc/respFactor; Reaction_time1=(reactTime)/(respFactor); change_value=1;

State == low_rate	TRUE	change_value == 1	NC	NC	NC	NC	calculation	change_value=0; rateDec=(LRL1-LRL);
	TRUE	rateAdap == 0	NC	NC	NC	NC	initial	LRL1 = LRL;
	NONE	NC	NC	NC	NC	after[Recovery_time1, sec]	rate_initial	Old_LRL = LRL1; Recovery_time = recovTime*60;

1.2 - Design Decisions

1.2.0 - Assignment 1

1.2.0.0 - Initial Plan

The project was initially divided into five independent sections that would be integrated once fully developed. The development of each section would be headed by an individual member of the group. While collaboration was encouraged between members, the expectation was that the sections would be largely completed independently. The five sections were the AOO stateflow, the VOO stateflow, the AAI stateflow, the VVI stateflow, and the Device-Control Monitor (DCM), also known as the user interface. The DCM is discussed in *2.0 - DCM Design (Assignment 1 - Part 2)*.

When the initial plan was presented to the advising teaching assistant, no pressing issues were flagged. It was noted that this division of labour was unorthodox and thus may present unforeseen issues. However, it was ultimately decided that the group would proceed with this plan due to the flexibility it would allow in a difficult virtual environment with scheduling conflicts.

The original timeline allowed for one week for individual section development, one week for module integration, and one week for hardware hiding. No further specifics were decided in order to adapt to the changing needs of the project.

1.2.0.1 - Executed Design Plan

In practice, several elements of the initial plan were altered or replaced to better suit the reality of the project. The most drastic change was that the modular sectioning and integration was scrapped in favor of collaboratively working on each element. From October 12 2020 to October 16 2020, daily meetings were held where members would discuss the theory behind the project to choose the most appropriate path to develop the project. In this way, all four stateflows were developed. Due to this change in procedure, there is no more need for a dedicated integration step.

1.2.0.1.0 - Reasons for Changes in Plan

As stated in *1.2.0*, the initial “divide-and-conquer” strategy was created for ease of development while connecting using a virtual remote environment. This strategy reduced the amount of group meetings required and was thus more convenient during a time with varied schedules and locations. However, there were downsides that had not been considered.

Since a large part of the purpose of this project is exploratory experimental learning, no members had prior knowledge of the tools being used. As such, a large part of the time spent independently was used on familiarizing team members with the tools required by the project. This was a necessary investment towards project development and skill acquisition but spending more time working independently would have been inefficient. By coming together to create the state flows together, the group members were able to establish a more robust shared foundation of information. Not only did this allow all members to gain equal technical skill, but it allowed ideas to be examined and refined from all angles.

Furthermore, by virtue of the design specifications, there was a high degree of code reusability between the stateflows. By working on the stateflows consecutively as a group, these patterns were easily identified and utilized for greatest efficiency. In this way, a great deal of redundant labour was avoided.

1.2.0.2 - Planned Future Changes

All design decisions were made to the best of the team's current ability. Future development will be primarily to do with adapting the existing work to achieve the goals of follow-up assignments. As such, it is hard to predict what changes will be made. However, there are a few changes that can be easily predicted.

It is known that the additional pacing mode DOO will be added to the system. As of now, the plan is to develop DOO using the same collaborative method used to develop AOO, VOO, AAI, and VVI.

It is also known that the timing cycles AOOR, VOOR, AAIR, VVIR, and DOOR will be added to future iterations of the project. Since the collaborative system was chosen for its resilience and efficiency, it is also the planned course of action to develop the timing cycles.

The values underlined in *Table 1.1.0.0*, *Table 1.1.1.0*, *Table 1.1.2.0*, and *Table 1.1.3.0* should instead be altered using parameters within a subsystem.

1.2.0.2.0 - Mistakes to Rectify

As the four pacing modes are not united within one subsystem, they do not have common controlled variables. This means that parameters must be changed within the stateflow. This is risky (greater risk of accidentally modifying critical code) and inconvenient (requires the user to access the back-end code). In future iterations, a subsystem will be used to create common inputs and standardized outputs for the system.

The controlled variable Heartrate is currently in units of milliseconds between beats. This is not user-friendly as beats per minute is the standard way of communicating a human's heart rate. In future iterations, Heartrate will be entered into the system in beats per minute and converted to milliseconds internally.

1.2.1 - Assignment 2

1.2.1.0 - Initial Plan

The initial design plan was based on 1.2.1 - Executed Plan due to its success during Assignment 1. Namely, it was expected that the new stateflows would be collaboratively developed from scratch and integrated into the existing system.
Furthermore, it was planned to rectify all mistakes from 1.2.2.0 - Mistakes to Rectify. The most pressing issue from the referred section was the addition of a subsystem to alter the parameters from.
The plan was not developed in any further detail since it was likely that it would be changed to accommodate obstacles.

1.2.1.1 - Executed Plan

Instead of developing the stateflows from the ground up as initially envisioned, the existing AOO, VOO. AAI, and VVI stateflows were replicated and altered to include the rate-adaptive threshold functionality. This excludes DOO, which was independently developed, and DOOR, which was adapted from DOO.

1.2.1.1.0 - Reasons for Changes

Unlike 1.2.1.0 - Reasons for Changes, the changes were implemented to prevent redundant work instead of to redirect inefficient efforts. By identifying common elements between the pacing modes, the associated code was able to be reused.

1.3 - Annotated Simulink Diagrams

1.3.0 - AOO Stateflow

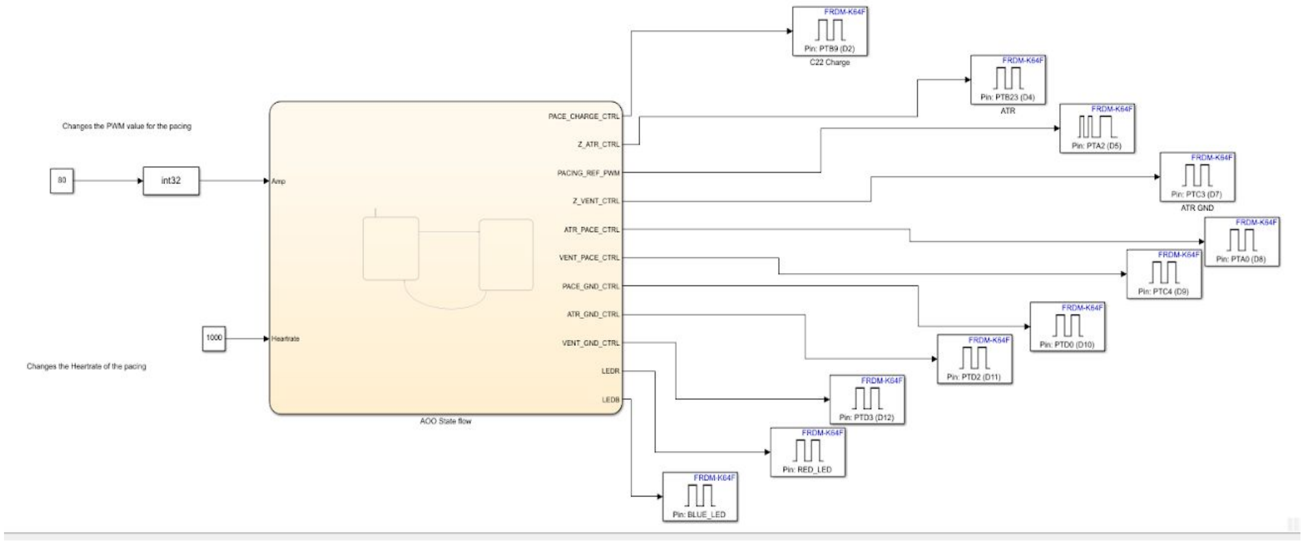
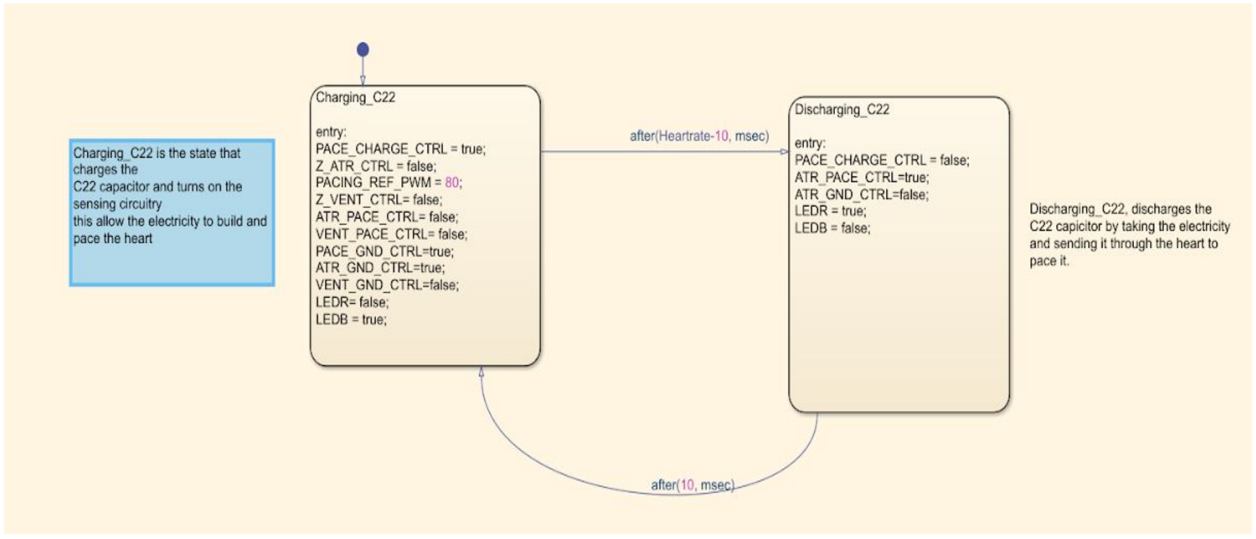


Figure 1.3.0.0 - Overall AOO Simulink Function



Charging_C22 is the state that charges the C22 capacitor and turns on the sensing circuitry this allow the electricity to build and pace the heart

```
Charging_C22

entry:
PACE_CHARGE_CTRL = true;
Z_ATR_CTRL = false;
PACING_REF_PWM = 80;
Z_VENT_CTRL= false;
ATR_PACE_CTRL= false;
VENT_PACE_CTRL= false;
PACE_GND_CTRL=true;
ATR_GND_CTRL=true;
VENT_GND_CTRL=false;
LEDR= false;
LEDB = true;
```

```
Discharging_C22

entry:
PACE_CHARGE_CTRL = false;
ATR_PACE_CTRL=true;
ATR_GND_CTRL=false;
LEDR = true;
LEDB = false;
```

Discharging_C22, discharges the C22 capicitor by taking the electricity and sending it through the heart to pace it.

Figure 1.3.0.1 - AOO Stateflow

1.3.1 - VOO Stateflow

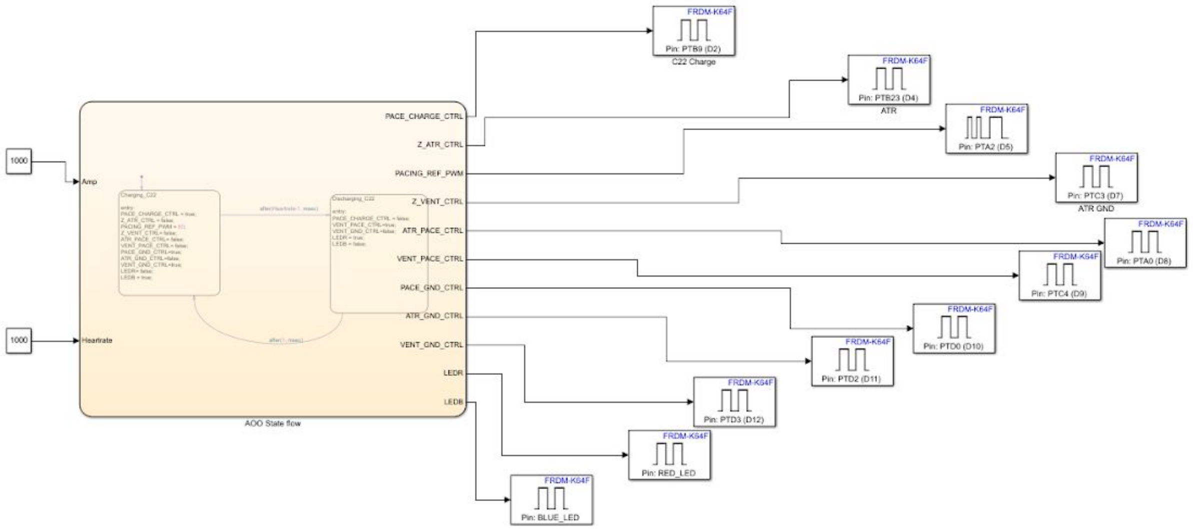
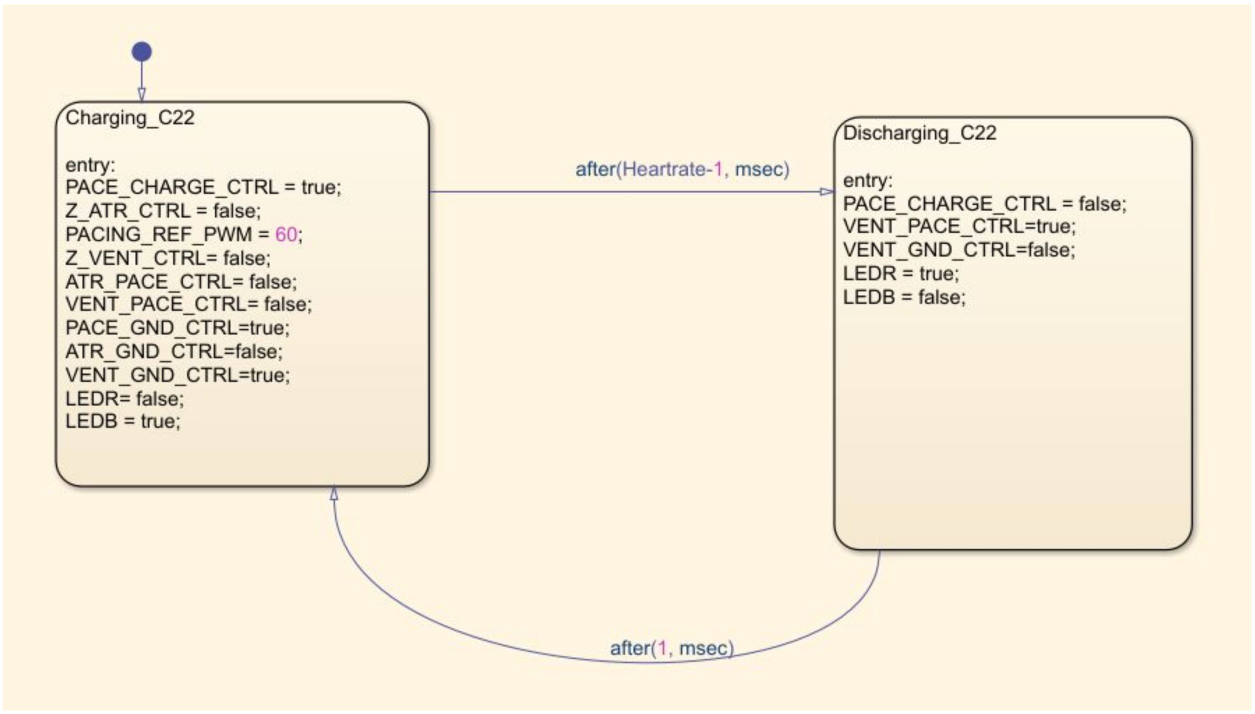


Figure 1.3.1.0 - Overall VOO Simulink Function



```
Charging_C22

entry:
PACE_CHARGE_CTRL = true;
Z_ATR_CTRL = false;
PACING_REF_PWM = 60;
Z_VENT_CTRL= false;
ATR_PACE_CTRL= false;
VENT_PACE_CTRL= false;
PACE_GND_CTRL=true;
ATR_GND_CTRL=false;
VENT_GND_CTRL=true;
LEDR= false;
LEDB = true;
```

```
Discharging_C22

entry:
PACE_CHARGE_CTRL = false;
VENT_PACE_CTRL=true;
VENT_GND_CTRL=false;
LEDR = true;
LEDB = false;
```

Figure 1.3.1.1 - VOO Stateflow

1.3.2 - VVI Stateflow

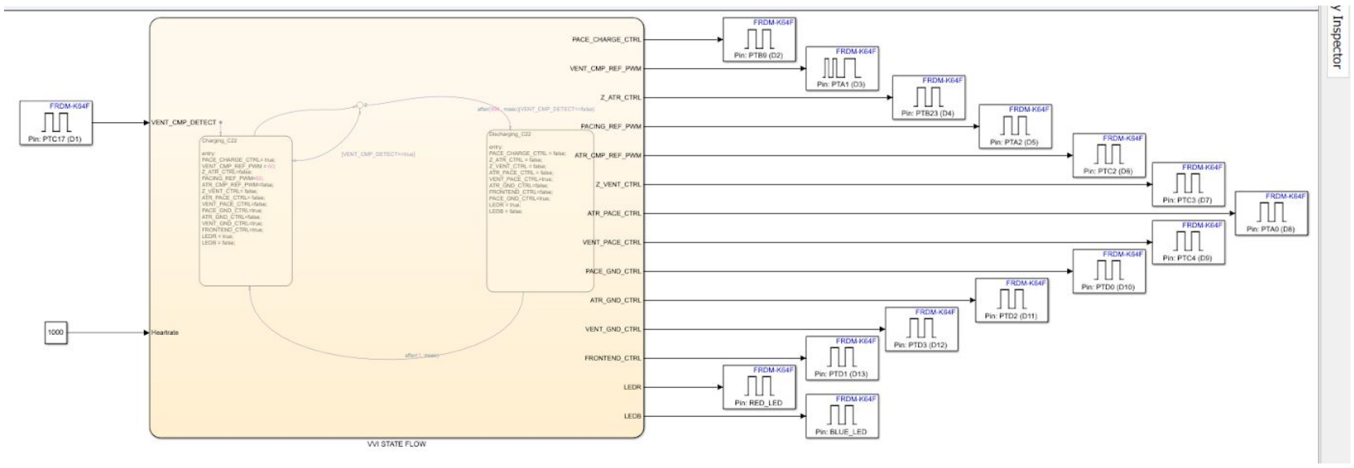
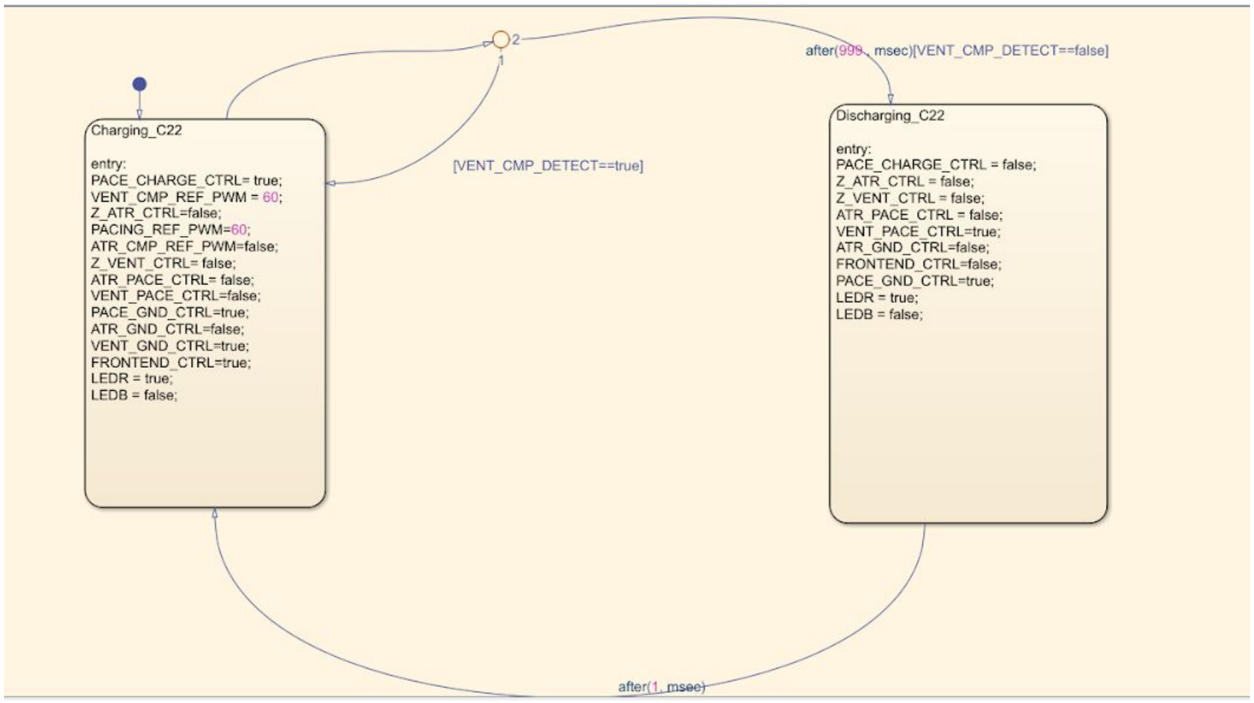


Figure 1.3.2.0 - Overall VVI Simulink Function

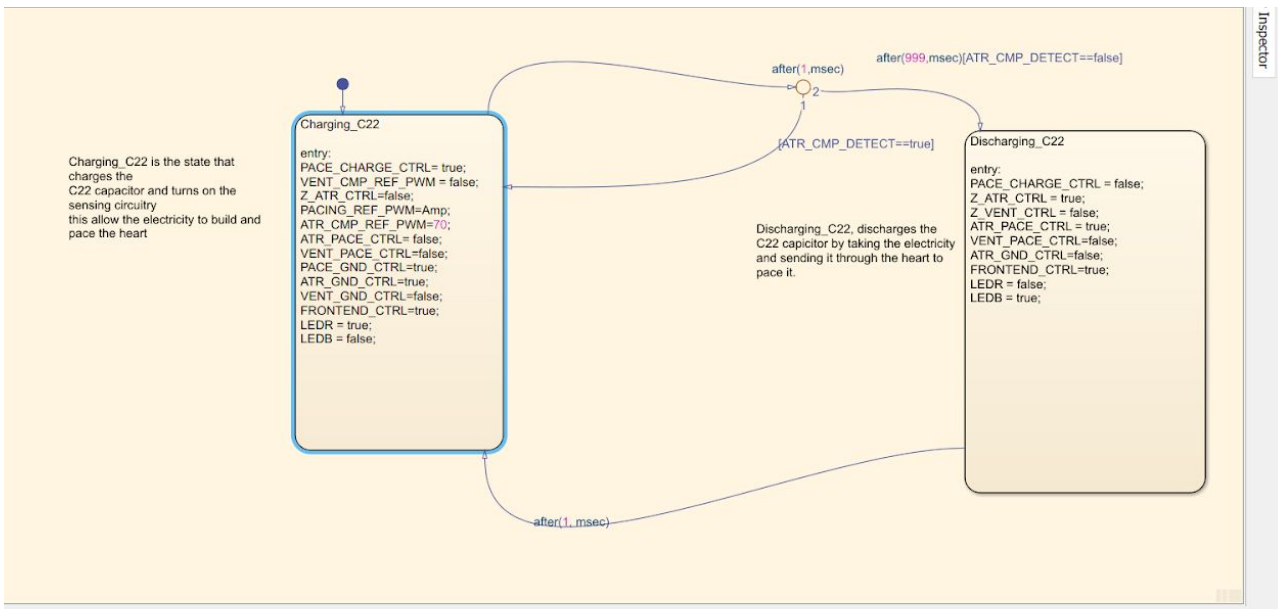


```
Charging_C22

entry:
PACE_CHARGE_CTRL= true;
VENT_CMP_REF_PWM = 60;
Z_ATR_CTRL=false;
PACING_REF_PWM=60;
ATR_CMP_REF_PWM=false;
Z_VENT_CTRL= false;
ATR_PACE_CTRL= false;
VENT_PACE_CTRL=false;
PACE_GND_CTRL=true;
ATR_GND_CTRL=false;
VENT_GND_CTRL=true;
FRONTEND_CTRL=true;
LEDR = true;
LEDB = false;
```

```
Discharging_C22

entry:
PACE_CHARGE_CTRL = false;
Z_ATR_CTRL = false;
Z_VENT_CTRL = false;
ATR_PACE_CTRL = false;
VENT_PACE_CTRL=true;
ATR_GND_CTRL=false;
FRONTEND_CTRL=false;
PACE_GND_CTRL=true;
LEDR = true;
LEDB = false;
```

Figure 1.3.2.1 - VVI Stateflow

1.3.3 - AAI Stateflow



Charging_C22 is the state that charges the C22 capacitor and turns on the sensing circuitry this allow the electricity to build and pace the heart

```
Charging_C22

entry:
PACE_CHARGE_CTRL= true;
VENT_CMP_REF_PWM = false;
Z_ATR_CTRL=false;
PACING_REF_PWM=Amp;
ATR_CMP_REF_PWM=70;
ATR_PACE_CTRL= false;
VENT_PACE_CTRL=false;
PACE_GND_CTRL=true;
ATR_GND_CTRL=true;
VENT_GND_CTRL=false;
FRONTEND_CTRL=true;
LEDR = true;
LEDB = false;
```

Discharging_C22, discharges the C22 capicitor by taking the electricity and sending it through the heart to pace it.

```
Discharging_C22

entry:
PACE_CHARGE_CTRL = false;
Z_ATR_CTRL = true;
Z_VENT_CTRL = false;
ATR_PACE_CTRL = true;
VENT_PACE_CTRL=false;
ATR_GND_CTRL=false;
FRONTEND_CTRL=true;
LEDR = false;
LEDB = true;
```

Figure 1.3.3.0 - Overall AAI Simulink Function

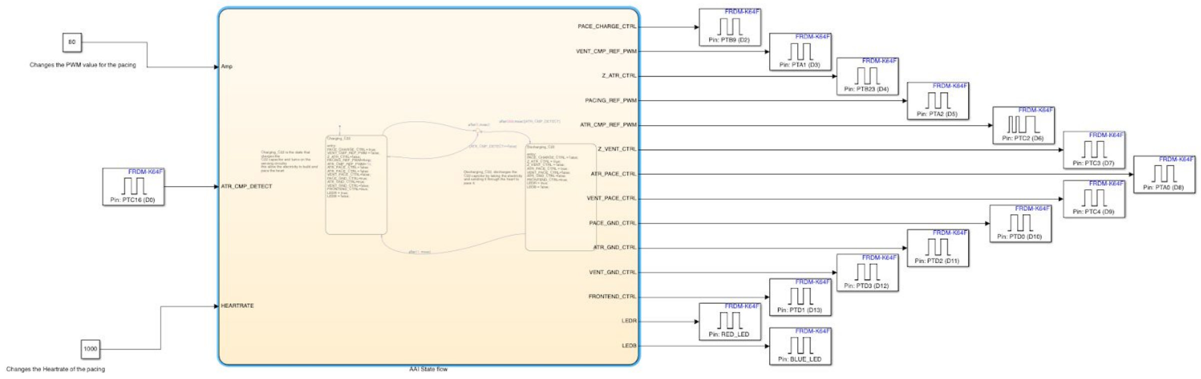


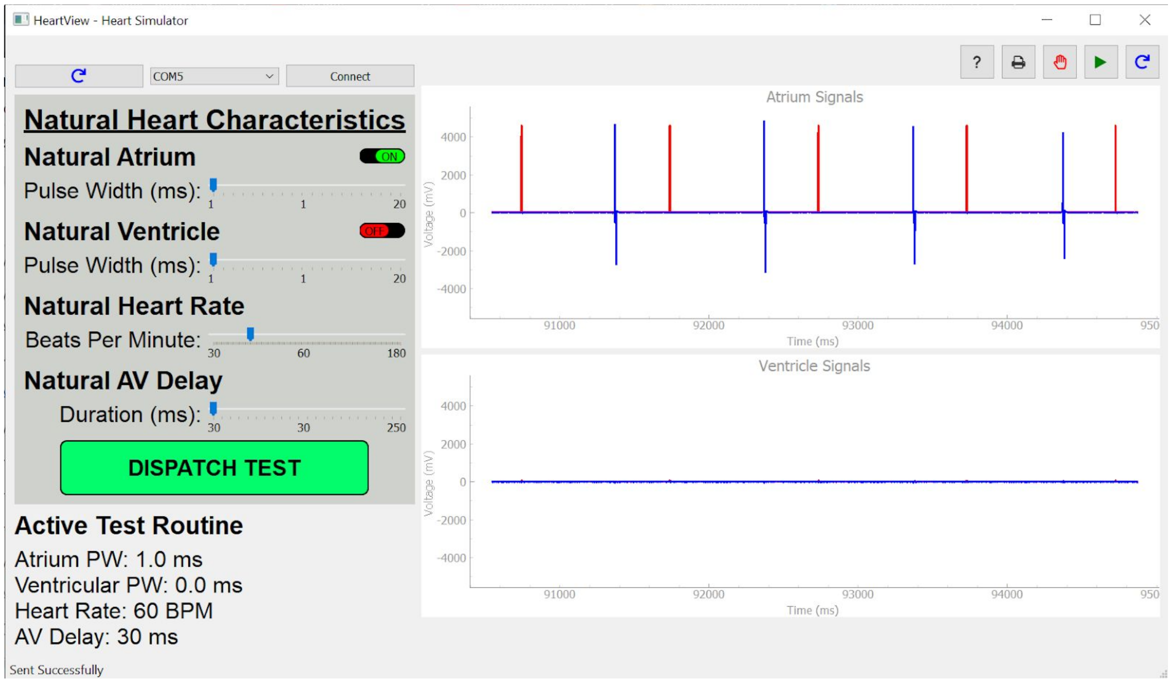
Figure 1.3.3.1 - AAI Stateflow

1.4 - Testing

For testing of the pacemaker, we used the heartview to monitor if the conditions are met throughout various pacing modes.

1.4.0 - AOO Test

1.4.0.0 - AOO Test 1



Natural Heart Characteristics

Natural Atrium ON

Pulse Width (ms): 1

Natural Ventricle OFF

Pulse Width (ms): 1

Natural Heart Rate

Beats Per Minute: 60

Natural AV Delay

Duration (ms): 30

Active Test Routine

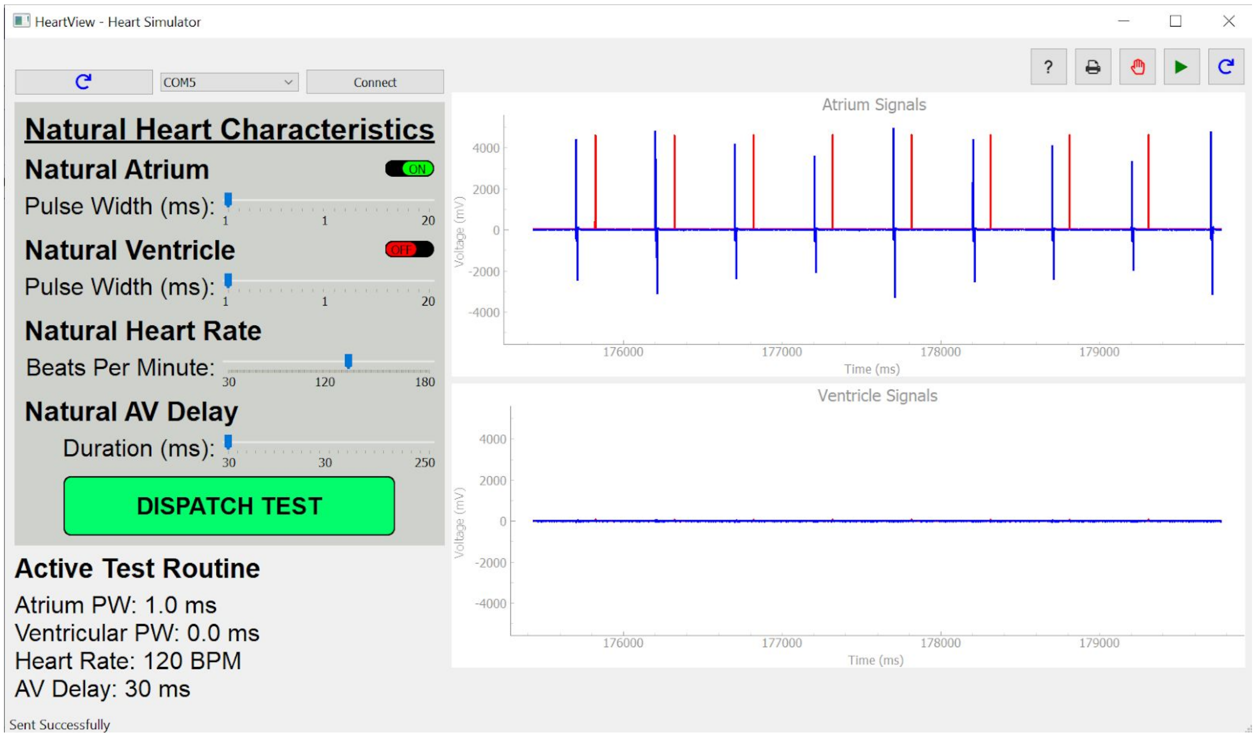
Atrium PW: 1.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 60 BPM
AV Delay: 30 ms

Sent Successfully

Figure 1.4.0.0.0 - AOO Test 1

AOO Test 1 Result: Pass - The atrium is paced once every second as it is set in Simulink. The natural atrium is turned on for reference and set to 60 beats per minute.

1.4.0.1 - AOO Test 2



Natural Heart Characteristics

Natural Atrium ON

Pulse Width (ms): 1

Natural Ventricle OFF

Pulse Width (ms): 1

Natural Heart Rate

Beats Per Minute: 120

Natural AV Delay

Duration (ms): 30

Active Test Routine

Atrium PW: 1.0 ms

Ventricular PW: 0.0 ms

Heart Rate: 120 BPM

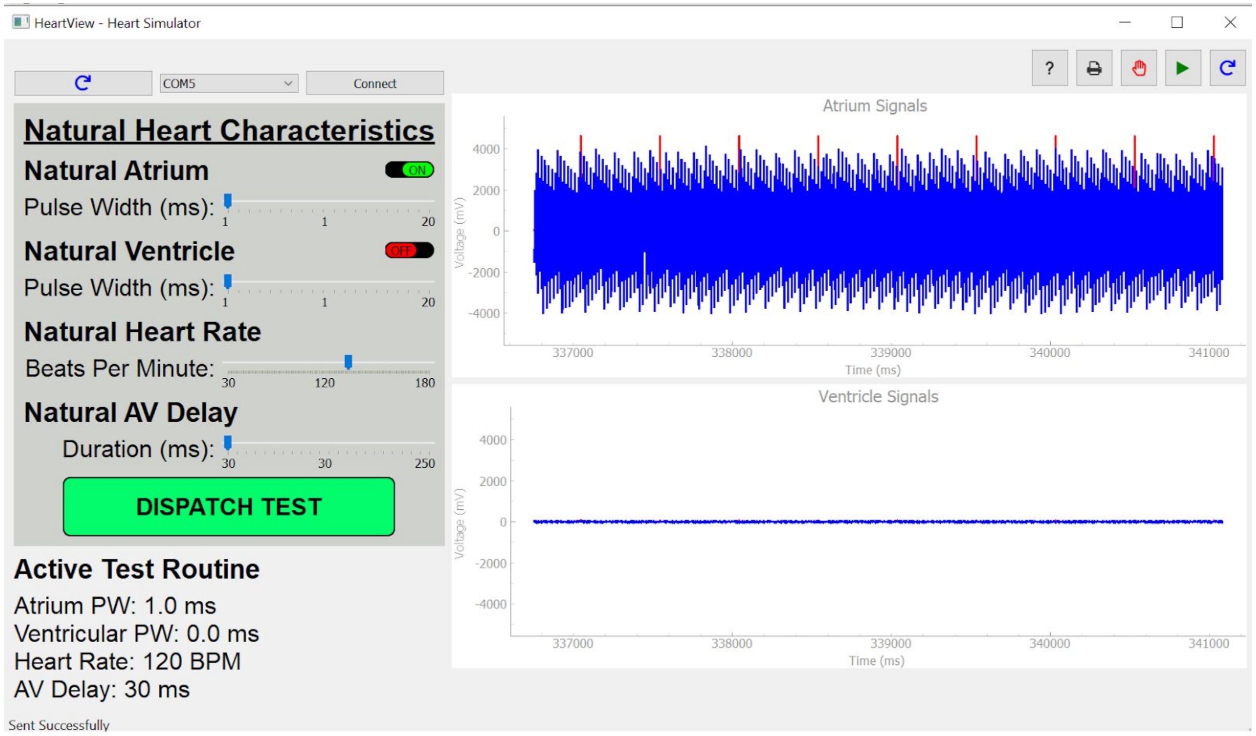
AV Delay: 30 ms

Sent Successfully

Figure 1.4.0.1.0 - AOO Test 2

AOO Test 2 Result: Pass - The atrium is paced twice every second as it is set in Simulink. The natural atrium is turned on for reference and set to 120 beats per minute.

1.4.0.2 - AOO Test 3



Active Test Routine
Atrium PW: 1.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 120 BPM
AV Delay: 30 ms

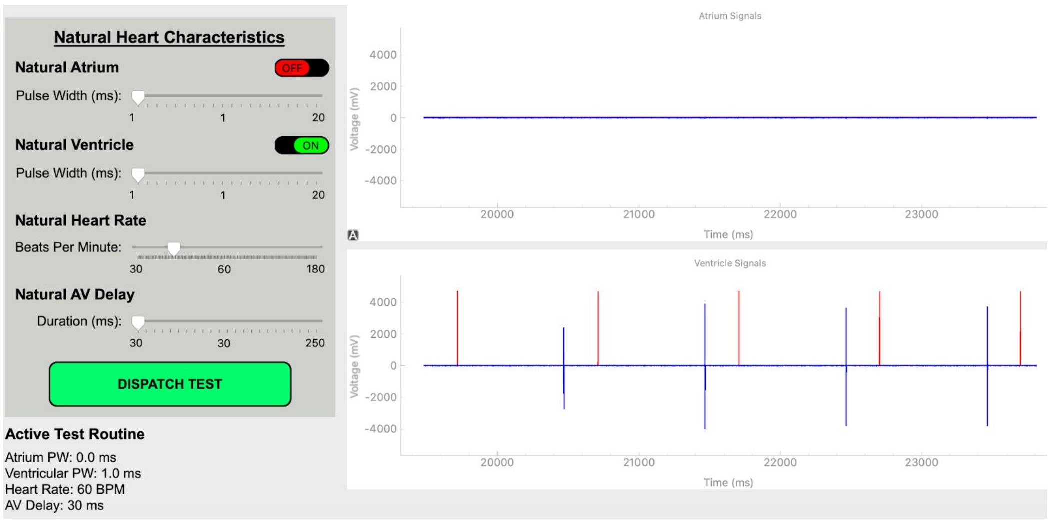
Sent Successfully

Figure 1.4.0.2.0 - AOO Test 3

AOO Test 3 Result: Fail - The atrium is set to be paced every zero seconds. This should be an error however it sends a new pulse instantly.

1.4.1 - VOO Test

1.4.1.0 - VOO Test 1

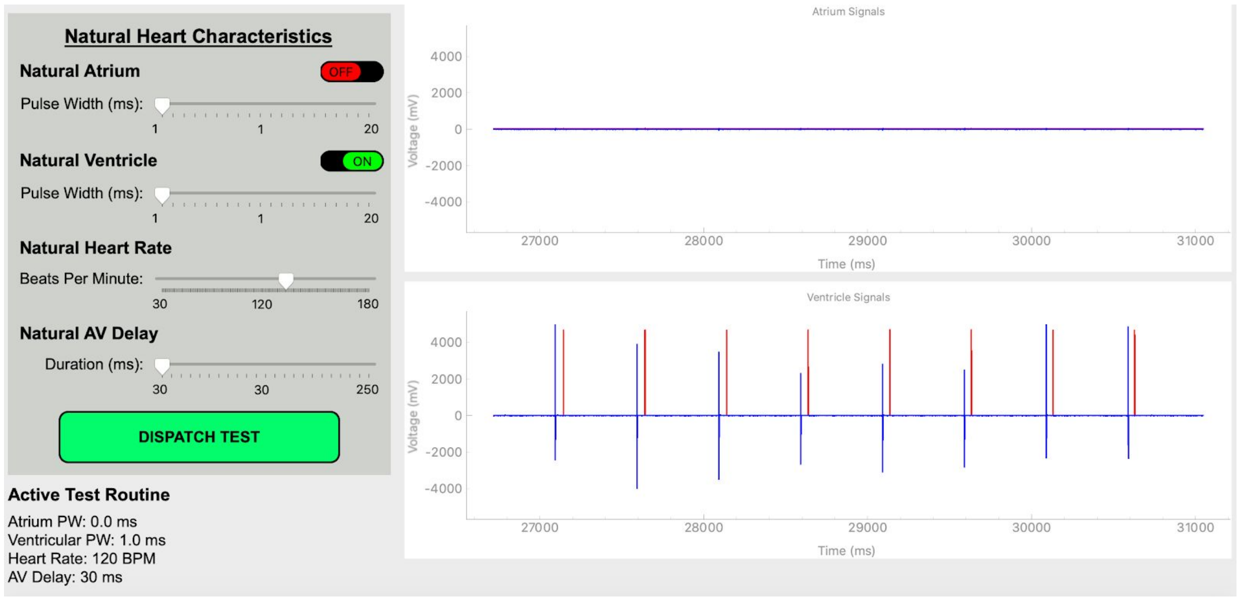


Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 60 BPM
AV Delay: 30 ms

Figure 1.4.1.0.0 - VOO Test 1

VOO Test 1 Result: Pass - The ventricle is paced once every second as it is set in Simulink. The natural ventricle is turned on for reference and set to 60 beats per minute.

1.4.1.1 - VOO Test 2

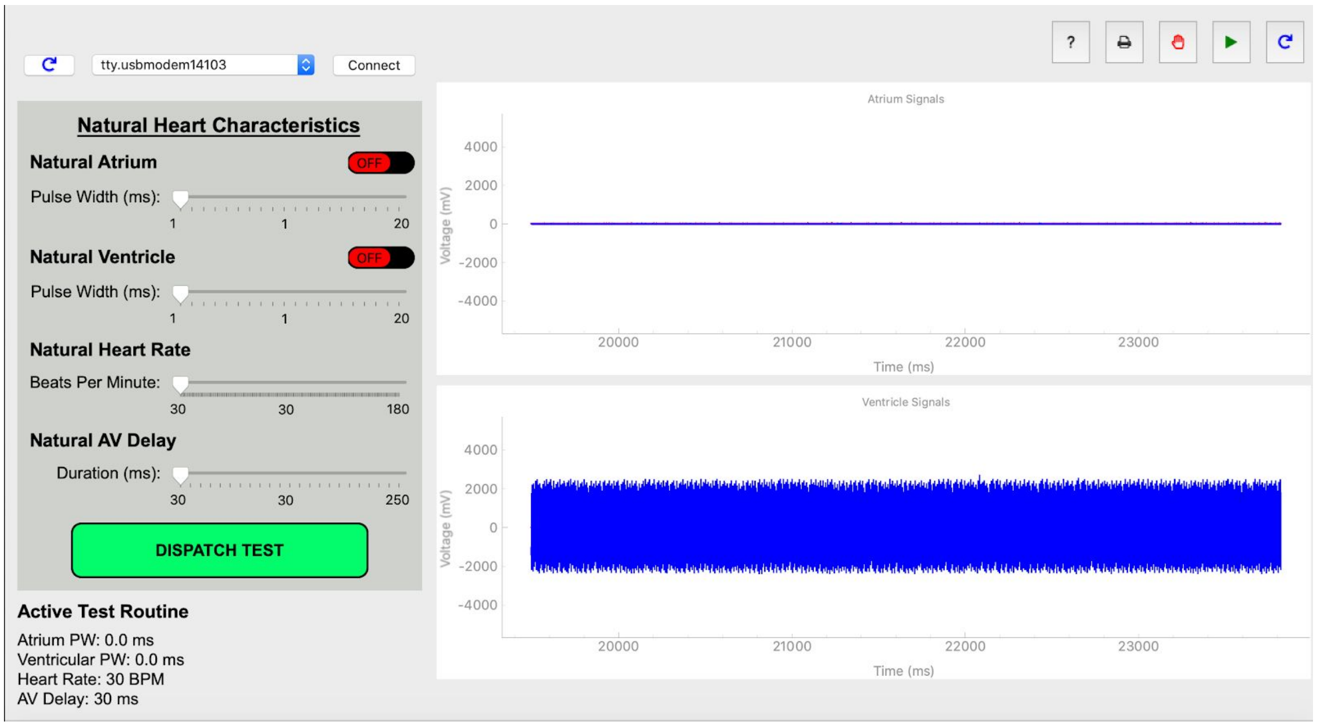


Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 120 BPM
AV Delay: 30 ms

Figure 1.4.1.1.0 - VOO Test 2

VOO Test 2 Result: Pass - The ventricle is paced twice every second as it is set in Simulink. The natural ventricle is turned on for reference and set to 120 beats per minute.

1.4.1.2 - VOO Test 3



Natural Heart Characteristics

Natural Atrium: OFF
Pulse Width (ms): 1 (1–20)

Natural Ventricle: OFF
Pulse Width (ms): 1 (1–20)

Natural Heart Rate
Beats Per Minute: 30 (30–180)

Natural AV Delay
Duration (ms): 30 (30–250)

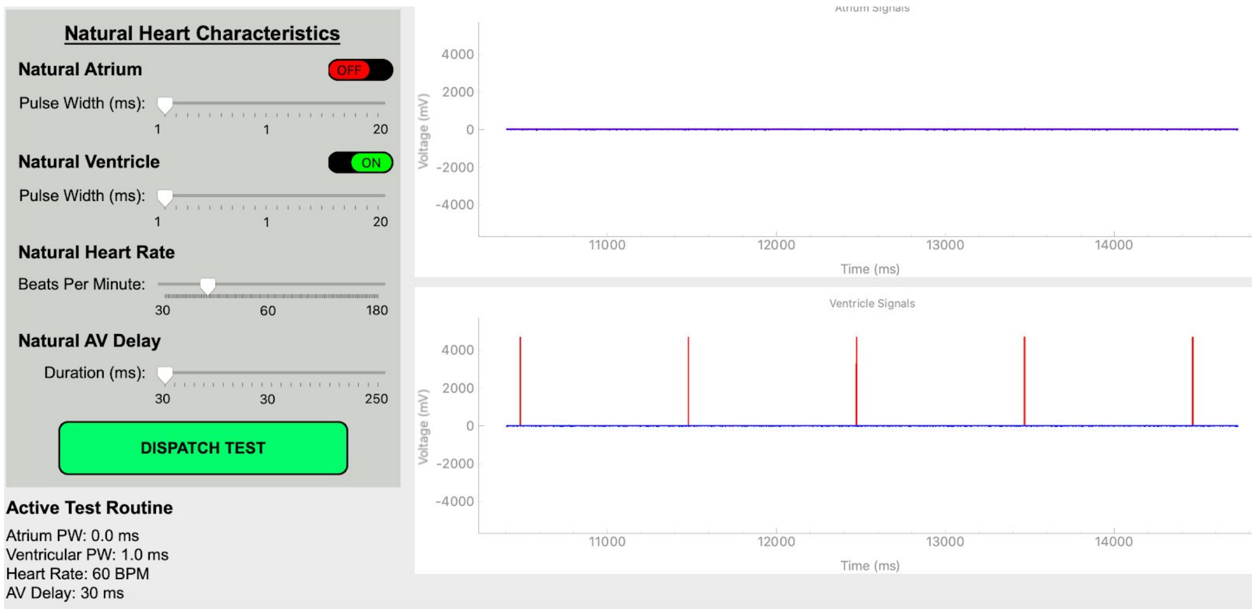
Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 30 BPM
AV Delay: 30 ms

Figure 1.4.1.2.0 - VOO Test 3

VOO Test 3 Result: Fail - The ventricle is set to be paced every zero seconds. This should be an error however it sends a new pulse instantly.

1.4.2 - VVI Test

1.4.2.0 - VVI Test 1

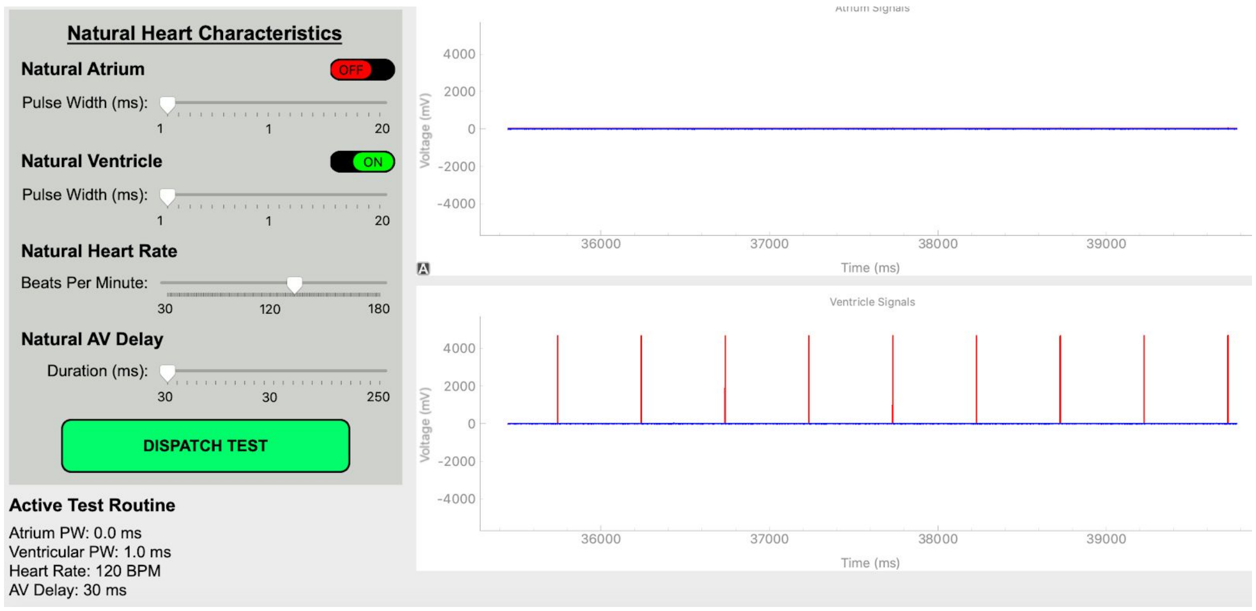


Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 60 BPM
AV Delay: 30 ms

Figure 1.4.2.0.0 - VVI Test 1

VVI Test 1 Result: Pass - The natural heart rate and the pacemaker are both set to 60 BPM and therefore the pacemaker does not pace.

1.4.2.1 - VVI Test 2

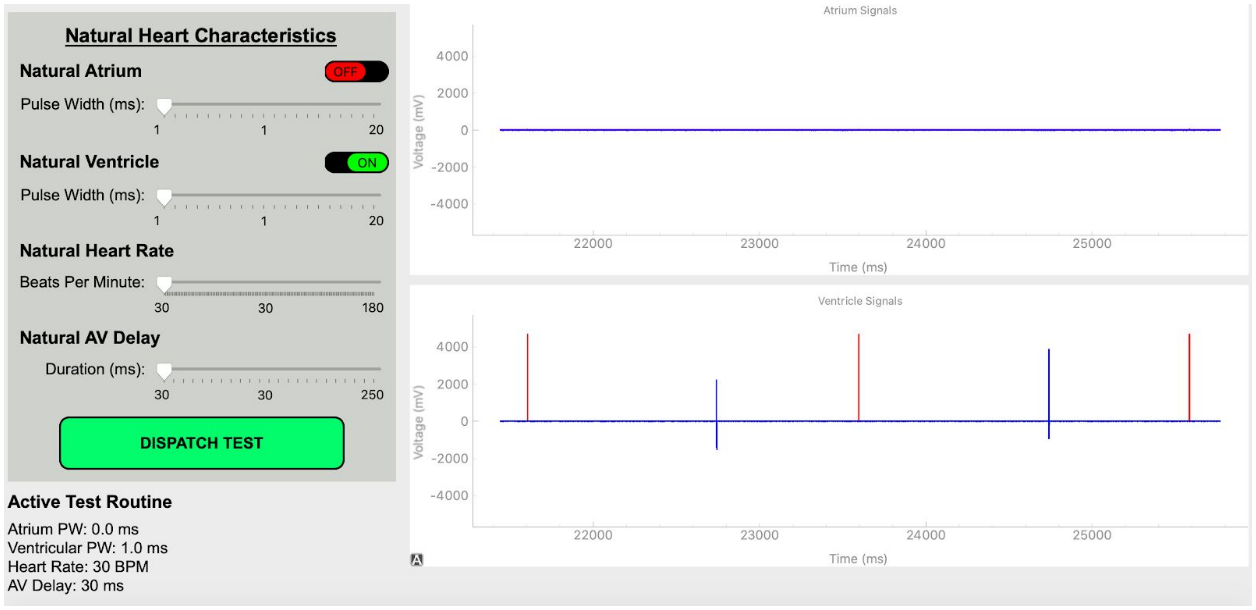


Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 120 BPM
AV Delay: 30 ms

Figure 1.4.2.1.0 - VVI Test 2

VVI Test 2 Result: Pass - The pacemaker is set to 60 BPM and the natural heart rate is set to 120 BPM. Since the heart is pacing faster than the pacemaker setting, the pacemaker does not need to generate a pulse.

1.4.2.2 - VVI Test 3

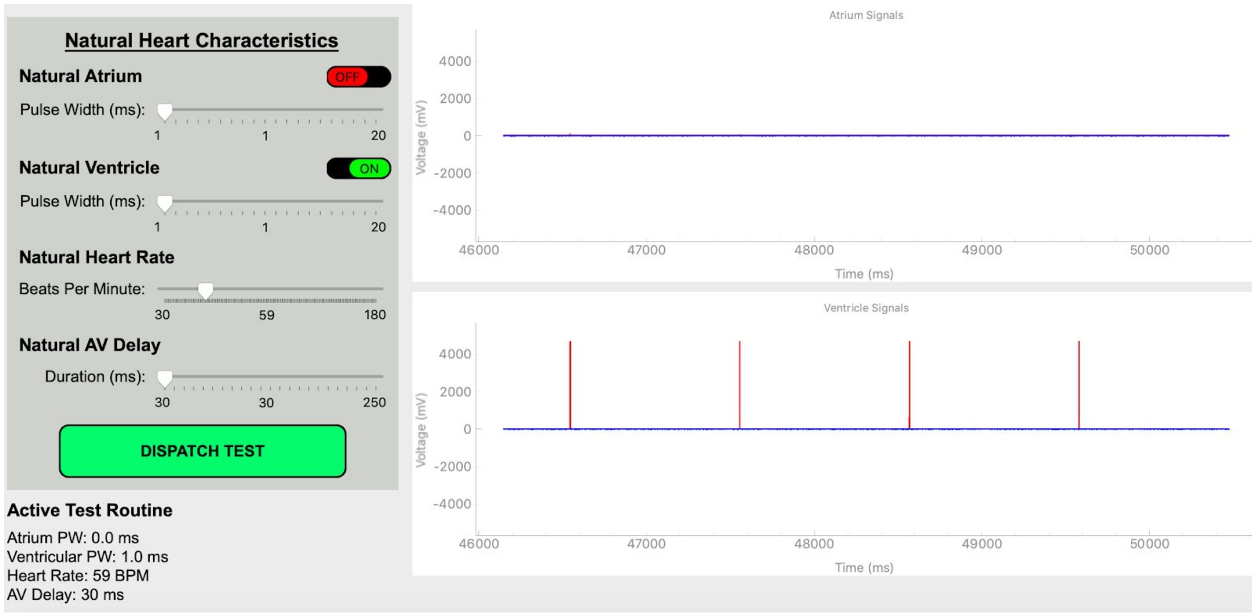


Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 30 BPM
AV Delay: 30 ms

Figure 1.4.2.2.0 - VVI Test 3

VVI Test 3 Result: Fail - The pacemaker is set to 60 BPM and the natural heart rate is set to 30BPM. The pacemaker should send a pulse exactly between the natural ventricular pulses. It does send one between, however it is slightly delayed and therefore this test is a fail.

1.4.2.3 - VVI Test 4



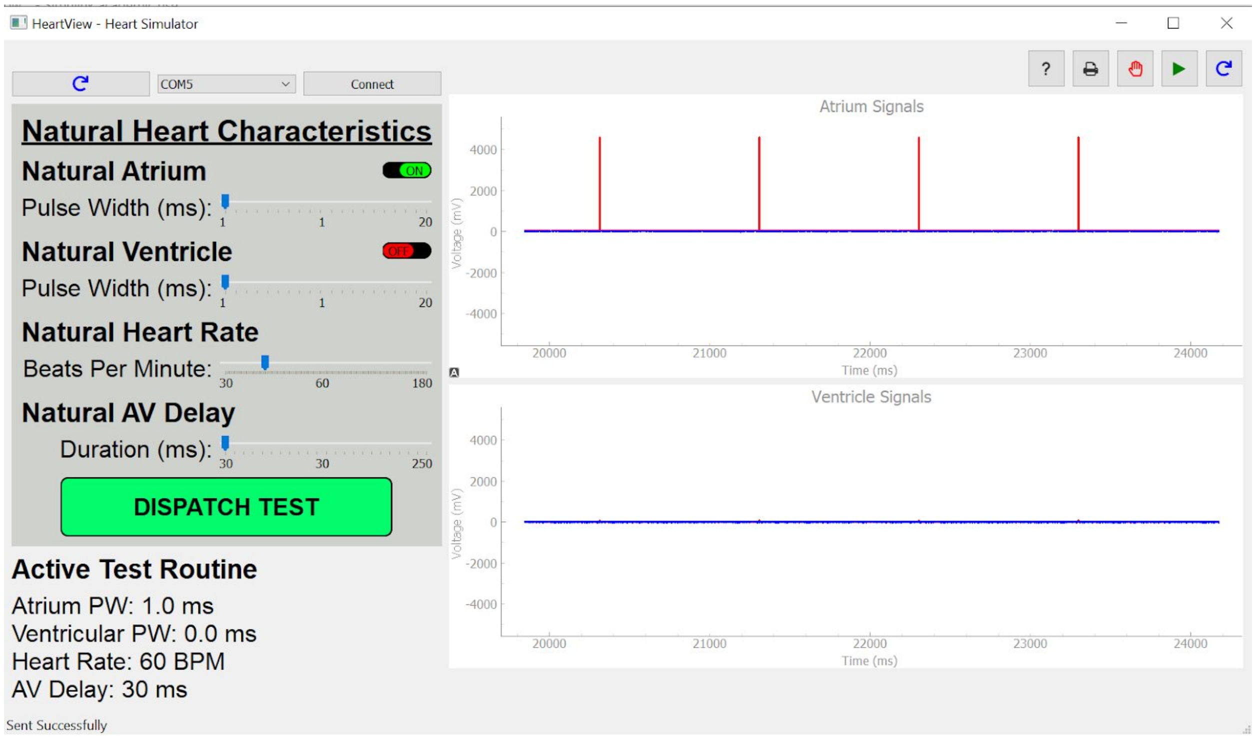
Active Test Routine
Atrium PW: 0.0 ms
Ventricular PW: 1.0 ms
Heart Rate: 59 BPM
AV Delay: 30 ms

Figure 1.4.2.3.0 - VVI Test 4

VVI Test 4 Result: Fail - The pacemaker is set to 60 BPM and the natural heart rate is set to 59 BPM. Since the natural heart rate is slower than the pacemaker setting, it should pulse just before the red lines, however it does not.

1.4.3 - AAI Test

1.4.3.0 - AAI Test 1



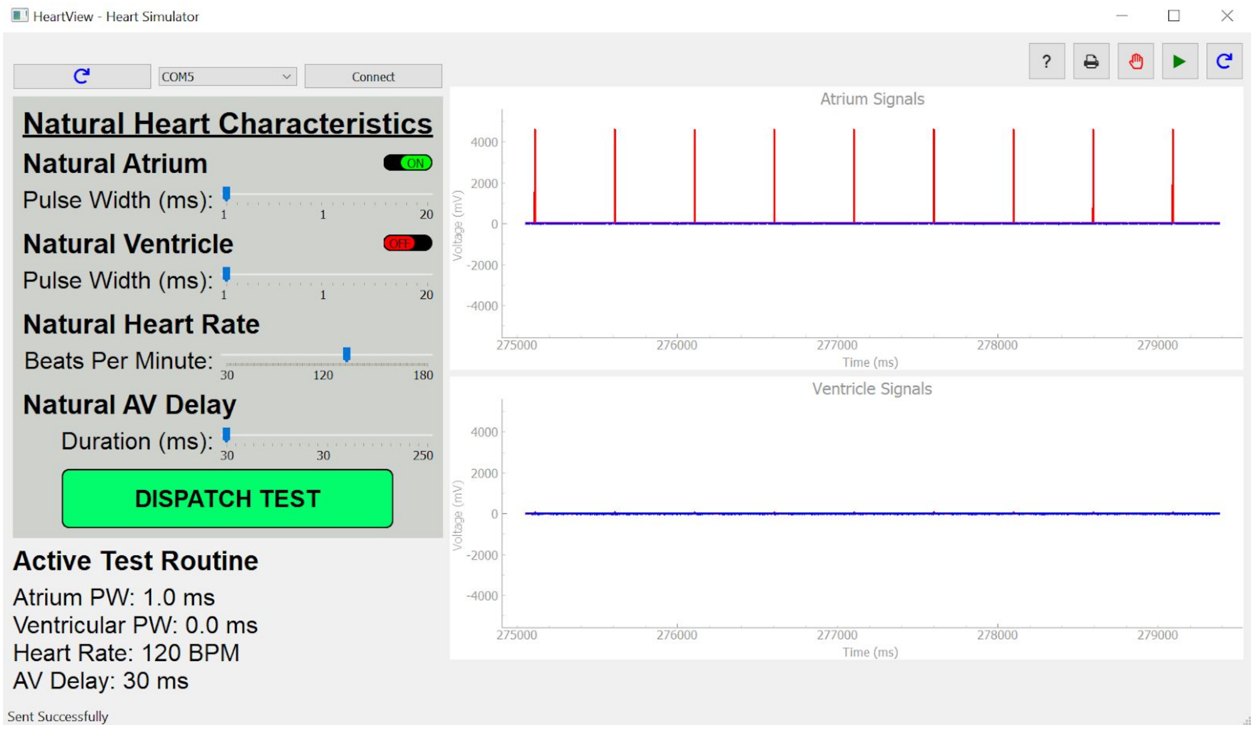
Active Test Routine
Atrium PW: 1.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 60 BPM
AV Delay: 30 ms

Sent Successfully

Figure 1.4.3.0.0 - AAI Test 1

AAI Test 1 Result: Pass - The natural heart rate and the pacemaker are set to 60BPM so the pacemaker does not pace.

1.4.3.1 - AAI Test 2



Natural Heart Characteristics

Natural Atrium
Pulse Width (ms): 1

Natural Ventricle
Pulse Width (ms): 1

Natural Heart Rate
Beats Per Minute: 120

Natural AV Delay
Duration (ms): 30

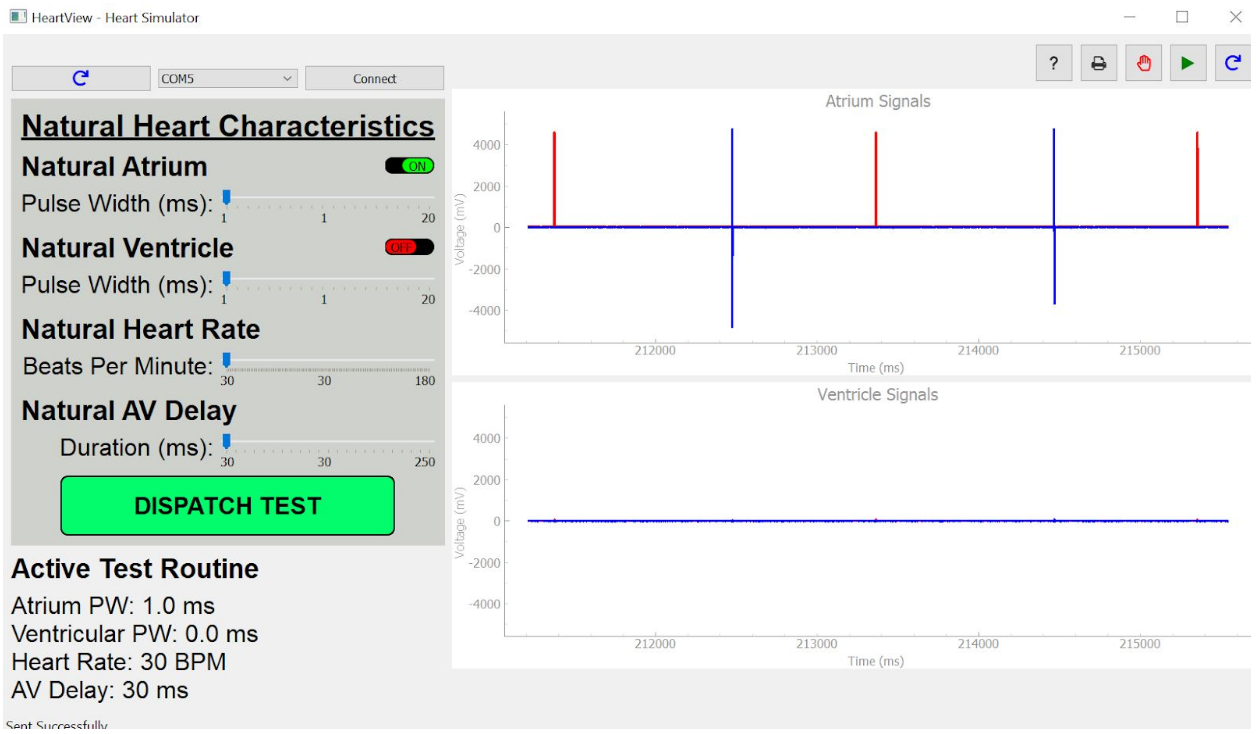
Active Test Routine
Atrium PW: 1.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 120 BPM
AV Delay: 30 ms

Sent Successfully

Figure 1.4.3.1.0 - AAI Test 2

AAI Test 1 Result: Pass - The natural heart rate is set to 120BPM and the pacemaker is set to 60BPM meaning that the pacemaker will not pace at all.

1.4.3.2 - AAI Test 3



Natural Heart Characteristics

Natural Atrium ON

Pulse Width (ms): 1

Natural Ventricle OFF

Pulse Width (ms): 1

Natural Heart Rate

Beats Per Minute: 30

Natural AV Delay

Duration (ms): 30

Active Test Routine

Atrium PW: 1.0 ms
Ventricular PW: 0.0 ms
Heart Rate: 30 BPM
AV Delay: 30 ms

Sent Successfully

Figure 1.4.3.2.0 - AAI Test 3

AAI Test 3 Result: Fail - The natural heart rate is set to 30BPM and the pacemaker is set to 60BPM, therefore the pacemaker's pacing should be between the interval of the natural heart rate but it is slightly off centred.

1.4.3.3 - AAI Test 4

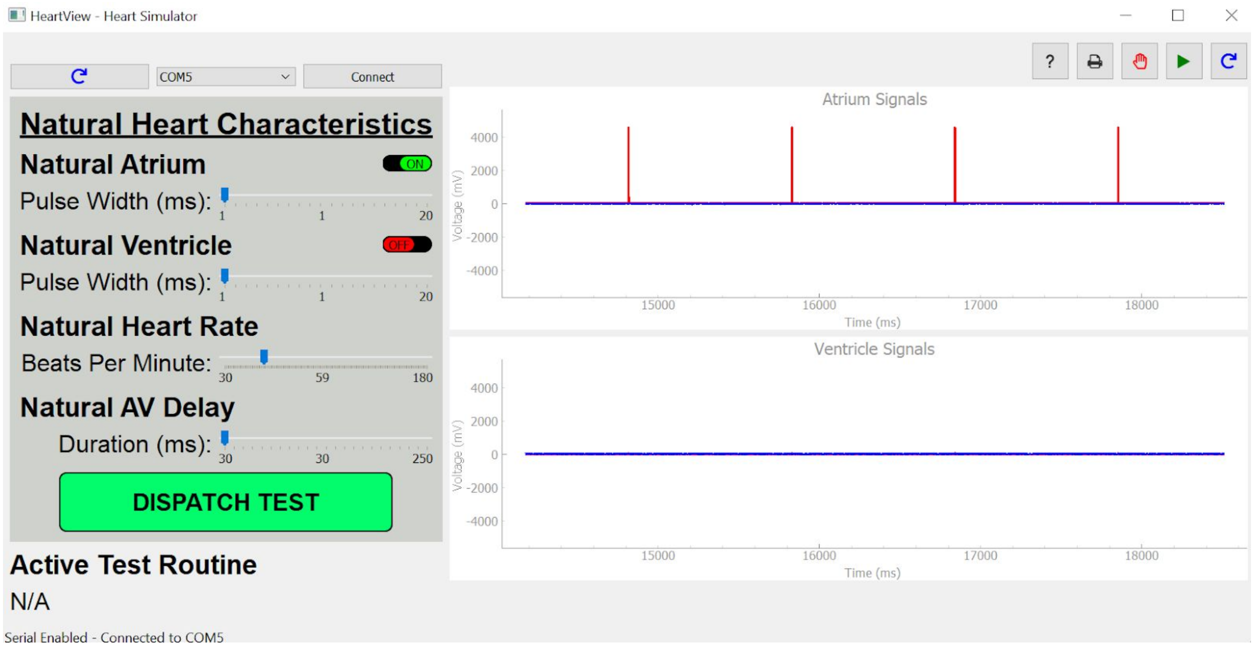


Figure 1.4.3.3.0 - AAI Test 4

AAI Test 4 Result: Pass - The natural heart rate is set to 59BPM while the heart is set to 60BPM meaning the pacemaker should be pacing but due to a time delay in the stateflow it does not.

2.0 - DCM Design

The objective was to design an interface to implement login/registration, the pacing modes mentioned in Part 1, storing the parameter data of all the pacing modes, and essential aspects of the interface with respect to 3.2.2 in [3].[2] The DCM was then expanded to include all new pacing modes (DOO, AOOR, VOOR, AAIR, VVIR, DOOR). This includes buttons, inputs, displays, and other associated features that were created during *Assignment 2*.

Further developments were also made that affected all pacing modes. These include the addition of serial communication between the DCM and the pacemaker, the addition of stored and alterable parameters within the DCM, and the ability to display egram data.

[2] Note: "" is used for the reference of either code of the Pacemaker App or a representation of files associated with the App. For example, "Var = 12".

2.1 - Requirement Changes

2.1.0 - Assignment 1

Additional functionality will be added to the DCM interface to incorporate integration with the pacemaker board. Four individual buttons (white, currently not functioning) will be appropriately marked to send the AAI, AAO, VVI, and VOO pacing modes from the DCM to the pacemaker.

2.1.1 - Assignment 2

Currently, functionality is prioritized over appearance in the DCM. In future iterations, the aesthetic appearance of the DCM will be prioritized. This includes the color scheme, the font, the shape and placement of the buttons, etc.

2.2 - Design Decision Changes

2.2.0 - Assignment 1

The DCM was created using Python 3.6.0 64-bit and Kivy version 1.11.1. These versions were chosen for their widespread proliferation providing the designers' with many resources and references. This will aid the design of future iterations.

2.2.0.0 - User Independent Data

When logging into the DCM, the data for the programmable parameters are not user specific. Instead, the data is exactly the same for every user. This defeats the purpose of having a login menu with different users since the data for the pacing states are shared for every user. In future iterations, each user will have unique data that is saved and associated with their account alone.

2.2.1 - Assignment 2

The DCM was once again created using Python 3.6.0 64-bit and Kivy version 1.11.1. It was decided to stay with these choices since they were successful during *2.0 - DCM Design (Assignment 1 - Part 2)*.

2.2.1

2.4 - DCM Modules

Note that lettering of the topics is based on Assignment 1 [2] *and Assignment 2* [1].

2.4.1 - DCM Tutorial

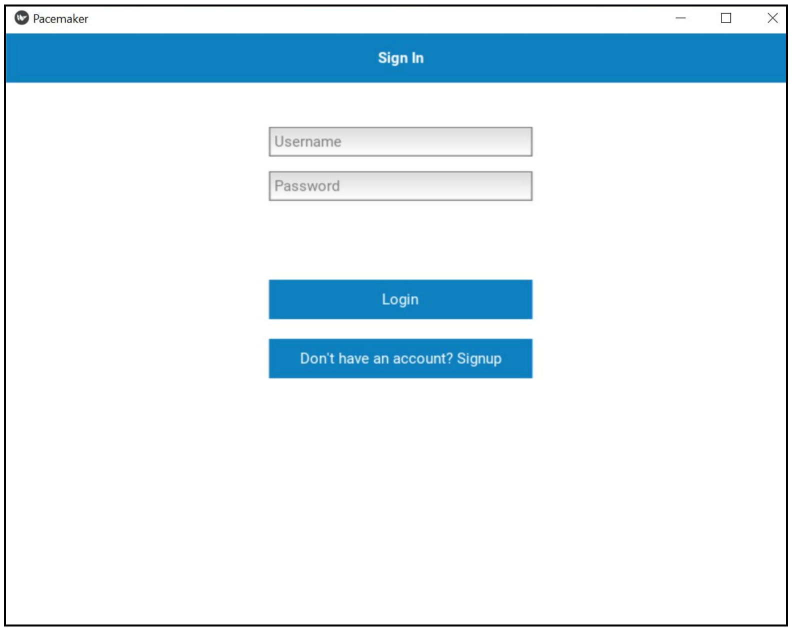


Figure 2.4.1.0 - Sign in Menu

Upon opening the program, the user is greeted by a sign-in page, with prompts to enter a username and password. The “Login” button can then be clicked to progress into the program. If an account has not yet been created, then the “Don’t have an account? Signup” button can be clicked to bring the user to the registration page.

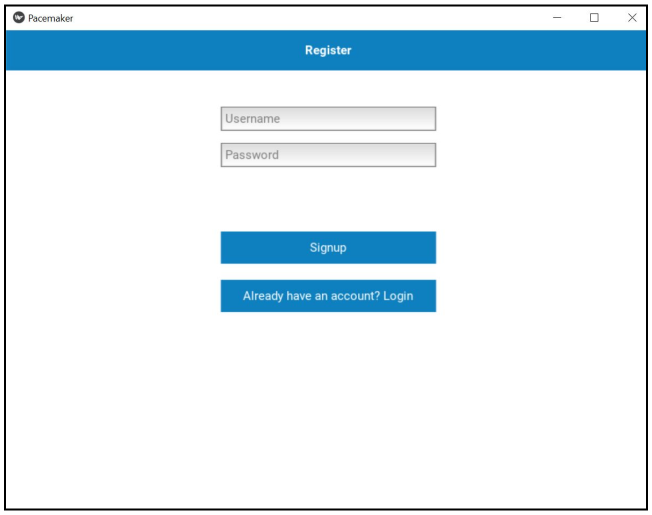


Figure 2.4.1.1 - Registration Menu

The registration page prompts the user to enter a unique username and password, then to click the “Signup” button. If an account was created previously, then the user is prompted to click the “Already have an account? Login” button.

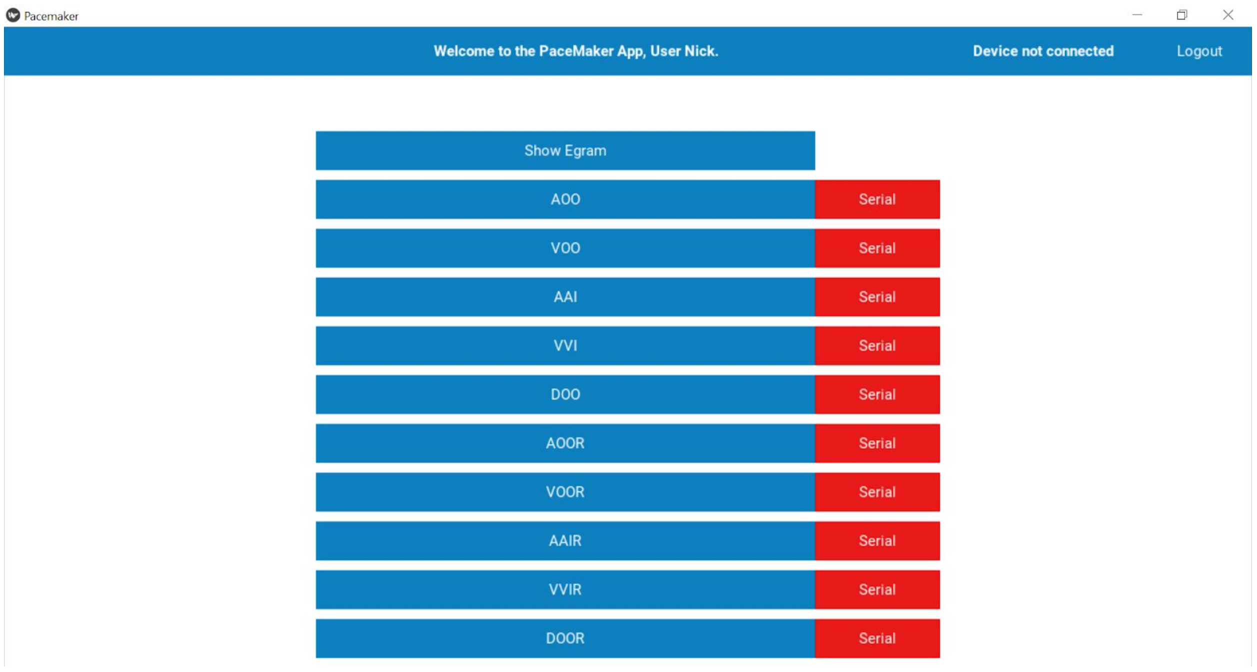


Figure 2.4.1.2 - Main Menu without pacemaker device connected

The main menu provides the option to choose one of the 10 potential pacing modes (AOO, VOO, AAI, VVI, DOO, AOOR, VOOR, AAIR, VVIR, DOOR). The top of the menu displays the username of the current user of the program. As can be seen at the top of the menu, “Device not connected” informs the user that there is currently no pacemaker connected to the program. This makes the red “serial” buttons dysfunctional until there is a pacemaker connected, as without a pacemaker, those buttons will not have any location to send the pacing modes. When a pacing mode is selected, the user is brought to the pacing mode page to specify requirements. The “Show Egram” button near the top takes the user to the Electrocardiogram page.

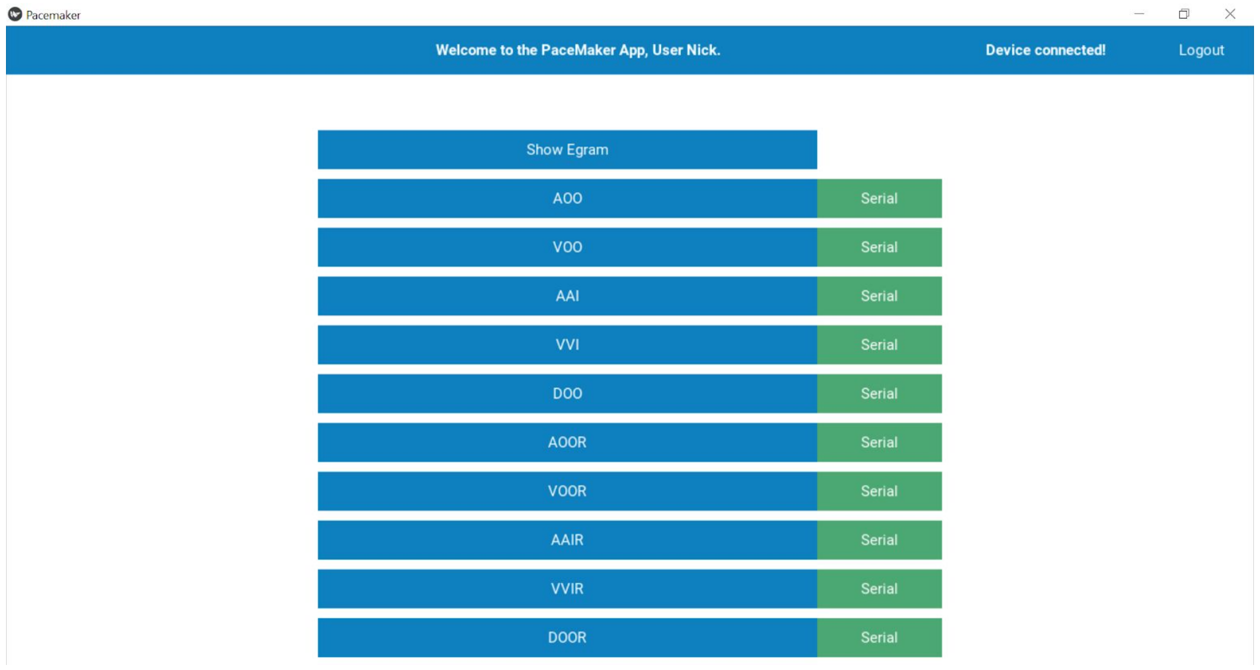
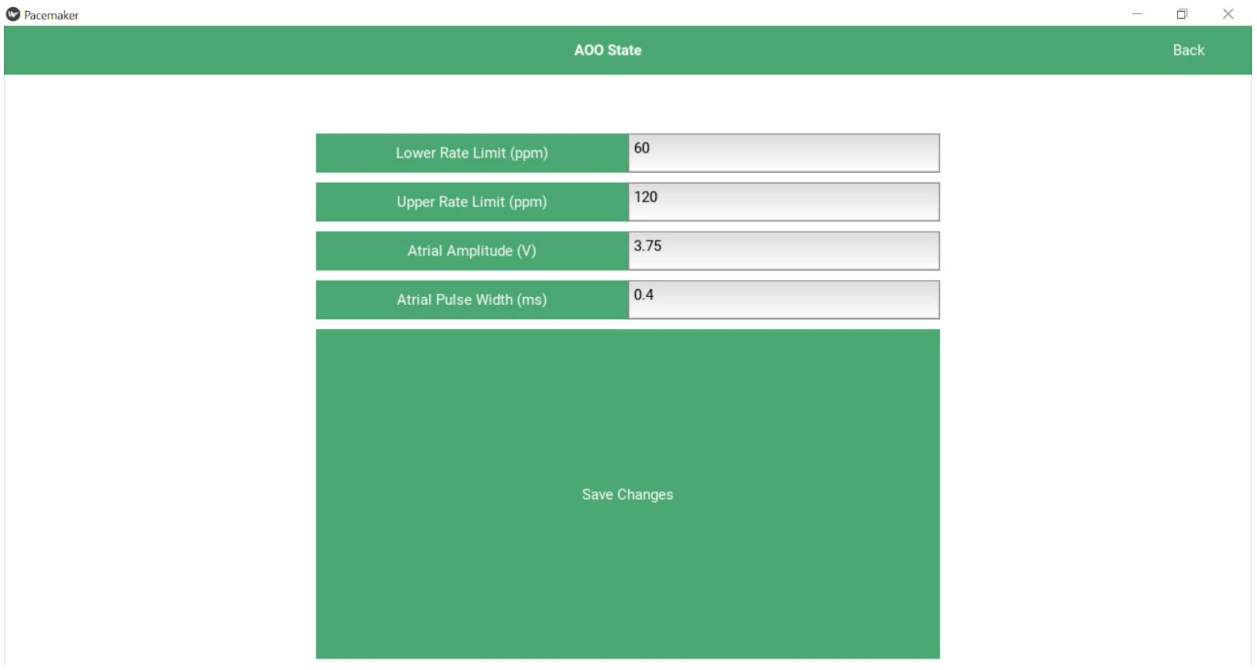


Figure 2.4.1.3 - Main Menu with pacemaker device connected

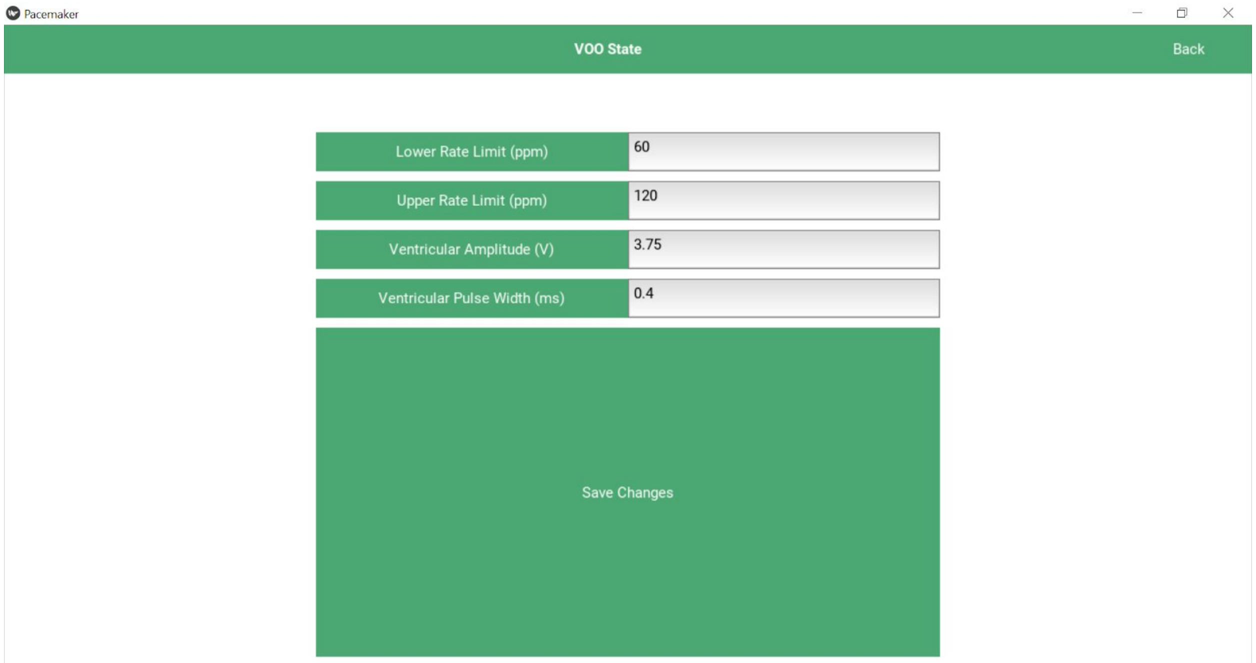
The main menu provides the option to choose one of the 10 potential pacing modes (AOO, VOO, AAI, VVI, DOO, AOOR, VOOR, AAIR, VVIR, DOOR). The top of the menu displays the username of the current user of the program. As can be seen at the top of the menu, “Device connected” informs the user that there is currently a pacemaker connected to the program. This gives the green “serial” buttons functionality, as they will send their respective pacing modes to the pacemaker when they are pressed. When a pacing mode is selected, the user is brought to the pacing mode page to specify requirements. The “Show Egram” button near the top takes the user to the Electrocardiogram page.



Lower Rate Limit (ppm): 60
Upper Rate Limit (ppm): 120
Atrial Amplitude (V): 3.75
Atrial Pulse Width (ms): 0.4

Figure 2.4.1.4 - AOO State Menu

The AOO state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.


Pacemaker

VOO State

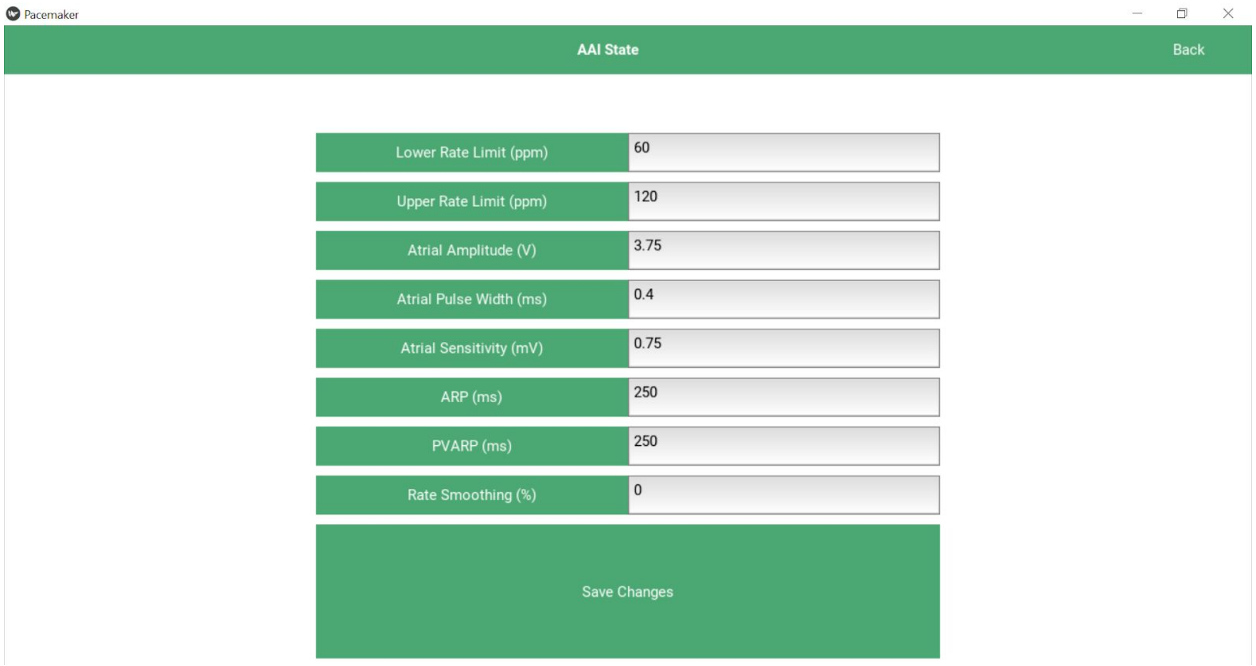
Back

Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Ventricular Amplitude (V)	3.75
Ventricular Pulse Width (ms)	0.4

Save Changes

Figure 2.4.1.5 - VOO State Menu

The VOO state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.


Pacemaker

AAI State

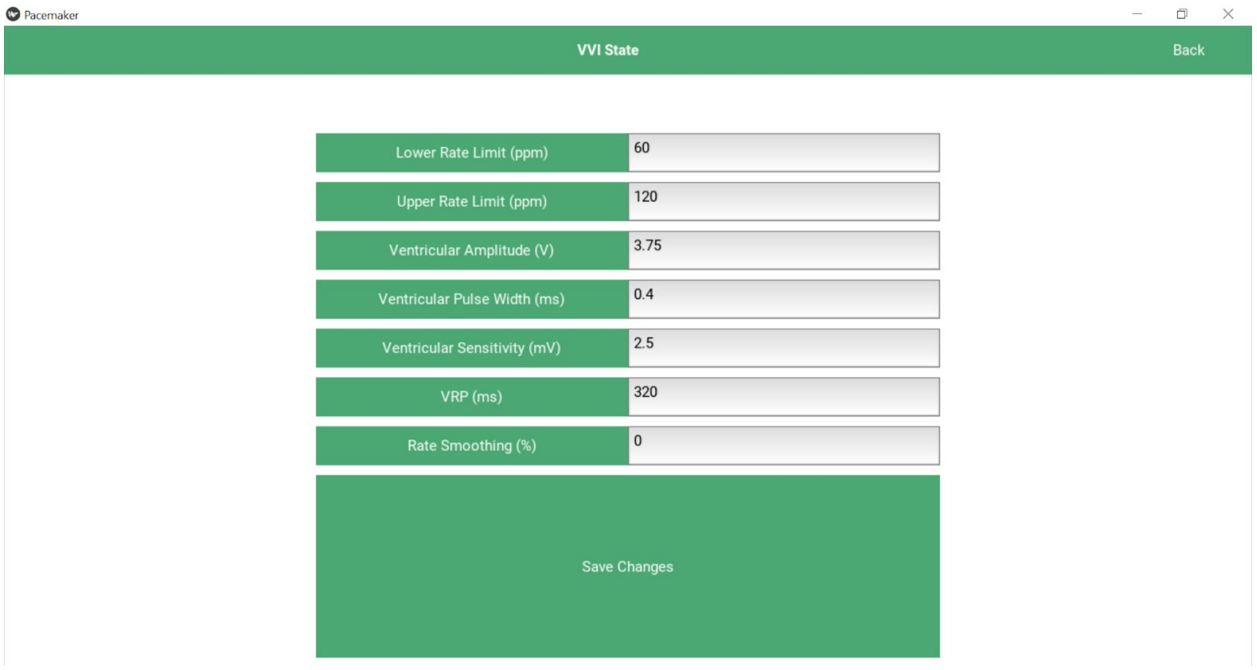
Back

Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Atrial Amplitude (V)	3.75
Atrial Pulse Width (ms)	0.4
Atrial Sensitivity (mV)	0.75
ARP (ms)	250
PVARP (ms)	250
Rate Smoothing (%)	0

Save Changes

Figure 2.4.1.6 - AAI State Menu

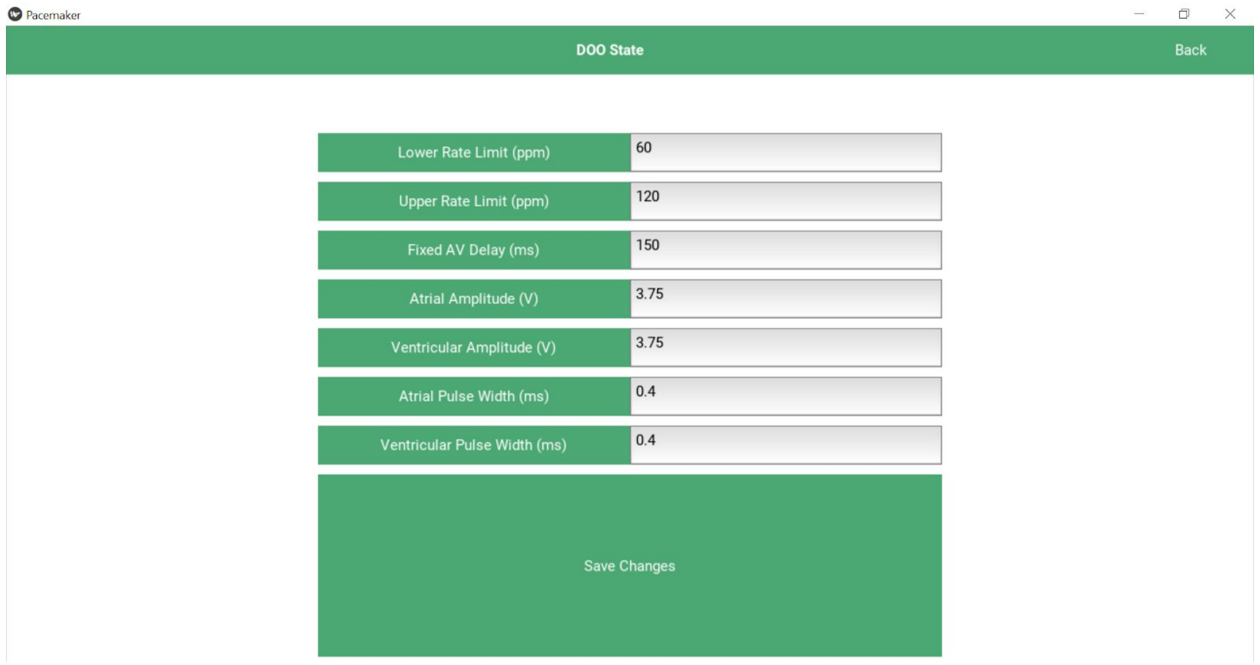
The AII state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.



Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Ventricular Amplitude (V)	3.75
Ventricular Pulse Width (ms)	0.4
Ventricular Sensitivity (mV)	2.5
VRP (ms)	320
Rate Smoothing (%)	0

Figure 2.4.1.7 - VVI State Menu

The VVI state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.



Pacemaker

DOO State

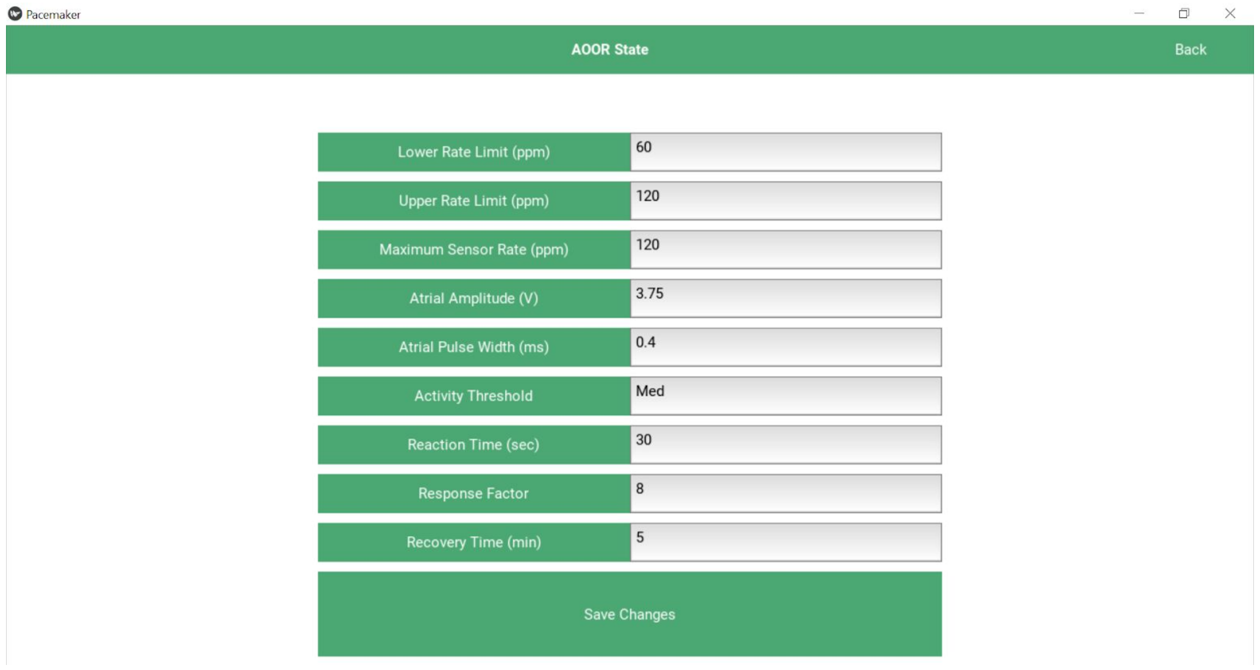
Back

Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Fixed AV Delay (ms)	150
Atrial Amplitude (V)	3.75
Ventricular Amplitude (V)	3.75
Atrial Pulse Width (ms)	0.4
Ventricular Pulse Width (ms)	0.4

Save Changes

Figure 2.4.1.8 - DOO State Menu

The DOO state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.



Pacemaker

AOOR State

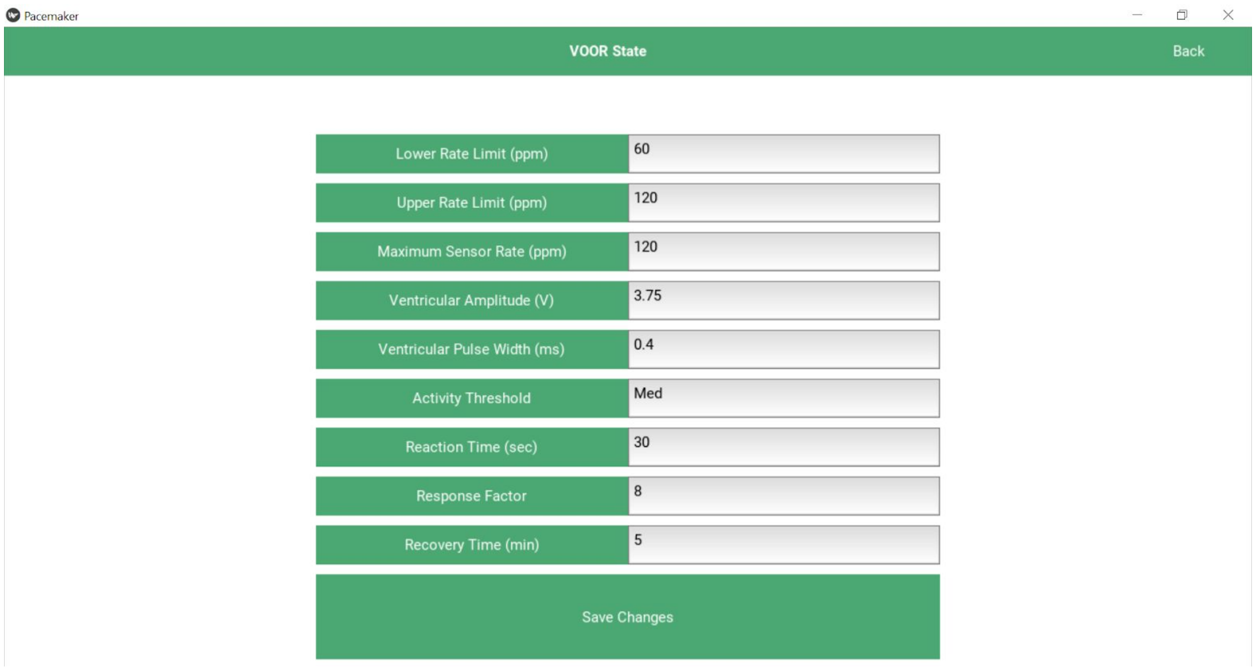
Back

Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Maximum Sensor Rate (ppm)	120
Atrial Amplitude (V)	3.75
Atrial Pulse Width (ms)	0.4
Activity Threshold	Med
Reaction Time (sec)	30
Response Factor	8
Recovery Time (min)	5

Save Changes

Figure 2.4.1.9 - AOOR State Menu

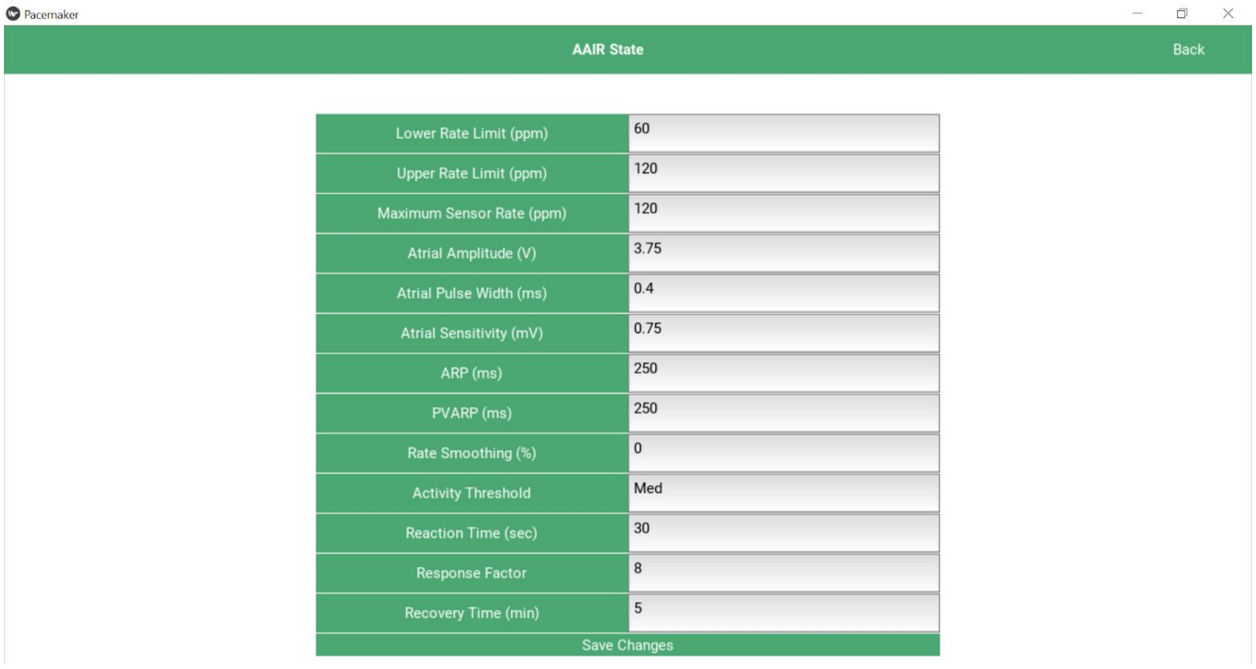
The AOOR state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.



Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Maximum Sensor Rate (ppm)	120
Ventricular Amplitude (V)	3.75
Ventricular Pulse Width (ms)	0.4
Activity Threshold	Med
Reaction Time (sec)	30
Response Factor	8
Recovery Time (min)	5

Figure 2.4.1.10 - VOOR State Menu

The VOOR state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.


Pacemaker

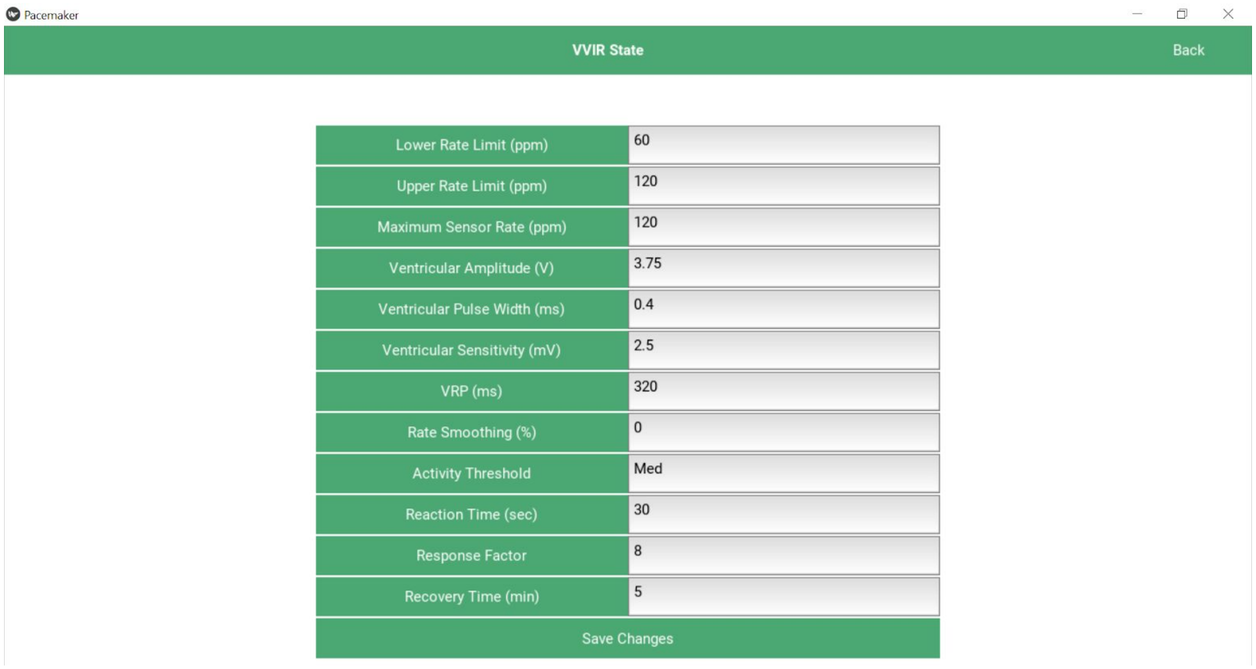
AAIR State | Back

Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Maximum Sensor Rate (ppm)	120
Atrial Amplitude (V)	3.75
Atrial Pulse Width (ms)	0.4
Atrial Sensitivity (mV)	0.75
ARP (ms)	250
PVARP (ms)	250
Rate Smoothing (%)	0
Activity Threshold	Med
Reaction Time (sec)	30
Response Factor	8
Recovery Time (min)	5

Save Changes

Figure 2.4.1.11 - AAIR State Menu

The AAIR state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.


Pacemaker

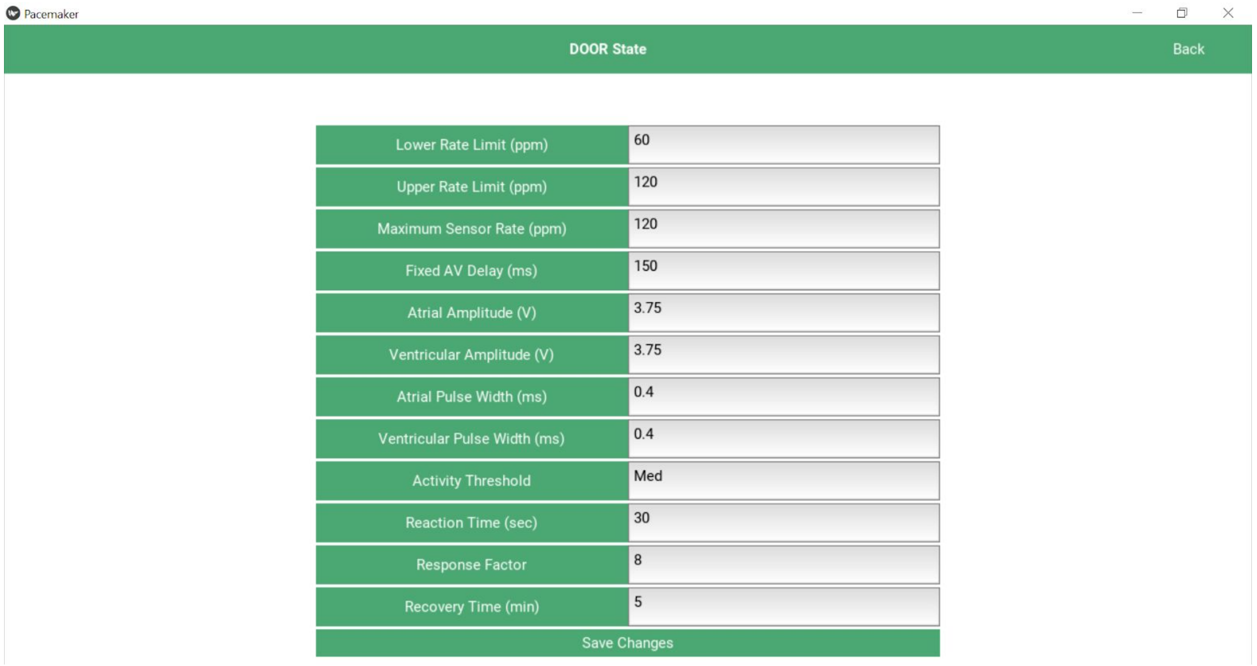
VVIR State | Back

Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Maximum Sensor Rate (ppm)	120
Ventricular Amplitude (V)	3.75
Ventricular Pulse Width (ms)	0.4
Ventricular Sensitivity (mV)	2.5
VRP (ms)	320
Rate Smoothing (%)	0
Activity Threshold	Med
Reaction Time (sec)	30
Response Factor	8
Recovery Time (min)	5

Save Changes

Figure 2.4.1.12 - VVIR State Menu

The VVIR state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.



Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Maximum Sensor Rate (ppm)	120
Fixed AV Delay (ms)	150
Atrial Amplitude (V)	3.75
Ventricular Amplitude (V)	3.75
Atrial Pulse Width (ms)	0.4
Ventricular Pulse Width (ms)	0.4
Activity Threshold	Med
Reaction Time (sec)	30
Response Factor	8
Recovery Time (min)	5

Figure 2.4.1.13 - DOOR State Menu

The DOOR state menu allows the user to place thresholds on the pacing mode, and save the changes. The user can then choose the “Back” option to return to the main menu.

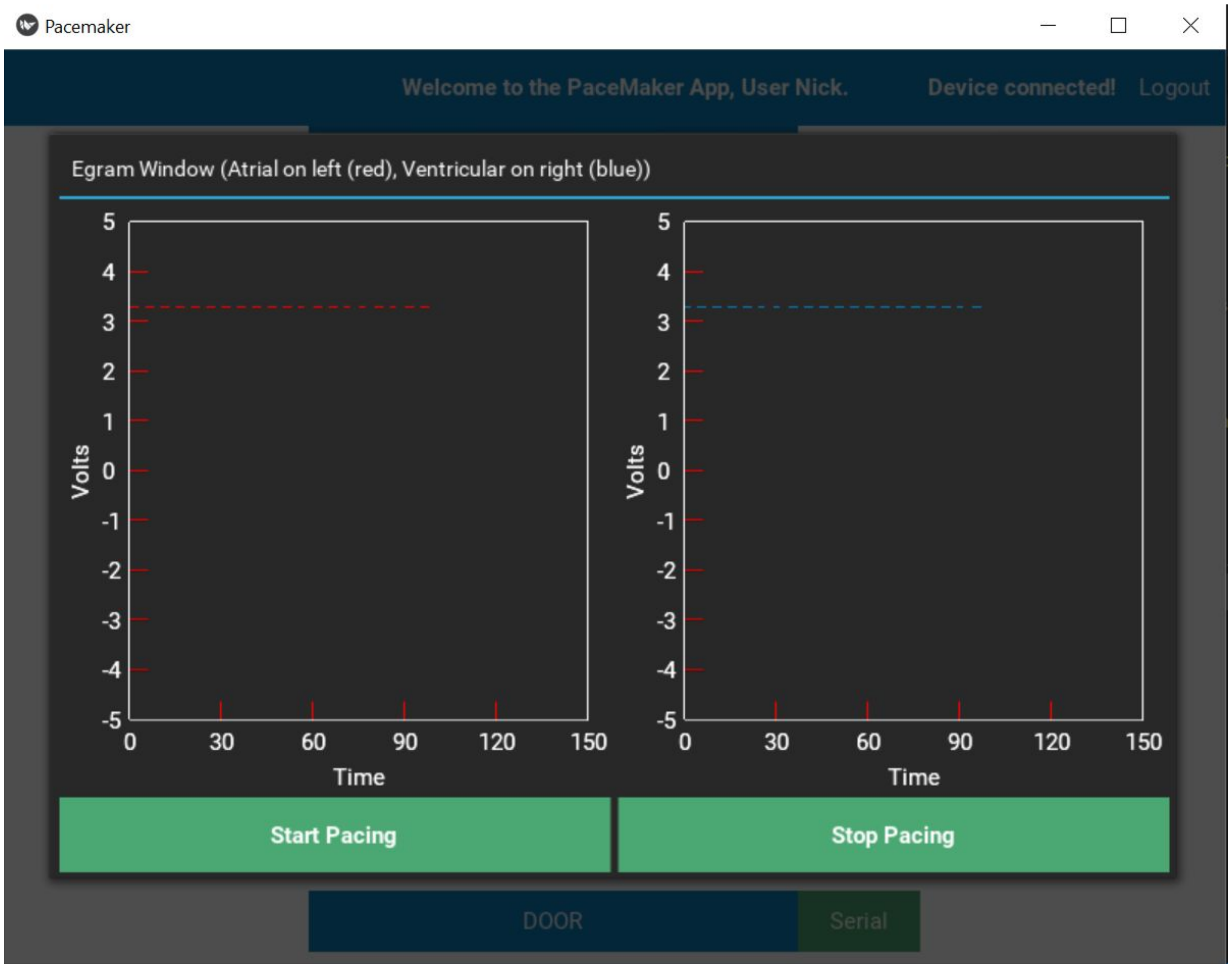

Pacemaker
Welcome to the PaceMaker App, User Nick.
Device connected!
Logout
Egram Window (Atrial on left (red), Ventricular on right (blue))
Volts
Time
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
30
60
90
120
150
Start Pacing
Stop Pacing
DOOR
Serial

Figure 2.4.1.13 - Egram Popup Window

The Egram Window displays an electrocardiogram of the heart. The left chart represents the atria of the heart, and the right window represents the heart's ventricles. Each chart displays the depolarization and repolarization of the respective compartments of the heart. When the "start pacing" button is pressed, the electrocardiogram begins to measure the electrical activity of the pacemaker, and displays the measured activity with the red and blue dotted lines. By pressing the "stop pacing" button, the electrocardiogram stops taking measurements of the electrical activity of the pacemaker.

2.4.2 - Login Menu Module

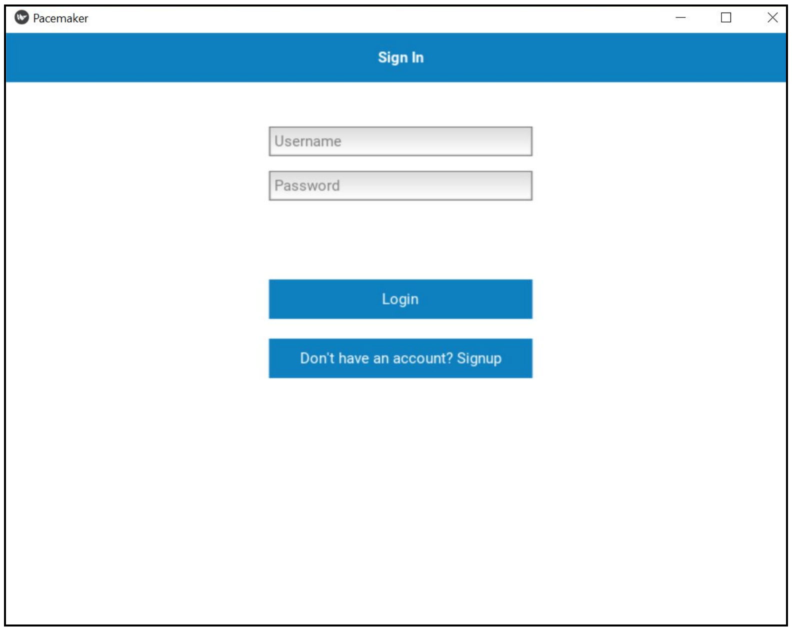


Figure 2.4.2.0 - Interface of Login Menu Module

A. The purpose of the module is to allow the user to log in to the Pacemaker App. The login menu is the first module the user sees on the startup of the Pacemaker App. This module is a simple form of encryption to prevent unregistered users from accessing the service provided by the application.

B. The secret of the module is the detection of connected USB devices when at the main menu. Clicking the calibration button determines the availability of a pacemaker board, and sets the appropriate pacing modes into effect.

C. See *Table 2.4.2.0 - Tabular Expression of Login Menu Module*.

Table 2.4.2.0 - Tabular Expression of Login Menu Module:

Functions	**Parameters** Two types: Direct, which means the function directly gets variable(s). Indirect, which means the function gets

	references to variable(s) which are defined inside the function For example: "def functionName(self):" Where "self" refers to the contents of the currently visible menu.
check_user	- Username Textbox Text (Indirect) - Password Textbox Text (Indirect) - Info Text Label (Indirect)
register_call	- Info Text Label (Indirect)
clearContents	- Username Textbox Text (Indirect) - Password Textbox Text (Indirect) - Info Text Label (Indirect)

check_user:

a. User clicks the "login" button which calls the function. The user is transitioned to the Main Menu module (class) if the user typed in a correct username and password, if not an Info Text is updated for the user to read.

b. No Global Variables between functions. The data structure is in the form of a txt file called "users.txt". This data structure contains all the usernames and passwords of all registered users using the Pacemaker App. The txt file is organized as a username followed by the corresponding password after a space in each line.

c. Function is private with respect to other modules (classes).

d. The function opens up the "users.txt" file to read from and enters a for loop for each line in the txt file. Each line is split into two parts at a point where a space string is found to represent the username (first part), and password (second part). If the username and password textbox text values match any username and password obtained from the txt file, break out of the for loop immediately then prompt the user about a successful login by changing the Info Text label text, and go to the Main Menu Module (class). In addition to this, the Username Textbox Text, Password Textbox Text, Info Text Label are all cleared by calling the **clearContents** function. If not, prompt the user about an unsuccessful login by changing the Info Text label text.

register_Call:

a. User clicks the "Signup Button" which calls the function. The user is transitioned to the Register Menu Module (class) if there are less than 10 registered users. If there are 10 registered users, then an error prompt pops up for the user.
b. No Global Variables between functions. The data structure is the same one mentioned in check_user function under part e.
c. Function is private with respect to other modules (classes).
d. The function first opens up the "user.txt" file to read and write to. The lines of the file are counted one by one. If the lines add up to a total value of 10, the Info Text Label text is changed to prompt the user about an error. If not, the Info Text Label text is changed to prompt the user about a successful registration, and the user is transferred to the Register Menu Module (class). In addition to this, the Username Textbox Text, Password Textbox Text, Info Text Label are all cleared by calling the **clearContents** function.

clearContents:

a. Username Textbox, Password Textbox, and Info Label are all cleared for the user if they decide to logout again.

b. No Global Variables between functions. No data structure used.

c. Function is private with respect to other modules (classes).

d. The function is called from the **check_users** and **register_Call** functions. Clears the Username Textbox Text, Password Textbox Text, and the Info Text Label Text.

2.4.3 - Registration Menu Module

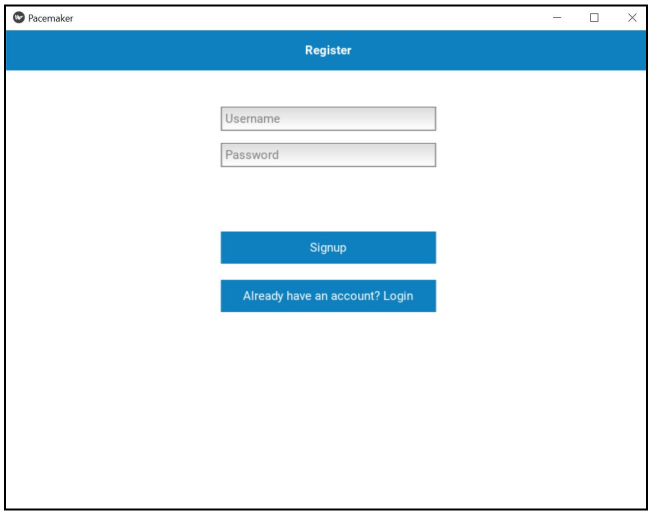


Figure 2.4.3.0 - Interface of Registration Menu Module

A. The purpose of the Register Menu Module is to allow the user to register into the Pacemaker App if they do not have an account already.

B. Not Applicable.

C. See *Table 2.4.3.0 - Tabular Expression of Registration Menu Module*.

Table 2.4.3.0 - Tabular Expression of Registration Menu Module:

Functions	**Parameters** Two types: Direct, which means the function directly gets variable(s). Indirect, which means the function gets references to variable(s) which are defined inside the function For example: “def functionName(self):” Where “self” refers to the contents of the currently visible menu.
check_user	- Username Textbox Text (Indirect) - Password Textbox Text (Indirect) - Info Text Label (Indirect)
login_call	None
clearContents	- Username Textbox Text (Indirect) - Password Textbox Text (Indirect) - Info Text Label (Indirect)
createStates	None

check_user:

a. User clicks the “Sign in” button which calls the function. The user is transitioned to the Main Menu module (class) if the user typed in a correct username and password, if not an error is prompted to the user.

b. No Global Variables between functions. The data structure is in the form of a txt file called “users.txt”. This data structure contains all the usernames and passwords of all registered users using the Pacemaker App. The txt file is organized as a username followed by the corresponding password after a space in each line.

c. Function is private with respect to other modules (classes).

d. The function first checks if the Username and Password Text Boxes contain a space, since it may mess up how the text files read from the "users.txt" files. If it does contain a space, prompt the user with error in the Info Text Label. If not, the function opens up the "users.txt" file to read from and enters a for loop for each line in the txt file. Each line is split into two parts at a point where a space string is found to represent the username (first part), and password (second part). If the username is already contained in the txt file, then break out and prompt the user about an error through the Info Text Label. If not, then write the Username and Password in the Textboxes into the first empty line of the txt file, prompt the user about successful login through the Info Text Label, clear contents of the Textboxes and Labels by calling the **clearContents** function and transition to the Main Menu Module. The **createStates** function is also called to create all the data files for the states.

login_Call:

a. User clicks the "Already have an account? Login" Button and transitions to the Login Menu Module (class).

b. No Global Variables between functions. No data structure used.

c. Function is private with respect to other modules (classes).

d. The function transitions the user to the Login Menu Module (class).

clearContents:

a. Username Textbox, Password Textbox, and Info Label are all cleared for the user if they decide to logout again.
b. No Global Variables between functions. No data structure used.
c. Function is private with respect to other modules (classes).
d. The function is called from the **check_users** and **login_Call** functions. Clears the Username Textbox Text, Password Textbox Text, and the Info Text Label Text.

createStates:

a. The state files are created when the user registers. These files contain names of the users who registered, followed by their individual inputs for each state.
b. The global variable user is gotten which contains the user's username. The data structure is in the form of a txt file called the respective pacing name followed by ".txt". For example, "AOO.txt". This data structure contains all the Programmable Parameter values (mentioned in the Function-Parameter Table of the Pacing Mode Menu Modules below) in each line.
c. Function is private with respect to other modules (classes).
d. The function is called from the **check_users** function. Creates a new set of parameter values in the data structure text files for the new users. The parameters are set to their nominal default values.

2.4.4 - Main Menu Module

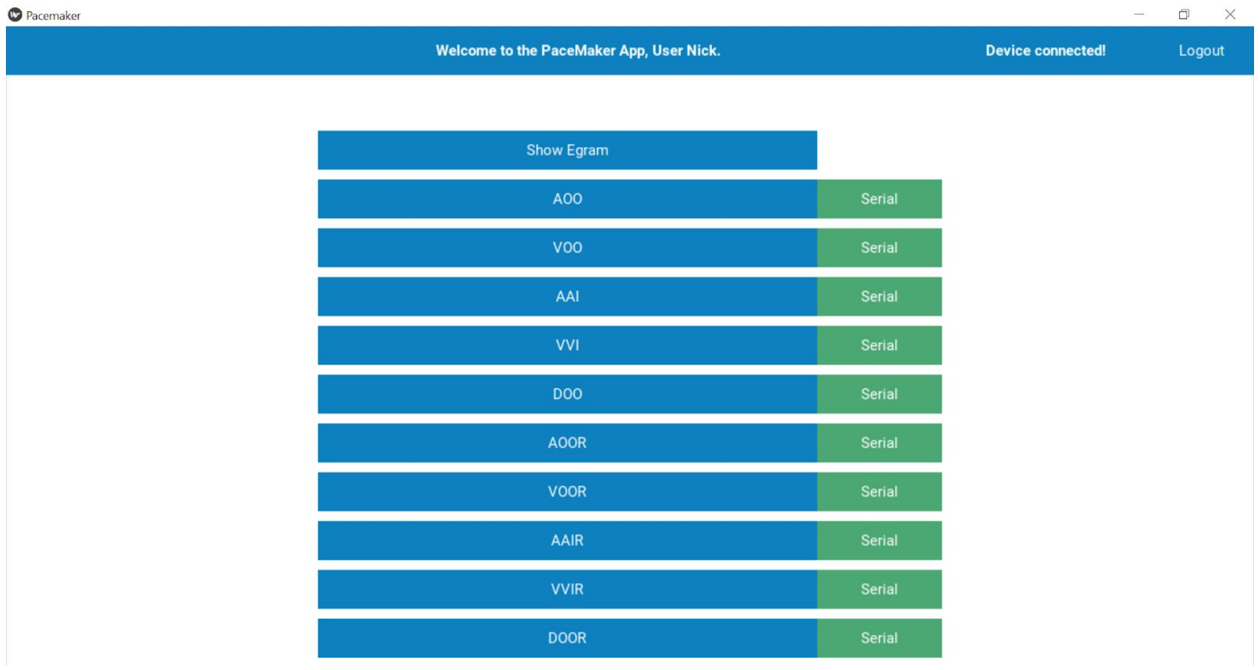


Figure 2.4.4.0 - Interface of Main Menu Module

A. The purpose of the Main Menu Module is to allow the user to access different pacing modes of the Pacemaker, calibrate their device and upload the pacing modes to the Pacemaker, or logout.

B. When the user transitions to the Main Menu Module, a function activates without the user knowing. This function is called **on_pre_enter** (defined below).

C. See *Table 2.4.4.0 - Tabular Expression of Main Menu Module*

Table 2.4.4.0 - Tabular Expression of Main Menu Module:

<u>Functions</u>	**<u>Parameters</u>** Two types: Direct, which means the function directly gets variable(s). Indirect, which means the function gets references to variable(s) which are defined inside the function

	For example: “def functionName(self):” Where “self” refers to the contents of the currently visible menu.
on_pre_enter	- Welcome Message Text (Indirect)
on_enter	None
logout_Call	None
egram_Call	None
<u>Set of State_Call Functions:</u> AOO_Call VOO_Call AAI_Call VVI_Call DOO_Call AOOR_Call VOOR_Call AAIR_Call VVIR_Call DOOR_Call	None
<u>Set of SerialState_Call Functions:</u> SerialAOO_Call SerialVOO_Call SerialAAI_Call SerialVVI_Call SerialDOO_Call SerialAOOR_Call SerialVOOR_Call SerialAAIR_Call SerialVVIR_Call SerialDOOR_Call	None
SerialCheck	- connectionPrompt Label Text (indirect) - All the Serial Button Label Text (indirect)
SerialConnect	None
SerialSend	None

on_pre_enter:

a. When the user transitions to the they will see their username in the top right of the welcome message.
b. Uses a global variable called user which contains the username of the currently logged in user. No data structure.
c. Function is private with respect to other modules (classes).
d. The function updates the Welcome label with the username concatenated with it.

on_enter:

e. When the user transitions to the Main Menu Module, if the user plugged in the pacemaker board before logging in, the user will see that the "Device not connected" label will now say "Device connected!".
f. Global variable called "deviceConnected" is shared with the **on_enter** function. No data structure required (always checks when transitioning to the main menu).
g. Function is private with respect to other modules (classes).
h. The function calls the function **serial_check**

logout_Call:

a. User clicks the "Logout" Button which transitions the user to the Login Menu Module (class)
b. No Global Variables between functions. No data structure used.
c. Function is private with respect to other modules (classes).
d. The function transitions to the Login Menu Module (class).

egram_Call:

e. User clicks the "Show Egram" Button which pops up the Egram Popup window.

f. Uses global variable popupWindow so that python can recognize the popup as the egram window. No data structure used.
g. Function is private with respect to other modules (classes).
h. The function triggers when the “Show Egram” button is clicked and generates a popupwindow with an identical structure as the Egram Window defined in the Pacemaker.kv file.

State_Call:

i. User clicks the corresponding State Button which pops up the corresponding State Module Window.
j. None.
k. Function is private with respect to other modules (classes).
l. The function triggers when the corresponding State Button is clicked and transitions to the corresponding State Module Window.

SerialState_Call:

m. User clicks the “Serial” Button beside the corresponding State Button which seemingly does nothing. But the pacemaker updates with the corresponding parameter values of the state which was beside the Serial Button.
n. Refer to the Pacing Modes Menu Modules Tabular Expressions doc for the global variables (replace word TextBoxes with Global Variables). Data Structure for the respective states are used.
o. Function is private with respect to other modules (classes).
p. The function opens up the corresponding state txt file for reading and grabs the values which the current user has and serial communicates them to the pacemaker

board by calling the function **serialSend**.

serialCheck_Call:

a. None, users cannot interact with this function directly. It is responsible for changing the text to device connected when a pacemaker device is plugged in.
b. Checks the Global Variable deviceConnected to see if a device is connected. No data structures used.
c. Function is private with respect to other modules (classes).
d. The function calles the function serialConnect to see if a device is connected. It uses the global variable deviceConnected to see if a device is connected and updates the color of the serial buttons and connectionPrompt accordingly.

serialConnect_Call:

e. None, users cannot interact with this function directly.
f. Updates the Global Variable deviceConnected when a device is connected and updates the serialPort global variable. No data structures used.
g. Function is private with respect to other modules (classes).
h. Function opens up the serial stream at all the com ports and checks to see if they are connected to 1 by 1. It then updates the deviceConnected and serialPort global variable accordingly.

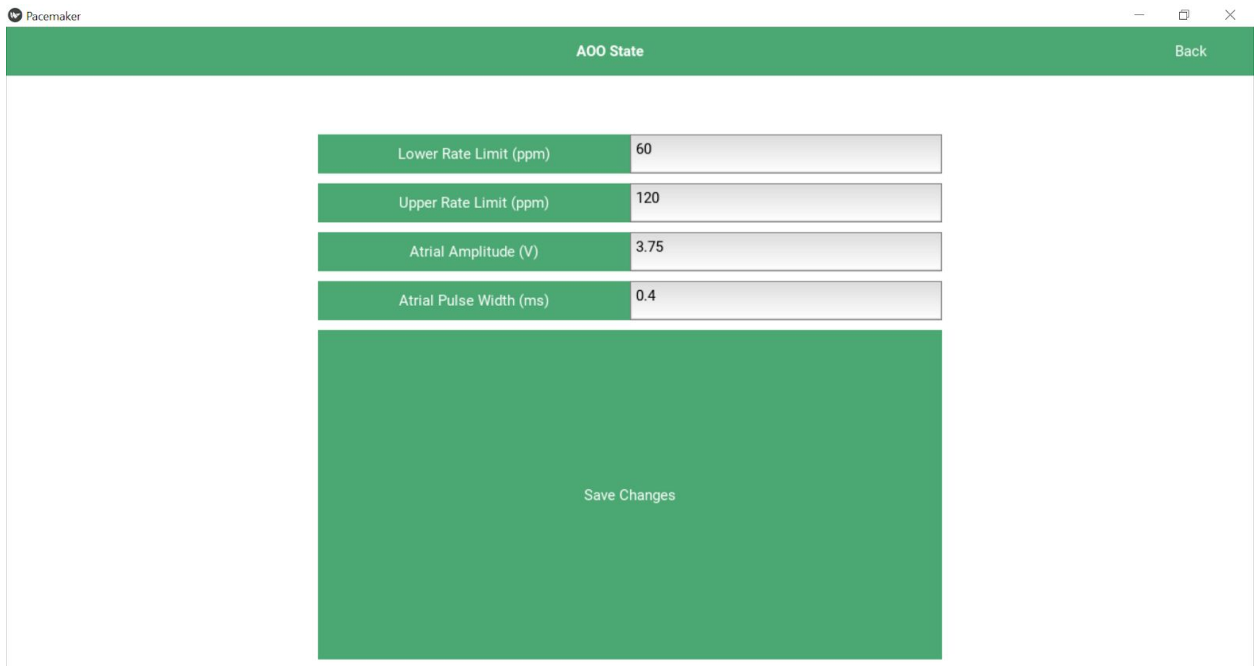
serialSend_Call:

i. None, users cannot interact with this function directly.
j. Global Variable serialPort is used to send data to the correct port and the data from the Global Variables for all the programmable parameters are sent through the serial port. No data structures used.

k. Function is private with respect to other modules (classes).

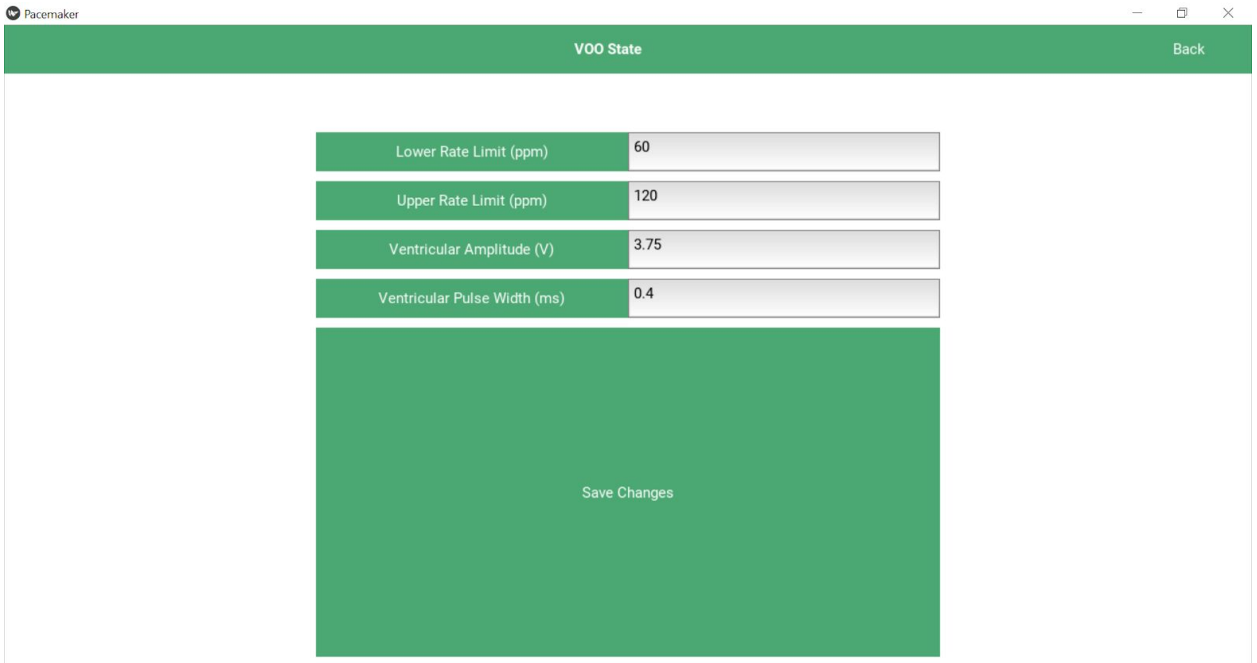
l. The function opens up the port which was referred by the serialPort global variable and sends all the global variable parameter data which was used through it.

2.4.4 - Pacing-Mode (AOO, VOO, AAI, and VVI) Menu Modules



Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Atrial Amplitude (V)	3.75
Atrial Pulse Width (ms)	0.4

Figure 2.4.4.0 - Interface of AOO Pacing Mode


Pacemaker

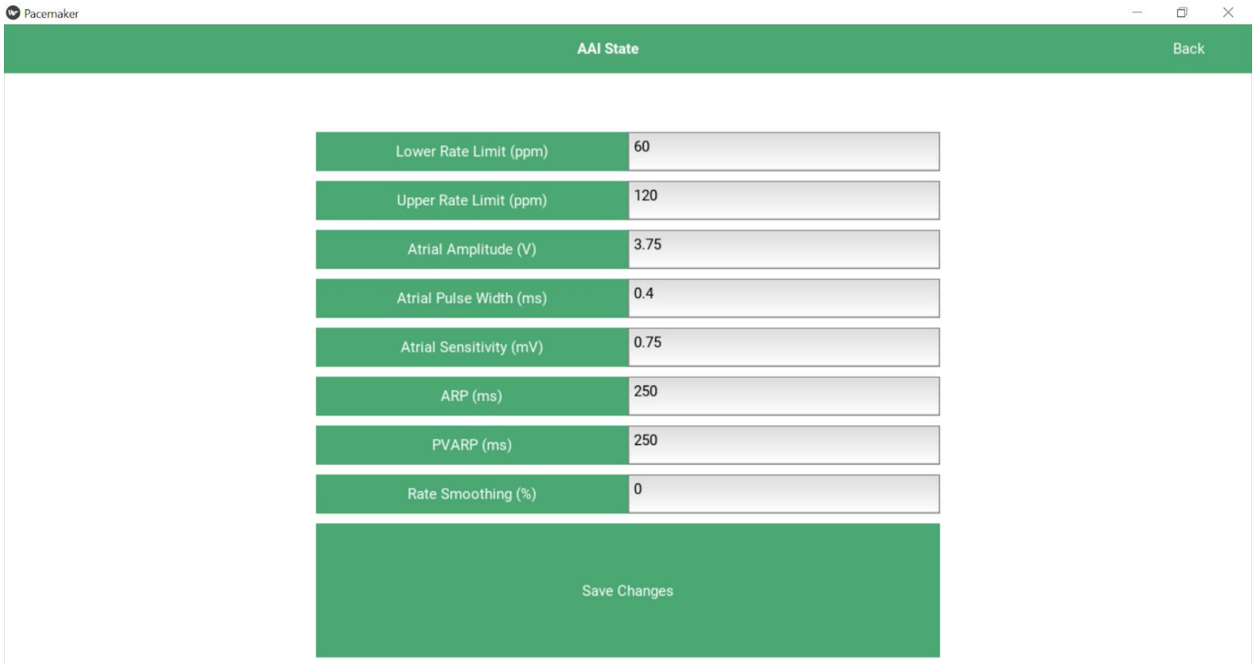
VOO State

Back

Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Ventricular Amplitude (V)	3.75
Ventricular Pulse Width (ms)	0.4

Save Changes

Figure 2.4.4.1 - Interface of VOO Pacing Mode


Pacemaker

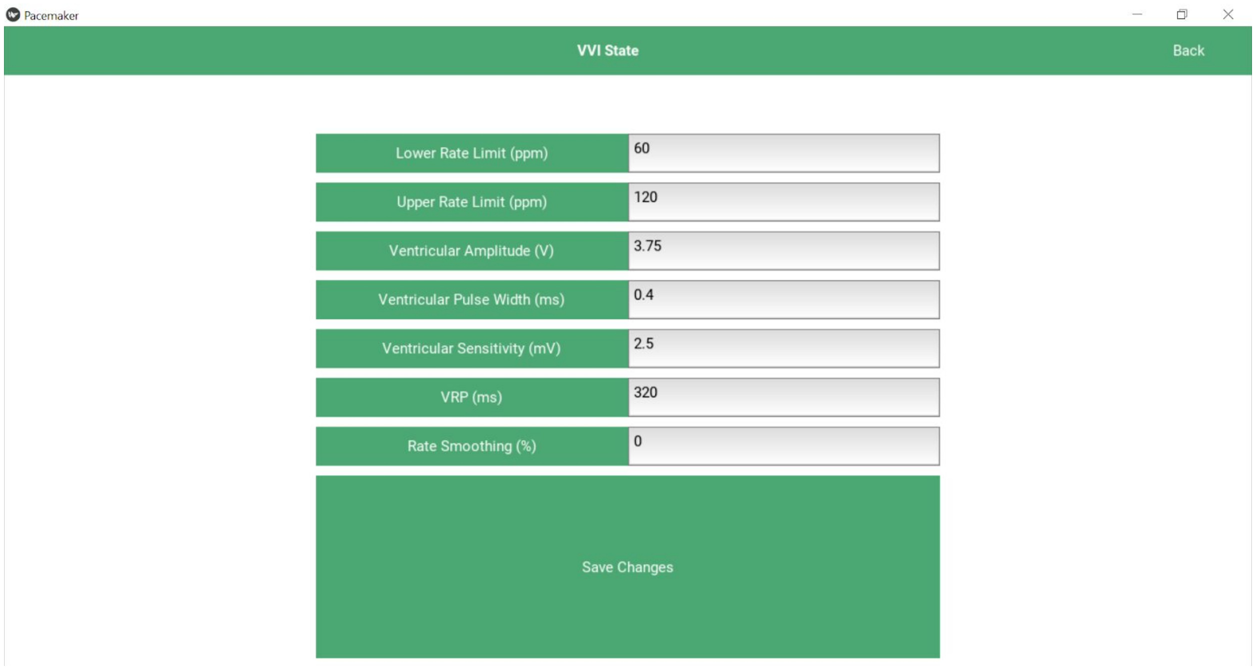
AAI State

Back

Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Atrial Amplitude (V)	3.75
Atrial Pulse Width (ms)	0.4
Atrial Sensitivity (mV)	0.75
ARP (ms)	250
PVARP (ms)	250
Rate Smoothing (%)	0

Save Changes

Figure 2.4.4.2 - Interface of AAI Pacing Mode



Lower Rate Limit (ppm)	60
Upper Rate Limit (ppm)	120
Ventricular Amplitude (V)	3.75
Ventricular Pulse Width (ms)	0.4
Ventricular Sensitivity (mV)	2.5
VRP (ms)	320
Rate Smoothing (%)	0

Figure 2.4.4.3 - Interface of VVI Pacing Mode

A. The purpose of the Pacing-Mode Menu Modules is to allow the user to set the different programmable parameters for the pacing modes.

B. When the user transitions to the Pacing-Mode Menu Modules, a function activates without the user knowing. This function is called **on_pre_enter** (defined below).

C. See *Table 2.4.4.0 - Tabular Expression of Pacing Mode Menu Modules*.

Table 2.4.4.1 - Tabular Expression of Pacing Mode Menu Modules:

Functions	**Parameters** Two types: Direct, which means the function directly gets variable(s). Indirect, which means the function gets references to variable(s) which are defined inside the function For example: "def functionName(self):" Where "self" refers to the contents of the currently visible menu.

on_pre_enter	<u>Programmable Parameters Non Rate Adaptive Modes:</u>
save_Call	

<u>AOO</u>	<u>AAI</u>	<u>VOO</u>	<u>VVI</u>	<u>DOO</u>
Lower Rate Limit Textbox				
Upper Rate Limit Textbox				
-	-	-	-	Fixed AV Delay Textbox
Atrial Amplitude Textbox		-	-	Atrial Amplitude Textbox
-	-	Ventricular Amplitude Textbox		
Atrial Pulse Width Textbox		-	-	Atrial Pulse Width Textbox
		Ventricular Pulse Width Textbox		
-	Atrial Sensitivity Textbox	-	Ventricular Sensitivity Textbox	-
-	ARP Textbox	-	VRP Textbox	-
-	PVARP Textbox	-	-	-
-	Rate Smoothing Textbox	-	Rate Smoothing Textbox	-

<u>Programmable Parameters Rate Adaptive Modes:</u>

<u>AOOR</u>	<u>AAIR</u>	<u>VOOR</u>	<u>VVIR</u>	<u>DOOR</u>
Lower Rate Limit Textbox				
Upper Rate Limit Textbox				
Maximum Sensor Rate Textbox				

-	-	-	-	Fixed AV Delay Textbox
Atrial Amplitude Textbox		-	-	Atrial Amplitude Textbox
-	-	Ventricular Amplitude Textbox		
Atrial Pulse Width Textbox		-	-	Atrial Pulse Width Textbox
-	-	Ventricular Pulse Width Textbox		
-	Atrial Sensitivity Textbox	-	Ventricular Sensitivity Textbox	-
-	ARP Textbox	-	VRP Textbox	-
-	PVARP Textbox	-	-	-
-	Rate Smoothing	-	Rate Smoothing	-
Activity Threshold Textbox				
Reaction Time Textbox				
Response Factor Textbox				
Recovery Time Textbox				

Info Text Label

back_Call	None

on_pre_enter:

a. When a user enters any of the Pacing-Mode Menu Modules, the previous values of the Programmable Parameters (mentioned in the Function-Parameter Table above) which they saved show up in the text boxes.

b. No Global Variables between functions. The data structure is in the form of a txt file called the respective pacing name followed by ".txt". For example, "AOO.txt" . This data structure contains all the Programmable Parameter values (mentioned in the Function-Parameter Table above) in each line.
c. Function is private with respect to other modules (classes).
d. The function first opens up the corresponding pacing mode ".txt" for reading. Each line of the file is read and the values are then directly inputted into their corresponding Programmable Parameter Textboxes.

save_Call:

a. User clicks the "Save" button, the values of the Programmable parameters (mentioned in the Function-Parameter Table above) are stored in the corresponding .txt file.
b. The Global Variable calle user is used which gets the user's username. The data structure is in the form of a txt file called the respective pacing name followed by ".txt". For example, "AOO.txt". This data structure contains all the Programmable Parameter values (mentioned in the Function-Parameter Table above) in each line with the user name of the user at the top of their respective parameters.
c. Function is private with respect to other modules (classes).
d. The function first opens up the corresponding pacing mode ".txt" for writing. The values of the corresponding Programmable Parameter textboxes are first checked to see if they contain an error. If an error appears, prompt the user by updating the Info Text Label. If no error occurs, prompt the user about success by updating the Info Text Label and write the values of the Programmable Parameter Textboxes

into the corresponding ".txt" file.

back_Call:

a. User clicks the "Back" Button which transitions the user to the Main Menu Module (class)
b. No Global Variables between functions. No data structure used.
c. Function is private with respect to other modules (classes).
d. The function transitions to the Main Menu Module (class).

2.4.4 - Egram Popup Window Module

A. The purpose of the Egram Module is to view the Electrograms of the Atrial and Ventricular portions of the heart.
B. None.
C. See *Table 2.4.4.0 - Tabular Expression of Egram Popup Window Module*.

Table 2.4.4.2 - Tabular Expression of Pacing Mode Menu Modules:

Functions	**Parameters** Two types: Direct, which means the function directly gets variable(s). Indirect, which means the function gets references to variable(s) which are defined inside the function For example: "def functionName(self):" Where "self" refers to the contents of the currently visible menu.
start	None
ATRReceive	None
VENTReceive	None
stop	None

start:

e. When the Start Button is clicked, it displays the electrograms for the Atrial and Ventricular accordingly.
f. Global Variable deviceConnected is used to check if the device is connected first. No data structure is used.
g. Function is private with respect to other modules (classes).
h. The function calls the **ATRReceive and the VENTReceive** continuously to update the heart graphs with the corresponding voltages over time.

ATRReceive / VENTReceive:

e. None.
f. Uses the Global Variable ATRVENTVALUES to share both the information with each other. No data structures are used.
g. Function is private with respect to other modules (classes).
h. The functions call the pacemaker device serially and continuously reads in data from the input streams, organizes them as atrial or ventricular values and outputs them over time accordingly.

stop:

e. User clicks the "Stop" Button which stops the graph from updating.
f. No Global Variables between functions. No data structure used.
g. Function is private with respect to other modules (classes).
h. The function stops the continuous calls of the ATRReceive and the VENTReceive functions.

2.4.5 - Tabular Expressions

Table 2.4.5.0 - Tabular Expression of check_user:

Initially: valid = False		check_user
username == credentials[0]	password == credentials[1]	valid = True

2.4.5.0 - Tabular Expressions of on_pre_enter

Table 2.4.5.0.0 - Tabular Expression of on_pre_enter in AOOWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.AAText.text = line.replace("\n","")
counter == 3	self.ids.APWText.text = line.replace("\n","")

Table 2.4.5.0.1 - Tabular Expression of on_pre_enter in VOOWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.VAText.text = line.replace("\n","")
counter == 3	self.ids.VPWText.text = line.replace("\n","")

Table 2.4.5.0.2 - Tabular Expression of on_pre_enter in AAIWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")

counter == 2	self.ids.AAText.text = line.replace("\n","")
counter == 3	self.ids.APWText.text = line.replace("\n","")
counter == 4	self.ids.ASText.text = line.replace("\n","")
counter == 5	self.ids.ARPText.text = line.replace("\n","")
counter == 6	self.ids.PVARPText.text = line.replace("\n","")
counter == 7	self.ids.RSText.text = line.replace("\n","")

Table 2.4.5.0.3 - Tabular Expression of on_pre_enter in VVIWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.VAText.text = line.replace("\n","")
counter == 3	self.ids.VPWText.text = line.replace("\n","")
counter == 4	self.ids.VSText.text = line.replace("\n","")
counter == 5	self.ids.VRPText.text = line.replace("\n","")
counter == 6	self.ids.RSText.text = line.replace("\n","")

Table 2.4.5.0.4 - Tabular Expression of on_pre_enter in DOOWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.FAVDText.text = line.replace("\n","")
counter == 3	self.ids.AAText.text = line.replace("\n","")
counter == 4	self.ids.VAText.text = line.replace("\n","")
counter == 5	self.ids.APWText.text = line.replace("\n","")
counter == 6	self.ids.VPWText.text = line.replace("\n","")

Table 2.4.5.0.5 - Tabular Expression of on_pre_enter in AOORWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.MSRText.text = line.replace("\n","")
counter == 3	self.ids.AAText.text = line.replace("\n","")
counter == 4	self.ids.APWText.text = line.replace("\n","")
counter == 5	self.ids.ATText.text = line.replace("\n","")
counter == 6	self.ids.RT1Text.text = line.replace("\n","")
counter == 7	self.ids.RFText.text = line.replace("\n","")
counter == 8	self.ids.RT2Text.text = line.replace("\n","")

Table 2.4.5.0.6 - Tabular Expression of on_pre_enter in VOORWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.MSRText.text = line.replace("\n","")
counter == 3	self.ids.VAText.text = line.replace("\n","")
counter == 4	self.ids.VPWText.text = line.replace("\n","")
counter == 5	self.ids.ATText.text = line.replace("\n","")
counter == 6	self.ids.RT1Text.text = line.replace("\n","")
counter == 7	self.ids.RFText.text = line.replace("\n","")
counter == 8	self.ids.RT2Text.text = line.replace("\n","")

Table 2.4.5.0.7 - Tabular Expression of on_pre_enter in AAIRWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.MSRText.text = line.replace("\n","")
counter == 3	self.ids.AAText.text = line.replace("\n","")
counter == 4	self.ids.APWText.text = line.replace("\n","")
counter == 5	self.ids.ASText.text = line.replace("\n","")
counter == 6	self.ids.ARPText.text = line.replace("\n","")
counter == 7	self.ids.PVARPText.text = line.replace("\n","")
counter == 8	self.ids.RSText.text = line.replace("\n","")
counter == 9	self.ids.ATText.text = line.replace("\n","")
counter == 10	self.ids.RT1Text.text = line.replace("\n","")
counter == 11	self.ids.RFText.text = line.replace("\n","")
counter == 12	self.ids.RT2Text.text = line.replace("\n","")

Table 2.4.5.0.8 - Tabular Expression of on_pre_enter in VVIRWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.MSRText.text = line.replace("\n","")
counter == 3	self.ids.VAText.text = line.replace("\n","")
counter == 4	self.ids.VPWText.text = line.replace("\n","")
counter == 5	self.ids.VSText.text = line.replace("\n","")
counter == 6	self.ids.VRPText.text = line.replace("\n","")
counter == 7	self.ids.RSText.text = line.replace("\n","")

counter == 8	self.ids.ATText.text = line.replace("\n","")
counter == 9	self.ids.RT1Text.text = line.replace("\n","")
counter == 10	self.ids.RFText.text = line.replace("\n","")
counter == 11	self.ids.RT2Text.text = line.replace("\n","")

Table 2.4.5.0.9 - Tabular Expression of on_pre_enter in DOORWindow:

	on_pre_enter
counter == 0	self.ids.LRLText.text = line.replace("\n","")
counter == 1	self.ids.URLText.text = line.replace("\n","")
counter == 2	self.ids.MSRText.text = line.replace("\n","")
counter == 3	self.ids.FAVDText.text = line.replace("\n","")
counter == 4	self.ids.AAText.text = line.replace("\n","")
counter == 5	self.ids.VAText.text = line.replace("\n","")
counter == 6	self.ids.APWText.text = line.replace("\n","")
counter == 7	self.ids.VPWText.text = line.replace("\n","")
counter == 8	self.ids.ATText.text = line.replace("\n","")
counter == 9	self.ids.RT1Text.text = line.replace("\n","")
counter == 10	self.ids.RFText.text = line.replace("\n","")
counter == 11	self.ids.RT2Text.text = line.replace("\n","")

2.4.5.1 - Tabular Expressions for save_Call

Table 2.4.5.1.0 - Tabular Expression of save_Call in AOOWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and	errorPrompt.text =

int(self.ids.LRLText.text) < 176)]	'[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text))	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
if self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[if self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Amplitude.[/color]'
self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Pulse Width.[/color]'
checker == 4	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.1 - Tabular Expression of save_Call in VOOWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR:

	Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text))	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05 and float(self.ids.VPWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05 and float(self.ids.VPWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Pulse Width.[/color]'
checker == 4	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.2 - Tabular Expression of save_Call in AAIWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate

	Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text))	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Amplitude.[/color]'
self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Pulse Width.[/color]'
self.isfloat(self.ids.ASText.text) and float(self.ids.ASText.text) >= 0.25 and float(self.ids.ASText.text) <= 10	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.ASText.text) and float(self.ids.ASText.text) >= 0.25 and float(self.ids.ASText.text) <= 10]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Sensitivity.[/color]'
self.ids.ARPText.text.isnumeric() and (int(self.ids.ARPText.text) >= 150 and int(self.ids.ARPText.text) <= 500)	errorPrompt = self.ids.infoText
NOT[self.ids.ARPText.text.isnumeric() and (int(self.ids.ARPText.text) >= 150 and int(self.ids.ARPText.text) <= 500)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid ARP.[/color]'
self.ids.PVARPText.text.isnumeric() and (int(self.ids.PVARPText.text) >= 150 and int(self.ids.PVARPText.text) <= 500)	errorPrompt = self.ids.infoText
NOT[self.ids.PVARPText.text.isnumeric() and (int(self.ids.PVARPText.text) >= 150 and int(self.ids.PVARPText.text) <= 500)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid PVARP.[/color]'
self.ids.RSText.text.isnumeric() and (int(self.ids.RSText.text) >= 0 and int(self.ids.RSText.text) <= 21)	errorPrompt = self.ids.infoText

NOT[self.ids.RSText.text.isnumeric() and (int(self.ids.RSText.text) >= 0 and int(self.ids.RSText.text) <= 21)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Rate Smoothing.[/color]'
checker == 8	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.3 - Tabular Expression of save_Call in VVIWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text))	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]
self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Amplitude.[/color]'
self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05 and float(self.ids.VPWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05	errorPrompt.text =

and float(self.ids.VPWText.text) <= 1.9]	'[color=#FF0000]ERROR: Invalid Ventricular Pulse Width.[/color]'
self.isfloat(self.ids.VSText.text) and float(self.ids.VSText.text) >= 0.25 and float(self.ids.VSText.text) <= 10	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VSText.text) and float(self.ids.VSText.text) >= 0.25 and float(self.ids.VSText.text) <= 10]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Sensitivity.[/color]'
self.ids.VRPText.text.isnumeric() and (int(self.ids.VRPText.text) >= 150 and int(self.ids.VRPText.text) <= 500)	errorPrompt = self.ids.infoText
NOT[self.ids.VRPText.text.isnumeric() and (int(self.ids.VRPText.text) >= 150 and int(self.ids.VRPText.text) <= 500)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid VRP.[/color]'
self.ids.RSText.text.isnumeric() and (int(self.ids.RSText.text) >= 0 and int(self.ids.RSText.text) <= 21)	errorPrompt = self.ids.infoText
NOT[self.ids.RSText.text.isnumeric() and (int(self.ids.RSText.text) >= 0 and int(self.ids.RSText.text) <= 21)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Rate Smoothing.[/color]'
checker == 7	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.4 - Tabular Expression of save_Call in DOOWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText

NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text))	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Amplitude.[/color]'
if self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[if self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Amplitude.[/color]'
self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Pulse Width.[/color]'
self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05 and float(self.ids.VPWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05 and float(self.ids.VPWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Pulse Width.[/color]'
checker == 7	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.5 - Tabular Expression of save_Call in AOORWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text)):	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Maximum Response Rate.[/color]'
self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Amplitude.[/color]'
self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Pulse Width.[/color]'

self.ids.ATText.text.lower() == "v-low" or self.ids.ATText.text.lower() == "low" or self.ids.ATText.text.lower() == "med-low" or self.ids.ATText.text.lower() == "med" or self.ids.ATText.text.lower() == "med-high" or self.ids.ATText.text.lower() == "high" or self.ids.ATText.text.lower() == "v-high"	errorPrompt = self.ids.infoText
NOT[self.ids.ATText.text.lower() == "v-low" or self.ids.ATText.text.lower() == "low" or self.ids.ATText.text.lower() == "med-low" or self.ids.ATText.text.lower() == "med" or self.ids.ATText.text.lower() == "med-high" or self.ids.ATText.text.lower() == "high" or self.ids.ATText.text.lower() == "v-high"]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Activity Threshold.[/color]'
self.ids.RT1Text.text.isnumeric() and (int(self.ids.RT1Text.text) > 9 and int(self.ids.RT1Text.text) < 51)	errorPrompt = self.ids.infoText
NOT[self.ids.RT1Text.text.isnumeric() and (int(self.ids.RT1Text.text) > 9 and int(self.ids.RT1Text.text) < 51)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Reaction Time.[/color]'
self.ids.RFText.text.isnumeric() and (int(self.ids.RFText.text) > 0 and int(self.ids.RFText.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RFText.text.isnumeric() and (int(self.ids.RFText.text) > 0 and int(self.ids.RFText.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Response Factor.[/color]'
self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Recovery Time.[/color]'
checker == 9	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.6 - Tabular Expression of save_Call in VOORWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText

NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text))	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Maximum Response Rate.[/color]'
self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Amplitude.[/color]'
self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Recovery Time.[/color]'
checker == 9	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.7 - Tabular Expression of save_Call in AAIRWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text))	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Maximum Response Rate.[/color]'
self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Amplitude.[/color]'
self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Sensitivity.[/color]'

self.ids.ARPText.text.isnumeric() and (int(self.ids.ARPText.text) >= 150 and int(self.ids.ARPText.text) <= 500)	errorPrompt = self.ids.infoText
NOT[self.ids.ARPText.text.isnumeric() and (int(self.ids.ARPText.text) >= 150 and int(self.ids.ARPText.text) <= 500)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid ARP.[/color]'
self.ids.PVARPText.text.isnumeric() and (int(self.ids.PVARPText.text) >= 150 and int(self.ids.PVARPText.text) <= 500)	errorPrompt = self.ids.infoText
NOT[self.ids.PVARPText.text.isnumeric() and (int(self.ids.PVARPText.text) >= 150 and int(self.ids.PVARPText.text) <= 500)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid PVARP.[/color]'
self.ids.RSText.text.isnumeric() and (int(self.ids.RSText.text) >= 0 and int(self.ids.RSText.text) <= 21)	errorPrompt = self.ids.infoText
NOT[self.ids.RSText.text.isnumeric() and (int(self.ids.RSText.text) >= 0 and int(self.ids.RSText.text) <= 21)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Rate Smoothing.[/color]'
self.ids.ATText.text.lower() == "v-low" or self.ids.ATText.text.lower() == "low" or self.ids.ATText.text.lower() == "med-low" or self.ids.ATText.text.lower() == "med" or self.ids.ATText.text.lower() == "med-high" or self.ids.ATText.text.lower() == "high" or self.ids.ATText.text.lower() == "v-high"	errorPrompt = self.ids.infoText
NOT[self.ids.ATText.text.lower() == "v-low" or self.ids.ATText.text.lower() == "low" or self.ids.ATText.text.lower() == "med-low" or self.ids.ATText.text.lower() == "med" or self.ids.ATText.text.lower() == "med-high" or self.ids.ATText.text.lower() == "high" or self.ids.ATText.text.lower() == "v-high"]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Activity Threshold.[/color]'
self.ids.RT1Text.text.isnumeric() and (int(self.ids.RT1Text.text) > 9 and int(self.ids.RT1Text.text) < 51)	errorPrompt = self.ids.infoText
NOT[self.ids.RT1Text.text.isnumeric() and (int(self.ids.RT1Text.text) > 9 and int(self.ids.RT1Text.text) < 51)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Reaction Time.[/color]'
self.ids.RFText.text.isnumeric() and (int(self.ids.RFText.text) > 0 and int(self.ids.RFText.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RFText.text.isnumeric() and (int(self.ids.RFText.text) > 0 and int(self.ids.RFText.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Response Factor.[/color]'

self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Recovery Time.[/color]'
checker == 13	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.8 - Tabular Expression of save_Call in VVIRWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text)):	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Maximum Response Rate.[/color]'
self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5	errorPrompt = self.ids.infoText

NOT[self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Amplitude.[/color]'
self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05 and float(self.ids.VPWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05 and float(self.ids.VPWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Pulse Width.[/color]'
self.isfloat(self.ids.VSText.text) and float(self.ids.VSText.text) >= 0.25 and float(self.ids.VSText.text) <= 10	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VSText.text) and float(self.ids.VSText.text) >= 0.25 and float(self.ids.VSText.text) <= 10]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Sensitivity.[/color]'
self.ids.VRPText.text.isnumeric() and (int(self.ids.VRPText.text) >= 150 and int(self.ids.VRPText.text) <= 500)	errorPrompt = self.ids.infoText
NOT[self.ids.VRPText.text.isnumeric() and (int(self.ids.VRPText.text) >= 150 and int(self.ids.VRPText.text) <= 500)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid VRP.[/color]'
self.ids.RSText.text.isnumeric() and (int(self.ids.RSText.text) >= 0 and int(self.ids.RSText.text) <= 21)	errorPrompt = self.ids.infoText
NOT[self.ids.RSText.text.isnumeric() and (int(self.ids.RSText.text) >= 0 and int(self.ids.RSText.text) <= 21)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Rate Smoothing.[/color]'
self.ids.ATText.text.lower() == "v-low" or self.ids.ATText.text.lower() == "low" or self.ids.ATText.text.lower() == "med-low" or self.ids.ATText.text.lower() == "med" or self.ids.ATText.text.lower() == "med-high" or self.ids.ATText.text.lower() == "high" or self.ids.ATText.text.lower() == "v-high"	errorPrompt = self.ids.infoText
NOT[self.ids.ATText.text.lower() == "v-low" or self.ids.ATText.text.lower() == "low" or self.ids.ATText.text.lower() == "med-low" or self.ids.ATText.text.lower() == "med" or self.ids.ATText.text.lower() == "med-high" or self.ids.ATText.text.lower() == "high" or self.ids.ATText.text.lower() == "v-high"]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Activity Threshold.[/color]'
self.ids.RT1Text.text.isnumeric() and (int(self.ids.RT1Text.text) > 9 and	errorPrompt = self.ids.infoText

int(self.ids.RT1Text.text) < 51)	
NOT[self.ids.RT1Text.text.isnumeric() and (int(self.ids.RT1Text.text) > 9 and int(self.ids.RT1Text.text) < 51)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Reaction Time.[/color]'
self.ids.RFText.text.isnumeric() and (int(self.ids.RFText.text) > 0 and int(self.ids.RFText.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RFText.text.isnumeric() and (int(self.ids.RFText.text) > 0 and int(self.ids.RFText.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Response Factor.[/color]'
self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Recovery Time.[/color]'
checker == 12	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

Table 2.4.5.1.9 - Tabular Expression of save_Call in DOORWindow:

Initially: errorPrompt = self.ids.infoText	save_Call
self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.LRLText.text.isnumeric() and (int(self.ids.LRLText.text) > 29 and int(self.ids.LRLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Lower Rate Limit.[/color]'
self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.URLText.text.isnumeric() and (int(self.ids.URLText.text) > 49 and int(self.ids.URLText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
checker == 2 and int(self.ids.URLText.text) < (int(self.ids.LRLText.text)):	errorPrompt.text =

	'[color=#FF0000]ERROR: Invalid Upper Rate Limit.[/color]'
self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)	errorPrompt = self.ids.infoText
NOT[self.ids.MSRText.text.isnumeric() and (int(self.ids.MSRText.text) > 49 and int(self.ids.MSRText.text) < 176)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Maximum Response Rate.[/color]'
self.ids.FAVDText.text.isnumeric() and (int(self.ids.FAVDText.text) > 69 and int(self.ids.FAVDText.text) < 301)	errorPrompt = self.ids.infoText
NOT[self.ids.FAVDText.text.isnumeric() and (int(self.ids.FAVDText.text) > 69 and int(self.ids.FAVDText.text) < 301)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Fixed AV Delay.[/color]'
self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.AAText.text) and float(self.ids.AAText.text) > 0 and float(self.ids.AAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Amplitude.[/color]'
self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VAText.text) and float(self.ids.VAText.text) > 0 and float(self.ids.VAText.text) <= 5]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Ventricular Amplitude.[/color]'
self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.APWText.text) and float(self.ids.APWText.text) >= 0.05 and float(self.ids.APWText.text) <= 1.9]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Atrial Pulse Width.[/color]'
self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05 and float(self.ids.VPWText.text) <= 1.9	errorPrompt = self.ids.infoText
NOT[self.isfloat(self.ids.VPWText.text) and float(self.ids.VPWText.text) >= 0.05	errorPrompt.text =

and float(self.ids.VPWText.text) <= 1.9]	'[color=#FF0000]ERROR: Invalid Ventricular Pulse Width.[/color]'
self.ids.ATText.text.lower() == "v-low" or self.ids.ATText.text.lower() == "low" or self.ids.ATText.text.lower() == "med-low" or self.ids.ATText.text.lower() == "med" or self.ids.ATText.text.lower() == "med-high" or self.ids.ATText.text.lower() == "high" or self.ids.ATText.text.lower() == "v-high"	errorPrompt = self.ids.infoText
NOT[self.ids.ATText.text.lower() == "v-low" or self.ids.ATText.text.lower() == "low" or self.ids.ATText.text.lower() == "med-low" or self.ids.ATText.text.lower() == "med" or self.ids.ATText.text.lower() == "med-high" or self.ids.ATText.text.lower() == "high" or self.ids.ATText.text.lower() == "v-high"]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Activity Threshold.[/color]'
self.ids.RT1Text.text.isnumeric() and (int(self.ids.RT1Text.text) > 9 and int(self.ids.RT1Text.text) < 51)	errorPrompt = self.ids.infoText
NOT[self.ids.RT1Text.text.isnumeric() and (int(self.ids.RT1Text.text) > 9 and int(self.ids.RT1Text.text) < 51)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Reaction Time.[/color]'
self.ids.RFText.text.isnumeric() and (int(self.ids.RFText.text) > 0 and int(self.ids.RFText.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RFText.text.isnumeric() and (int(self.ids.RFText.text) > 0 and int(self.ids.RFText.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Response Factor.[/color]'
self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)	errorPrompt = self.ids.infoText
NOT[self.ids.RT2Text.text.isnumeric() and (int(self.ids.RT2Text.text) > 1 and int(self.ids.RT2Text.text) < 17)]	errorPrompt.text = '[color=#FF0000]ERROR: Invalid Recovery Time.[/color]'
checker == 12	errorPrompt.text = '[color=#00FF11]Changes Saved Successfully.[/color]'

2.5 - DCM Test Cases

2.5.1 - Login Menu Test Cases #1 (Oct. 17th, 2020)

The login Menu was tested with an empty "users.txt" file to see if the program would break. The program responded accordingly, no usernames and passwords matched according to the software, so the program would only allow registration of new users which is to be expected. The new users registered would still be inputted into the "users.txt" file.

2.5.2 - Login Menu Test Cases #2 (Oct. 17th, 2020)

The login Menu was tested with an empty value for the username and password. The program did not work as expected, if the user inputted nothing into the login menu, the program would crash. This was fixed in the newest version of code.

2.5.3 - Register Menu Test Cases #1 (Oct. 19th, 2020)

The register Menu was tested with two users with same names and different passwords, and same names and same passwords. Both cases did not work as expected, the user with the same names and different passwords failed since both could register, the program could not understand the similarities between them as an error. This error has been accounted for and fixed in the newest version of code. The second case also failed with the same exact problems mentioned with the first case. This has also been accounted for with the newest version of the code.

2.5.4 - Register Menu Test Cases #2 (Oct. 19th, 2020)

The register Menu was tested with users with no name and password. The case did not work as expected, the program allows a user with no name and password to register. This is a very big error since the user should be expected to have a value for a name and password at least. This error has been fixed in the newest version of the code.

2.5.5 - Login, Register, and Main Menu Test Cases #1 (Oct. 20th, 2020)

The program was tested with multiple logins and logouts. This is to prevent the program from getting overloaded if it was already constrained from before. The following sequences were used to test the program:

Seq 1 - login, logout, registration, logout, login, logout, registration.

Seq 2 - login, logout, login, logout, login, logout.

Seq 3 - registration, logout, registration, logout, registration, logout.

The three sequences were chosen to mimic the practical use of the program, involving multiple attempts to sign in and sign out, to create various new accounts, and a combination of the two scenarios. All cases ended up working but all sequences ended up having the values of the last login or registration username and passwords saved when they logged out again. This is not particularly problematic but it is better for the user if when they logged out the username and password they entered were not saved in the textbox. This has been changed in the newest version of code.

3.0 - Conclusion

3.1 - Assignment 1

In summary, a Matlab Simulink file was created to simulate four modes of cardiac pacing for use in a human pacemaker device. These modes were AOO, VOO, AAI, and VVI. A Python GUI was designed to provide an interface between the user and the device. While functional, there were major errors in design, namely the accidental omission of a Simulink subsystem for hardware hiding and generalization purposes. This will be rectified in future iterations. Additional planned changes include the addition of AOOR, VOOR, AAIR, VVIR, and DOOR pacing modes and expanded GUI functionality.

3.2 - Assignment 2

To summarize, the Matlab Simulink file containing the pacing modes was updated to include six additional pacing modes (DOO, AOOR, VOOR, AAIR, VVIR, DOOR). Furthermore, all pacing modes were centralized within a subsystem in the Matlab Simulink file in order to control all pacing modes using a common set of parameters. The Python GUI interface was updated to accommodate these new features. Other new DCM features include the serial communication capabilities, the egram features, and storable alterable parameters. This is currently expected to be the final iteration of the project, but hypothetical future iterations would include focus on user accessibility and aesthetics as well as increased robustness and error handling.

3.3 - Final Conclusion

Thus, it can be concluded that a model pacemaker was successfully developed using Matlab Simulink, Python, and the provided hardware. A total of ten pacing modes are built into the system, each with customizable parameters. Other features include egram functionality and LED representations of the atrium and ventricle. As a final deliverable product, the system is effective save for the size of the hardware. Even discounting the board that models the heart, the total hardware is significantly larger than commercial pacemakers. The DCM is functional and fairly user-friendly, completing the project. Future iterations would focus on size reduction, increased user-functionality, and the addition of more pacing modes.

4.0 - References

All references were provided by A. Bokhari unless otherwise noted.

4.1 - Works Cited

[1] “Creating and Masking Subsystems - Video,” *Video - MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/videos/creating-and-masking-subsystems-69025.html. [Accessed: 23-Oct-2020].

Independently procured.

[2] A. Bokhari, “SE 3K04 Assignment 1.” McMaster University, Hamilton.

[3] G. Meyer, “pacemaker_shield_explained.” McMaster University, Hamilton, 03-Jul-2020.

[4] “PACEMAKER System Specification.” Boston Scientific, 03-Jan-2007.

[5] “Pacemakers,” *National Heart Lung and Blood Institute*. [Online]. Available: https://www.nhlbi.nih.gov/health-topics/pacemakers#:~:text=A pacemaker is a small,or rhythm of the heartbeat. [Accessed: 23-Oct-2020].

Independently procured.

[6] N. K. Singh, T. S. E. Maibaum, M. Lawford, and A. Wassyng, “Stateflow To Tabular Expressions.” McMaster University, Hamilton.

[7] A. Wassyng, “srsVVI r2.” McMaster University, Hamilton, 27-Sep-2011.

[8] (n.d.). Retrieved November 02, 2020, from https://pythonprogramming.net/introduction-kivy-application-python-tutorial/

[9] (n.d.). Retrieved November 02, 2020, from https://pythontic.com/app/kivy/box layout

[10] Box Layout¶. (n.d.). Retrieved November 02, 2020, from https://kivy.org/doc/stable/api-kivy.uix.boxlayout.html

[11] Jeff, JeffJeff 36111 gold badge22 silver badges1313 bronze badges, & PalimPalimPalimPalim 2. (1966, October 01). Kivy: How do I clear the changes on a screen if they were not saved. Retrieved November 02, 2020, from https://stackoverflow.com/questions/45398937/kivy-how-do-i-clear-the-changes-on-a-screen-if-they-were-not-saved

[12] KMOKMO 2511 silver badge55 bronze badges, & Dev-I-JDev-I-J 2988 bronze badges. (1968, September 01). The kivy.uix.listview module is not found. Retrieved November 02, 2020, from https://stackoverflow.com/questions/56614885/the-kivy-uix-listview-module-is-not-found

[13] Mike. (2020, January 31). Kivy 101: How to Use BoxLayouts. Retrieved November 02, 2020, from https://www.blog.pythonlibrary.org/2013/11/25/kivy-101-how-to-use-boxlayouts/

[14] Nadeem, NadeemNadeem 1788 bronze badges, & John AndersonJohn Anderson 15.8k22 gold badges99 silver badges2525 bronze badges. (1969, July 01). How can i create a autocomplete textinput in kivy. Retrieved November 02, 2020, from https://stackoverflow.com/questions/61618734/how-can-i-create-a-autocomplete-textinput-in-kivy

[15] Ray, S. G., Bhowmick, G. D., Ghangrekar, M. M., & Mitra, A. (n.d.). Advances in wastewater treatment by combined microbial fuel cell-membrane bioreactor.

[16] Screen Manager¶. (n.d.). Retrieved November 02, 2020, from https://kivy.org/doc/stable/api-kivy.uix.screenmanager.html

[17] A. Bokhari, “Assignment2.” McMaster University, Hamilton.

[18] AdaptAdapt, “struct.error: pack expected 1 items for packing (got 4) while Packing string,” *Stack Overflow*, 01-Sep-1965. [Online]. Available: https://stackoverflow.com/questions/38430702/struct-error-pack-expected-1-items-for-packing-got-4-while-packing-string. [Accessed: 30-Nov-2020].

[19] A. Gnsaftagnsaft 1, L. Lasramllasram 3, V. Sergienko, “packing and unpacking variable length array/string using the struct module in python,” *Stack Overflow*, 01-Nov-1959. [Online]. Available: https://stackoverflow.com/questions/3753589/packing-and-unpacking-variable-length-array-string-using-the-struct-module-in-py. [Accessed: 30-Nov-2020].

[20] A. Gnsaftagnsaft 1, L. Lasramllasram 3, V. Sergienko, “packing and unpacking variable length array/string using the struct module in python,” *Stack Overflow*, 01-Nov-1959. [Online]. Available: https://stackoverflow.com/questions/3753589/packing-and-unpacking-variable-length-array-string-using-the-struct-module-in-py. [Accessed: 30-Nov-2020].

[21] Atuldo, “atuldo/real-time-plot-microphone-kivy,” *GitHub*. [Online]. Available: https://github.com/atuldo/real-time-plot-microphone-kivy/blob/master/look.kv. [Accessed: 30-Nov-2020].

[22] Atuldo, “atuldo/real-time-plot-microphone-kivy,” *GitHub*. [Online]. Available: https://github.com/atuldo/real-time-plot-microphone-kivy/blob/master/main.py. [Accessed: 30-Nov-2020].

[23] B. Balasbalas 20611 silver badge1010 bronze badges and PrestonPreston 2, “Python serial write - lost bytes,” *Stack Overflow*, 01-Oct-1963. [Online]. Available: https://stackoverflow.com/questions/25394723/python-serial-write-lost-bytes. [Accessed: 30-Nov-2020].

[24] P. H. Brubaker and D. W. Kitzman, “Chronotropic incompetence: causes, consequences, and management,” *Circulation*, 08-Mar-2011. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3065291/. [Accessed: 30-Nov-2020].

[25] “Python Global, Local and Nonlocal variables,” *Programiz*. [Online]. Available: https://www.programiz.com/python-programming/global-local-nonlocal-variables. [Accessed: 30-Nov-2020].

[26] “Python struct.pack() Examples,” *Python Examples of struct.pack*. [Online]. Available: https://www.programcreek.com/python/example/129/struct.pack. [Accessed: 30-Nov-2020].

[27] tych0, “struct.error: required argument is not an integer. · Issue #62 · tych0/xcffib,” *GitHub*. [Online]. Available: https://github.com/tych0/xcffib/issues/62. [Accessed: 30-Nov-2020].