Zackary T. Morelli

zackmorelli@gmail.com

7/22/2021

The Dose Objective Check script is the first large program I made at Lahey, so it has some anachronisms, and even though I’ve reorganized it, it doesn’t have the best organization and there are some methods that are long and messy.

Dose Objective Check is an Eclipse script that is used to perform plan evaluation. It assesses the Organ-at-risk (OAR) dose objectives for a plan and then pulls DVH information from Eclipse to determine if the dose objectives are passing. Some dose objectives also have goals that we try to meet. The department has standardized lists of dose objectives for each standardized treatment site. The standard dose objective lists are broken down first by conventional or SRS/SBRT plans, and then the standard treatment sites for each. The user selects this information in the GUI so the program knows what dose objective list to use. The program also generates a PDF of the results (using the open-source Migradoc/PDF library) of its analysis so that it can be easily saved and documented.

Dose Objective Check can also analyze dose coverage for targets. This is pretty complicated. There are a number of separate windows that the program uses to collect information it needs to check target coverage. It also has to deal with figuring out which structures are the targets. A lot of the logic for this is in the PDFPreparation.cs file.

There are also a number of ways that the dose objective list used by the program can be changed, or made in different ways. On the Aria server where the .dll file is saved are also two text documents called “SRSROIList” and “ConventionalROIList”. These text documents are lists of the department’s standard dose objectives, written in a particular format, for all SRS and Conventional plans, respectively. Each dose objective is a line in the document with each informational element separated by commas. The end of each line is a list of all the treatment sites that the dose objective applies to. Every time the program runs, it reads in this information from the appropriate text document (using StreamReaders) and dynamically creates a list of dose objectives (using the program’s own dose objective class) that need to be analyzed. This is done in the LISTMAKER.cs file. The dose objective list is then passed to the analysis methods (one is for plans, the other is for plansums, but they are basically the same), which loop through the list, match the dose objective to the structure that it pertains to, and then carries out the analysis. The analysis code is a long series of if…else statements, where each if statement represents a type of dose objective (like max dose, D-type, V-type, etc.). Clinical goals made in Eclipse can be added to the program’s internal dose objective list. If the user checks off the box on the GUI indicating that they want to include their clinical goals, then there is some code at the beginning of each analysis method which will pull strings representing the clinical goals, convert them to my format, and add them to the list.

There is another workflow that exists using an Aria document. This document has a table of dose objectives that is imported and exported using VBA scripts that I wrote into the document and had installed in the MS Word startup folder in the Citrix image so they actually work. They are executed using keyboard shortcuts. The import script will import dose objectives for the treatment site selected from a drop-down in the document. The user can then edit the dose objectives that have populated the table. The export script then reads the table and creates a text document of the dose objectives, using my format, which is stored in a specific location. Whenever the LISTMAKER method runs, it checks to see if one of these custom documents has been made for the patient it is running on. The user can then choose to use that document, instead of the standard one, to create the dose objective list. This allows the program to be customizable. The Aria document is meant to be filled out by the MD, but this workflow can be used by anyone.

The dose objective check has a lot of details to it, so I’m not going to attempt to explain everything. You should be able to figure it out from my comments and examining the code in VS. The program is complete really, so there shouldn’t be a need to mess with it. I’ll at least give an overview here of the control flow of the program.

It starts in the standard way with the Script being created in the ScriptExecute file and calling the main GUI. Depending on what the user does, the GUI may ask for other information like laterality, how to handle dose for plansums, and coverage objectives for targets. These are separate windows. Once the user has filled out the GUI and clicked the Execute button, the GUI determines which plan they chose, if it is a plan or a plansum, and then passes all the relevant info to the analysis methods, which are kept in the DoseObjectiveAnalysis.cs file. The analysis methods call LISTMAKER, which was described earlier. The analysis methods then return the analyzed dose objective lists, which contain information about if each one is passing or not, and the estimated dose of each. The GUI then passes the list to the PDFPreparation methods (in PDFPreparation.cs), which begin the process of making the PDF. The program continues from there.

A lot of the logic for figuring out the target coverage is done in PDFPreparation, and the program will pull additional info from Eclipse related to the targets, if the user requested that. There are also a number of awaitable WinForms that might be called to get more information about the targets from the user. The logic is separated between conventional and SRS plans. The methods in PDFPreparation have parsed through and organized all the information needed to make the PDF, they call the export method of the PDFGeneration class, which then goes about actually making the PDF using the Migradoc/PDFSharp library. There are actually a number of different classes in their own files which are called to make separate parts of the document. Once the document has been made and saved, the program starts a separate Process on the computer that opens the PDF document it just made, which will happen using whatever program is the default for .PDF (hopefully adobe reader). That way the PDF is displayed to the user in adobe reader as the output of the program.