3004 AED Design Decisions

Design Principles/Pattern related:

The use of the Model View Controller design pattern: We can see that all of our project's objects are divided up solely into the three object types. First models: those being our patient and AED classes which contain the data along with some logic which helps present the model to the user. Then we have our view object which is what our ui object since as laid out in this pattern ui doesn't know what any of the data means or handles anything about the data does it just simply presents us with the applications data in this case that data being the AED's simulation. Then finally our controller which is our mainwindow, this object is our controller since it handles all of our communications between model and view objects handling most of the applications functionality by using our other two objects.

Boundary Control Entity pattern: We can see that as laid out in the BCE pattern actors can only interact with the boundary layers which function as an interface between the user's and our code which is exactly what our ui generated and managed by our controller mainwindow does. Then we have our controller mainwindow that handles the passing and processing of data between our entity and boundary objects, once again just as described in the BCE pattern. Then finally our entities, patients and actors just as laid out in the BCE pattern these objects simply store data for us and implement some of our various associated functions.

Difference between MVC and BCE: Basically BCE pattern is primarily focused on the interactions between the various parts of a system and of the actors. While MVE is much more focused on how the system transforms data and interactions from the actors into view objects aka presentations of the application. Though of course the interactions between the various objects are still important just as with BCE. Basically while these two patterns are very similar to each other the main difference I would say is that MVC specializes from the BCE into focusing on the presentation of objects on top of the interaction between these different object types making it applicable to less applications but more detailed in now this pattern works to assist in program design.

Object related:

Patient objects: Here we use patient objects to represent the complete set of all possible patients that is we have a patient representing every possible ECG. Currently we have 7 possible patients which represent 3 adults and 3 children and one for Asystole aka flatline. It should be noted that those 3 others for each child and adult represent normal, Ventricular fibrillation and Ventricular tachycardia. The use of this object class allows us to easily track important information about our current patient and also makes doing analysis trivial for our sim.

AED object: There is just one AED object representing the AED itself. The purpose of this class is to store information important for the AED to track and to perform additional AED functions which do not need to be done in the mainwindow class that handles the user interface for our simulation. The reason for this AED object is that it helps make our code much cleaner and easier to work with as we don’t need to implement a bunch of additional functions and store additional variables in our already huge mainwindow class and it also compartmentalizes our code. This makes sense as this is a key principle of object oriented programming.

QCustomPlot object: This class was not made by our group but is a commonly used 3rd party class used to create as the name suggests custom plots. This was needed for the implementation of the ECG inorder to get a plot in Qt representing our patients heart rhyme. Additionally it enabled us to stylize the plot to look just like the actual AED devices ECG. Though in hindsight we could have just used an image as we did not animate the graph. If we were to improve this we could animate this graph to look more like the real thing which is something an image can’t do.

AED simulation related:

Handling non-shockable rhythms and resuscitated patients: If a patient has a non shockable ECG we will just keep analyzing them and telling the user to perform CPR until the device is shut off or runs out of power. If the patient is revived their heart rhythm will return to normal and the device will keep checking on them for ECG updates along with telling them to perform CPR.

Shocks and resuscitation: We gave the probability of a shock resuscitating a patient as 1/5 this means that there is a good chance of the AED to resuscitate the patient but also if you delay a good chance of running out of battery before the patient is resuscitated.

Time: To help speed things up we can see throughout the simulation stuff like battery life, CPR duration, along with the time between steps and their wait times are all vastly shorter then they would be in real life. In a realistic scenario we would make the simulation much longer and much slower to better represent an actual real world use, but for purposes of this assignment I assume we probably don’t want to wait for stuff like this. For example waiting multiple hours for the battery to die would probably not be a good use of anyone's time.
CPR: For doing CPR we made things simple by just including two conditions when CPR is not enough or if it is good enough. This means we do not include a scenario where the AED operator is pressing too hard on the patient. This is because from what we can tell this is also the case for the actual device as it only mentions when you don’t push hard enough. In addition to this for ease of simulation you are not expected to toggle off CPR when it tells you to stop CPR this is because we assume the user automatically stops doing CPR as it would be a pain in the but to need to constantly toggle off and on the CPR button.

ECG: The ECG just uses static images that represent the current patient's heart rhythm at this time. When we switch patients aka switch patient’s current condition we automatically show the new current states ECG. This was done because in order to create a fully animated ECG we would have needed to spend a very significant amount of time trying to implement this function which we did not need to, and would be solely for visual appeal since we would not be reading in data points from the pads like in the real world device. If we had actual data points that would make life very easy as all we would need is a timer to update the plot for every new data point. Additionally the ECG is only available to display when the pads are correctly connected to the patient as otherwise the system will not be able to correctly read the ECG. This also means we can read ECG when the AED is off.

User interface related:

User interface division: Our user interface can be broken in two two main areas, the AED display and the area's above and to the right which represent our test controls for the simulation. This allows us to show what the AED looks like to a user, but also gives us access to control that allows us to run the simulation and change various test scenarios. The reason for this is we want to show and to simulate how an AED would look and function but we also want a way to run the simulation along with testing various features of the AED. Thus the best way we figured to do this was create 2 distinct main sections to our user interface ensuring we keep our AED display section as authentic to its real life look as possible.

Simulation controls: As mentioned before we have simulation controls here I will explain our reasoning for each one. First we have a patient selection on top left. We added this as a way to test different patient scenarios from the user interface thus we don't need to do anything in the console. Next we have the on/off button that is needed to simulate turning the AED on and off. We decided to keep it separate from the AED display section since the power button isn't a part of the AED screen even though it is unlike our other simulation controls there is an actual button on the device. Now for the DEFIB pads we have 4 different buttons representing 4 different control settings. Adult pads and Child pads which represent the pad that was used by the AED operator in this scenario, then two more buttons which represent placing the pads correctly or incorrectly. We decided that this would be the best way to allow the simulator user to select which pads are currently being used and select if pads are being correctly or incorrectly placed. This way we keep this control panel separate from our AED display and we also don't need to go back to the console and input these settings manually, along with making use of the simulation very simple and easy for users. Now for CPR simulation controls we use a simple toggle that tells the system if the user is currently doing CPR or not then two other buttons that tell the system if the CPR is good enough or not good enough. The reason for implementing it this way is once again ease of use for our simulator and separating it from the display so the display stays true to the real life version. Now for the ECG testing controls, this one is interesting because in the real system you can enable it through the AED settings. However we decided it would be too much of an extra workload increase to try and implement a proper representation of the AED settings menu just for this one setting, thus we included this control the same way we included the rest of our simulation other settings outside of the display section. Then finally we have our last section which allows us to select which of the power on self test settings the simulator will pass or fail. This enables us to simulate if there is an issue with the AED in some way through our UI. Once again this makes things quick and easy to simulate, along with being intuitive to users.

Audio output: For our simulation the Audio outputs are being printed to the console using qInfo(). This was done due to time considerations as console outputs are acceptable for this project it wasn’t worth the extra time to try and get audio working.