NIST CENTER FOR NEUTRON RESEARCH	**Document:**	Report		
	Title:	MUSE Documentation		
	Revision:	Release	**Date:**	7/31/2023

NCNR Report

MCNP Uranium Scout and Editor (MUSE) Documentation

Release

	Name	Signature	Date
Drafted by:	Duncan Beauch		
Reviewed by:	Abdullah Weiss		

NIST CENTER FOR NEUTRON RESEARCH	**Document:**	Report		
	Title:	MUSE Documentation		
	Revision:	Release	**Date:**	7/31/2023

Overview

MUSE stands for MCNP Uranium Scout and Editor. This is a project designed to produce Monte Carlo N-Particle transport code (MCNP) card decks using Python. It was initiated by Duncan Beauch as part of an 11-week CORE internship at NIST with the goal of modifying the material of multiple fuel plates at the push of a button. The program is written in Python and piloted through a Dash app user interface. The main features of this project allow a user to modify types of fuel within the reactor and rearrange the fuel assembly lattice structure. The program is intended to aid reactor engineering research by facilitating the process of modifying MCNP input files.

Prerequisites

The project relies on understanding from many different softwares, libraries, and methodologies, but most prominently MCNP. The Python backbone of the code is in essence an MCNP compiler capable of recognizing and dissecting MCNP cards as described in the MCNP 6.2 Theory and User Manual. This is executed with many different types of string manipulation techniques and therefore, extensive Python knowledge is required.

Dash is a stateless Python framework for building web applications developed by Plotly. Dash is written in Python and builds off Flask, Plotly.js, and React.js. Upon launch of the app, Dash callbacks control every component interacted with by the user and logic for user requests.

Object Oriented Programming (OOP) paradigm is used throughout the project. As such, OOP in Python will be necessary to work with the codebase. The project is shaped on the Model-View-Controller (MVC) architectural pattern. The implementation of this design will be explained in the Code Architecture section of this report.

Getting Started

The project repository can be found at [MUSE · GitLab (nist.gov)](). The app and code is contained in the Template_Editor directory. The mcnp_templates directory contains MCNP input decks for the NNS and NBSR designs. The Template_Editor directory is further divided into subdirectories, most notably: models, pages, controllers as will be outlined later in the Code Architecture section. To run the project follow these steps by running the given commands:

1. Clone the Repository
    a. git clone https://gitlab.nist.gov/gitlab/duncan.beauch/PythonToMCNP
2. Navigate to the Project Directory
    a. cd [repository_name]
3. Create and Activate Virtual Environment
    a. python -m venv venv
    b. source venv/bin/activate
4. Install Dependencies
    a. pip install -r requirements.txt
5. Run the app.py
    a. python app.py

UI Guide

Upon running app.py, the program will automatically open a new window on your default browser and display the app running on your default port. The Home page will be navigated to by default with information about the app similar to the Overview section of this report. There are five main tabs on the navigation bar for the app:

1. **Home**

   This page is a description of the project and its goals. It is a great example of the general layout of a page described below.

   a. The Navigation Bar occupies the left side of the app at all times. The current page is highlighted to orient the user while using the app.
   b. The Console located at the bottom of the page is collapsible via the minimize button on its top right corner. The Console displays system messages and is used to communicate actions performed by the user or errors encountered. The Console is collapsed by default on the Home page or unknown URLs and opened when navigating to other pages unless it has been minimized. The Print File button can be pressed at any time and will print the template file read at runtime to the default file path if one is not given.
   c. The Content of each page is located at the center of the page and is dictated by the layout defined in each of the files in the pages directory. Content is the only element meant to change while navigating between pages.

2. **Plate Maker**

   This page is centered around a 3D Plotly that displays the current plate to the user. The idea of this page is to offer the user ways to edit the material composition of fuel plates in the MCNP file.

   a. The 3D Plotly Display is used to select sections of a given plate. The plot can be panned by click and dragging or zoomed with a scroll input. Objects in the plot can be highlighted when clicked by default and informs the app of selected objects.
   b. The Select Options radio buttons to the left of the 3D plot control the behavior of highlighting the selected objects in the plot. With Single Select Mode, only one object in the plot can be highlighted at once and is the default mode for Select Options. The Multiselect Mode allows more than one object to be selected at once which is most useful for plate edits. The Unselect Mode is the opposite of the Multiselect Mode, when an object is selected with this mode it will be unhighlighted in the plot.
   c. The Highlight Options serve to minimize the tedious task of mass select or unselects. The All Sections button here will highlight all sections in the plot whereas the Unselect All button will unhighlight all sections in the plot.
   d. The Preview Plate dropdown menu and related Plate textarea are used in conjunction with the 3D plot to display your current changes. Selecting one of the

scouted plates from the dropdown will display all MCNP cells that comprise it. When a change is made by the Apply Changes button, the textarea will update to reflect the change made.

e. The <u>Sections</u> label under the 3D plot displays the currently selected MCNP cell numbers. Once a plate number is selected from the Preview Plate dropdown and an object in the 3D plot is highlighted, this section will display the corresponding cell numbers that can be verified in the Plate textarea.
f. The <u>Material</u> dropdown under the Sections label allows the user to select a new material for the highlighted cells. The material numbers displayed to the user as options are material numbers corresponding to the scouted materials in the supplied MCNP input file as the template.
g. Once a material and cells are selected through the page, the user can click the <u>Apply Changes</u> button and find the Console displays the changes that were made as well as the textarea preview updated to reflect those changes.

3. Fuel Assembly

This page is identical to the Plate Maker page in design and only differs slightly. This page is centered around a 3D Plotly that displays the assemblies to the user. The idea of this page is to offer the user ways to edit the types of plates in each assembly in the MCNP file.

a. The <u>3D Plotly Display</u> is used to select either assemblies or plates depending on the selected tab above the plot. The plot can be panned by click and dragging or zoomed with a scroll input. Objects in the plot can be highlighted when clicked by default and informs the app of selected objects.
b. The <u>Select Options</u> radio buttons to the left of the 3D plot control the behavior of highlighting the selected objects in the plot. With Single Select Mode, only one object in the plot can be highlighted at once and is the default mode for Select Options. The Multiselect Mode allows more than one object to be selected at once which is most useful for plate edits. The Unselect Mode is the opposite of the Multiselect Mode, when an object is selected with this mode it will be unhighlighted in the plot.
c. The <u>Highlight Options</u> serve to minimize the tedious task of mass select or unselects. The All Assemblies button here will highlight all assemblies in the assembly plot and the All Plates button will highlight all plates in the plate plot. The Unselect All button will unhighlight all objects in both the assembly and plate plots.
d. The <u>Preview Assembly</u> dropdown menu and related textareas are used in conjunction with the 3D plot to display your current changes. Selecting one of the scouted assemblies from the dropdown will display all MCNP cells that comprise it. When a change is made by the Apply Changes button, the textarea will update to reflect the change made. The <u>Assembly Tab</u> of the textarea shows every cell contained in the MCNP universe of the assembly number. The Fuel Lattice cell is also displayed here despite not being a direct member of the assembly universe. The <u>Plates Tab</u> of the textarea shows every cell used by the Fuel Lattice sorted by

its universe and therefore its plate type. Each plate even if used more than once in the lattice is only displayed here once.

e. The Assemblies label under the 3D plot displays the currently selected MCNP universe numbers of the assemblies highlighted in the plot. The Plates label under the 3D plot displays the currently selected MCNP universe numbers of the plates highlighted in the plot.
f. The Plate Type dropdown under the Sections label allows the user to select a new plate for the highlighted plates in each of the highlighted assemblies. The numbers displayed to the user as options are universe numbers corresponding to the scouted fuel plates in the supplied MCNP input file as the template.
g. Once at least one plate, at least one assembly, and a plate type has been selected, the user can click the Apply Changes button and find the Console displays the changes that were made as well as the textarea previews updated to reflect those changes.

4. **Legacy Assembly**

This page is the initial implementation of the Fuel Assembly page. It allows the user to modify existing plate composition through a series of dropdown selects rather than the 3D plot in the Fuel Assembly page. The Legacy Assembly page cannot change the plate layout within assemblies, it can only change materials of all plates, all sections within a plate, or an individual section. What it lacks in function it makes up for in generality: this page is functional with any MCNP input file whose fuel assemblies are built the same as in the NNS and NBSR file. This page should be phased out in favor of the Fuel Assembly page's 3D selection when it is capable of a dynamic 3D plot generation.

a. The Current Assembly dropdown displays the universe numbers of all Assembly objects that were read from the input file. Upon selection, the preview textareas update to display the currently selection Assembly's contents along with the Fuel Section and Fuel Lattice input boxes.
b. The Plate dropdown displays the universe numbers of the fuel plates found in the fill parameter of the current Assembly's Fuel Lattice Cell. The option "All Plates" can also be selected from this menu, in which case all sections of all plates in the fuel assembly will receive the material change from the edit.
c. The Plate Edit button is used in conjunction with the Plate dropdown. This button opens a Dash Modal menu that offers the user to select one or all of the sections in the current plate selected from the Plate dropdown. It displays the current material of the selected section and then the user can set the selected sections to the new material input by hitting the Apply Changes button.
d. The Material dropdown displays all of the materials found in the input file to be used in conjunction with the Change All Fuel button. Once a material is selected and the button pressed, all sections of all fuel plates in all assemblies will be changed to the selected material. These changes can be viewed with the preview textarea and updates upon changes made.

5. **Card Views**

The Card Views dropdown tab on the navigation bar shows the user more pages that can be used to troubleshoot their input file. With the exception of the Universes page, these pages display a single MCNP card at a time with a similar layout.

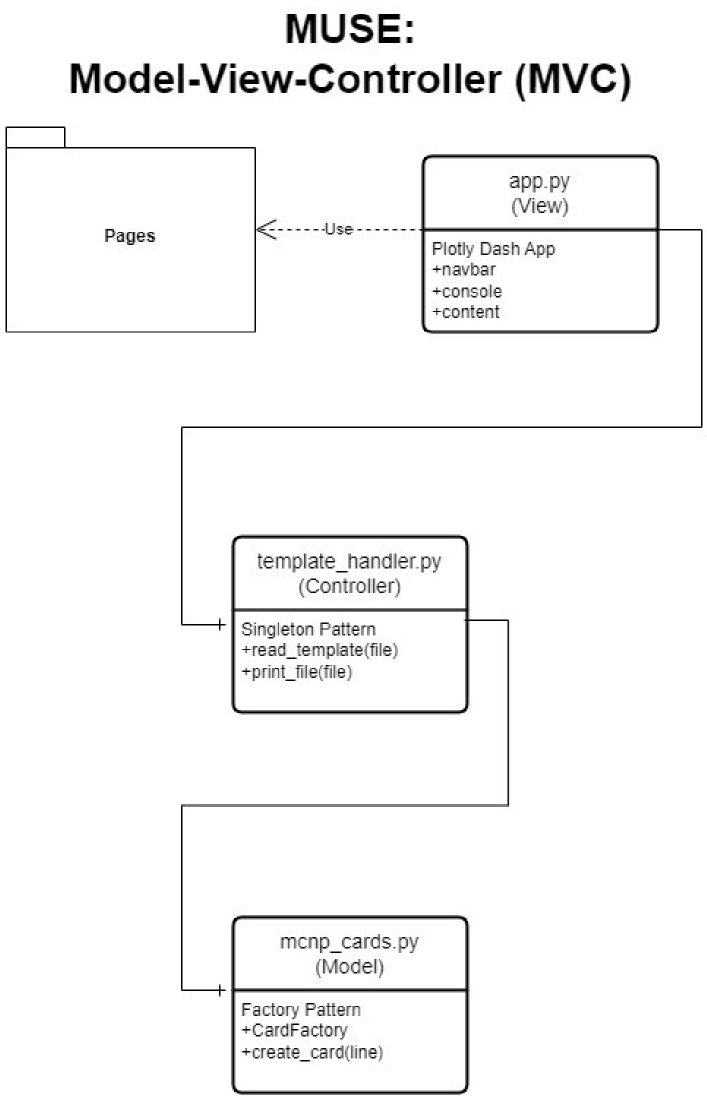
a. The Current Card dropdown displays the numbers of each relevant MCNP card. Upon selection, the Print Preview will update to reflect how the current card will be printed in if the Print File button is clicked. The input boxes below the dropdown off the user a way to modify the relevant parameters of the card. Once a change has been made, the Apply Changes button can be clicked and a message appearing in the Console will inform the user of the status of the action and if a change was made, the Print Preview will update to reflect the change.
b. For the Cell Cards page, the Related Cells textarea tab shows the origin cell if the current cell is a LikeCell or shows the children cells if the current cell is a RegularCell or VoidCell.
c. The Universes page can be helpful for debugging universe contents. The Current Universe dropdown displays the universe numbers of all found universes in the input file. Upon selection, the Universe Contents textarea tab will display all MCNP cells that are grouped in that universe. The Filled Cells textarea tab will display all MCNP cells that use the current universe as a part of their fill parameter.

Code Architecture

The code follows a few architectural and design patterns. As mentioned in the Prerequisites, the app follows an MVC architecture for its UI. The files are grouped into directories called "models", "pages", and "controllers" to make this evident. A diagram of the project architecture can be found at the bottom of this section.

1. The Model is the "mcnp_cards.py" file that contains class definitions for the CardFactory, Assembly, and Card objects. These definitions control the translation of a single line of MCNP code into a Python object, storage of its data, and string representation called when printed. The CardFactory class, as the name suggests, is implemented with a Factory design pattern. The CardFactory class is instatiated by the controller at runtime and "create_cards" method called for each line found in the input file. The CardFactory decides how to filter the line through various regular expressions and returns a Card of the relevant type. More on the contents of each class can be found in the Class Definitions section.
2. The Views are found in the "pages" directory and control the page layout displayed to the user as well as logic for user requests. The page layouts are called by app.py as the url is changed and isolated to the content portion of the page displayed to the user to not overlap with the navigation bar or console. It is important to note that this is not a strict adherence to the MVC framework as user requests should be handled by the Controller. However, due to the standard for Dash layout-callback interaction it was best to couple these components in the same file. A more strict MVC structure can be applied by moving every page callback to a new file in the "controllers" directory and adding relevant imports.
3. The Controller is the "template_handler.py" file found in the "controllers" directory and contains the class definition for the Template_Handler object which follows a Singleton design

pattern. As such, the "controllers" directory also contains a "template_handler_instance.py" file that instantiates the single instance of the Template_Handler and is used to circumvent circular import statements.



Class Definitions

Since the MCNP cards that are manipulated represent an abstract object, the Object-Oriented Programming paradigm is inherently useful for this project's data representation. The classes that will be described here are found in the "template_handler.py" and "mcnp_cards.py" files. A UML diagram of the "mcnp_cards.py" file and its class definitions can be found at the bottom of this section.

1. **Template_Handler Class**

This class as mentioned in the Code Architecture section follows a Singleton design pattern. The Template_Handler object contains the default printing filepath, all data structures used to store information read from the input file, and various regular expessions to identify parameters of cell cards. Data structures consist of arrays for the raw lines read from the input file and dictionaries for the instantiated representations as Card objects. The methods of the Template_Handler class are described below.

a. The <u>read_template</u> function is called by "app.py" at runtime and passed the filepath of the input file to be parsed. This function has multiple helper methods that organize the structure of reading and cleaning data. These functions will be described in detail below.
b. The <u>clean_comments</u> function is a helper method for read_template. It performs a reverse order search over the given array of raw lines to find comments around cell, surface, or material cards. It assigns a comment to a card number if used as a '$' comment on the same line as the card number or a 'cC' comment the line before, prioritizing '$' comments over 'cC'. The resulting arrays are not returned but rather modified directly as an attribute of the Template_Handler class.
c. The <u>join_card_pieces</u> function appends the cleaned lines read from the input file to eachother if the lines follow a line continuation rule followed by MCNP: if a line ends with "&" or begins with five consecutive leading white space characters. The _recurse_continue function is a helper method for this operation and is a recursive function to determine if the line continues by these rules. The resulting arrays are not returned but rather modified directly as an attribute of the Template_Handler class.
d. The <u>make_cards</u> function is called on each of the arrays of cleaned MCNP lines from the input file. This method instantiates the CardFactory class and calls its create_card method to create a Card object from the cleaned input file line. The created card is then sorted by its type and added to the relevant dictionary keyed by its unique number.
e. The <u>set_like_cell</u> function is used to update the attributes of LikeCell objects created by the make_cards function. The operation is only performed on LikeCell objects and finds the origin cell then determines the attributes to be set with respect to its origin and changes field.
f. The <u>dissect_like_param</u> function is a helper method for the set_like_cell function and is used both in creating the LikeCell objects and modifying its attributes later. This function interprets the changes field of the LikeCell with respect to the origin cell attributes and updates attributes to remain consistent with MCNP syntax.
g. The <u>make_assemblies</u> function creates the Assembly objects for the user to interact with. It searches through each Cell object found in the input file and adds the Cell object to the all_fuel_plates dictionary keyed by its universe if any of the zaid statements in the Cell's material definition contains uranium. This is the origin of the program's name as a "uranium scout". The identified Cells are then organized into plates by universe association and then plates assembled into Assemblies by universe association.
h. The <u>apply_comments</u> function is the final step in the read_template function. This method searches through the comments identified in the clean_comments function and sets the comment attribute of a Card object if it was identified.

i. The print_file function is called when the user clicks on the Print File button on the console of the app. This controls the printing order and format of each card created by the program. It uses the print_dict_list and print_cards functions as helper methods to print certain data structures correctly. The style of each printed card is not controlled here but rather at the __str__ method found in the class definition of the Card object.

2. CardFactory Class

This class is responsible for creating Card objects from a string containing an MCNP line. The regular expressions for sorting each line to its respective card are precompiled and stored as attributes here to be accessed in the create_cards function.

3. Card Class

The Card Class is the parent class for all subsequent Card subclasses. This was done mainly for input validation but offers a way to define behavior for all Card subclasses, namely, comment interaction. The set_comment method is used in the apply_comments method in the Template_Handler to set the comment attribute of a Card and get_inline_comment method is used to produce an inline comment during a __str__ call. Each subclass of the Card class has unique constructors and __str__ methods but follow the same general format: the constructors used broken down regular expressions to extract unique parameters and store them as attributes to be accessed by the __str__.

4. Cell Class

The Cell class is the parent class for the RegularCell, VoidCell, and LikeCell classes. These subclasses make a super call Cell's __init__ through which the universe number and fill uses are separated from the param section of the Cell and added to Template_Handler's data structures. The string representation that is called when an object is printed is controlled at the subclasses' __str__ level and varies greatly between the three types of Cells.

5. Surface Class

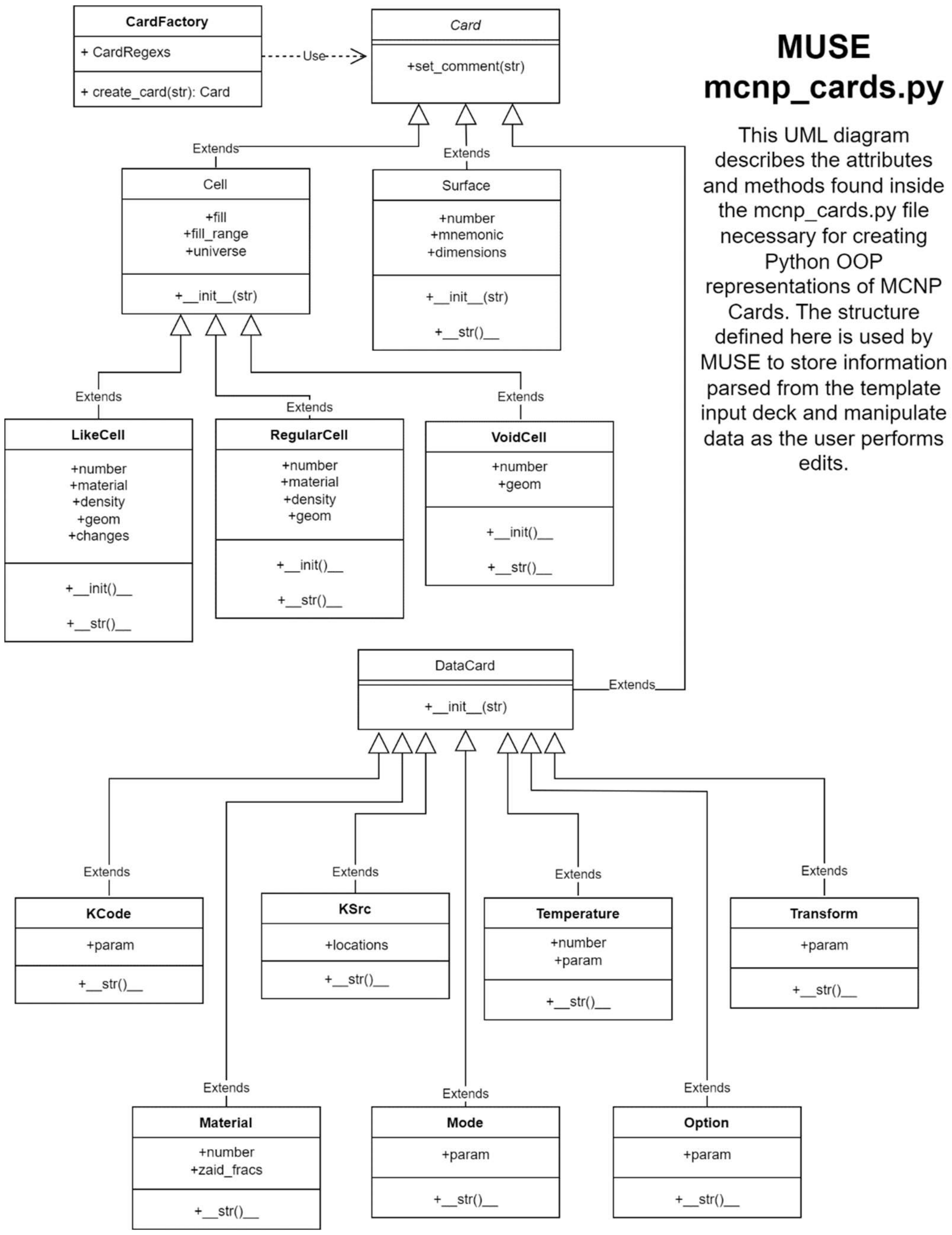
This class creates attributes and string representation of MCNP surface cards. The definition is relatively straight forward and follows the general format described in the Card Class section.

6. DataCard Class

This class inherets from Card and has many subclasses referring to the MCNP Data Card section of the input file. The important one to note is the Material class which is used extensively. Materials have a number and an array attribute that contains the zaid-fraction pairs of the material. When the __str__ is called for a Material, each zaid-fraction pair is printed on a new line with a comment attached to the end identifying the isotope for readability. This feature can be turned off on the app if the Element Comments checkbox is unmarked. The zaid_to_isotope function accomplishes this by accessing the element() function from the mendeleev library unless the result has already been accessed in the element_symbols dictionary working as a literal cache.

7. Assembly Class

This class is instantiated after all of the others during the make_assemblies function in the Template_Handler. The Assembly class holds useful information about an assembly found in the input file such as its universe number, fuel section Cell, fuel lattice Cell, and an array of the other components found in its universe. The Assembly class is not printed to the output file and its __str__ is only used for debugging purposes.



MUSE
mcnp_cards.py

This UML diagram describes the attributes and methods found inside the mcnp_cards.py file necessary for creating Python OOP representations of MCNP Cards. The structure defined here is used by MUSE to store information parsed from the template input deck and manipulate data as the user performs edits.

Development Recommendations

The project has followed the PEP-8 Style Guide for Python in regards to comments, pythonic style, spacing, organization, and naming conventions. The project has used a GitLab repository for version control that is linked in the Getting Started section of this report. The commit history can be found there from the initialization of the project. It is recommended that these aspects remain intact throughout futher development although change is reasonably expected.

There are many directions development can work towards given the solid base of MCNP generality in translating MCNP input file text to OOP representation. Below is a list of improvements and new features that can expand the project's scope and capabilities as well as solution methods given the current design.

1. Plate Creator

The Plate Maker page is well suited to edit a fuel plate consisting of numerous section Cell cards. However, it is only capable of editing existing plates found in the input file and cannot create a new plate. This is a reasonable request because swapping out new plates in the assembly is its main use and by modifying an existing plate rather than a new plate all together, there is a great risk of modifying an unintended plate elsewhere in the geometry.

To accomplish this, a section needs to be added to the page allowing the user to create a new Fuel Plate. This new fuel plate would be instantiated as 30 LikeCells with origins set to the model fuel plate in the input file and universe set to one not already in the Template_Handler's all_universes dictionary. The LikeCells should then be appended to a list and added to the Template_Handler's all_fuel_plates dictionary keyed by the new unique universe number.

2. Dynamic 3D Plots

A prominent limitation to the current app is its speciality with the NNS input file. As described in the Legacy Assembly Page section of this report, the 3D plot generated in the Fuel Assembly page is hard-coded in the "select_graphs.py" file to create the relevant selection plots. However, functionality exists for the program to modify materials and fuel plates of the simulation as demonstrated in the Legacy Assembly Page.

To accomplish this, the 3D plots should be generated in a function called at the end of the read_template function in the Template_Handler. This function should recognize the number of Assemblies created earlier and generate a new grid based on this. The order of selected assemblies must also be modified as the current implementation relies on the static fuel management scheme of the NNS design found in the "select_graphs.py" file. The number of plates generated in the plot can be found by looking at the fuel_lattice.fill attribute of each Assembly and generating a plate for each entry. The section plot can be generated based on the number of Cell cards in the all_fuel_plates[plate_u] list. The highlight_selected callbacks for each 3D plot will need to be modified to select Cells by their geometric position in the input file. With the current implementation, the selected Cell is found by increasing order of its card number in the list and may not function correctly for all input files.

3. Undo Option

A standard feature on many editing tools allows the user to undo their last change. This would certainly be useful for this project with increasingly complex changes being made to the input file. A mistake such as replacing all Cell materials at once can erase a lot of work easily and can be avoided.

To accomplish this, the report of Console messages should be modified. Rather than simply a message informing the user of the success of their change or not, more detailed messages need to be installed. Upon pressing the Undo button, the code should find the last message printed to the console and using this information revert the change to the last state of the Template_Handler's data structures.

4. File Presets

An important feature for making this project generalizable is file preset selection. Rather than having the template input file read from a hard-coded filename, the project can access several preset input files stored in the mcnp_templates directory. The Home page should then be a preset selector page that displays the stored presets or new upload and one must be selected before proceeding to the editorial pages.

To accomplish this, the call to template.read_template in "app.py" must be modified to accept input from the Home page. If the user decides to select a different preset, a new function needs to be made to reset the fields of the Template_Handler class and start anew.

5. Webhosting

A limitation to the project's accessibility is its implementation as a locally hosted app. This would be simple to solve with a NCNR hosted server that runs the program as a new session per user and therefore navigated to at any time. This would then be best to refactor the data structures in the Template_Handler to a database rather than arrays and dictionaries to be stored between sessions and eat less server memory.

6. Surface Scaling

Geometry manipulation is a feature that was considered in this project's development. The idea is to modify the core's geometry by adding new fuel assemblies or scaling the size of the fuel assemblies with a button or slider. This requires representation of the assemblies and surfaces which MUSE can handle and an understanding of surface coordinates which MUSE does not have logic for but has access to the necessary information. A large amount of work is necessary to make this feature a reality but its usefulness is undeniable.