

# The HUNTER Dynamic Human Reliability Analysis Tool: Graphical User Interface

Jeeyea Ahn,<sup>a\*</sup> Ronald Boring,<sup>b\*</sup> Thomas Ulrich,<sup>b</sup> Jooyoung Park,<sup>b</sup> and Yunyeong Heo<sup>a</sup>

<sup>a</sup>Department of Nuclear Engineering, Ulsan National Institute of Science and Technology (UNIST),  
Ulsan, Republic of Korea

<sup>b</sup>Human Factors and Reliability Department, Idaho National Laboratory,  
Idaho Falls, Idaho, USA

\*Contacts: jeeyea@unist.ac.kr; ronald.boring@inl.gov

---

**Abstract:** The initial version of the Human Unimodel for Nuclear Technology to Enhance Reliability (HUNTER) simulation software implementation was a Python command line application. The model defining the scenario was manually constructed within input decks that were not user friendly. For the current version of HUNTER, a graphical user interface (GUI) was designed and developed to ease the model creation, editing, and simulation execution. The GUI supports creation and editing of the overall model and sub-model elements grouped under the three main simulation modules (i.e., Individual, Environment, Task modules). The GUI provides distinct interface sections with visual representations for each of these simulation modules. Visual representations of model objects and their visual arrangement within the modules clearly convey the underlying structure of the simulation code and ease model creation and editing. For example, what was input through the command line or a text editor in the past is now implemented as an intuitive and visualized interaction such as clicking a button or turning a toggle on and off so that analysts can focus on creating the model without having to contend with esoteric text files and their detailed syntax requirements. While the GUI presented in this paper is specific to HUNTER, the interface design concepts are generalizable to benefit other dynamic modeling environments.

---

## 1. INTRODUCTION

The Human Unimodel for Nuclear Technology to Enhance Reliability (HUNTER) framework was developed to support dynamic human reliability analysis (HRA) [1-3]. HUNTER is designed to create a virtual operator model that interfaces with a virtual plant model. The purpose of the software is to model variable event progressions, which allow determination of frequencies of particular types of human errors but also exploration of what-if scenarios that would otherwise be difficult to model with traditional static HRA techniques.

One barrier to the adoption of dynamic tools is the complexity of using them. The original HUNTER software [1] was command-line based, and the information input and output was text-based through a terminal window. In addition, each module was written as a separate application, such that the parts of HUNTER were not fully integrated. This required running independent codes for the segmented functions, providing input manually, and manipulating the modules via command line.

With the recent advent of a standalone integrated software version of HUNTER [3-4], a major goal is ease of use. To simplify the analysis process, it is necessary to implement software that integrates the fragmented code into one wrapper and provides a user-friendly interface. To that end, we created a graphical user interface (GUI) for HUNTER to ease modeling functions. This GUI also links previously disparate pieces of code into a single standalone software framework.

The essential functions included in the HUNTER GUI include:

1. Ability to create new models
2. Ability to load existing models
3. Ability to edit models according to HUNTER functions such as performance shaping factors (PSFs)
4. Ability to save models
5. Ability to run the models with different modeling configurations
6. Ability to view the results of model runs
7. Ability to save and export model runs.

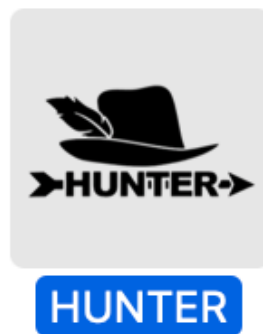
HUNTER's interface functionality is expected to evolve and mature with each successive version of the HUNTER software code. The current version captures the above functionality but does not implement a GUI for each different element. Elements 1–4, which correspond to model setup, feature a convenient GUI in which all functions can be performed using simple mouse clicks and inputs. Elements 5–7, which correspond to model runs, still largely make use of the command-line structure native to the underlying Python development environment. As additional use cases are identified and new models are created in HUNTER, it is expected that additional GUI functionality will be developed to support Elements 5–7.

This article uses example screenshots to demonstrate and explain the GUI. Because HUNTER was developed in Python, it is cross compatible with different operating systems.\* The screens depicted in this paper were captured from HUNTER as it is being operated using Microsoft Windows.

## 2 STARTING PAGES

### 2.1 Splash Screen

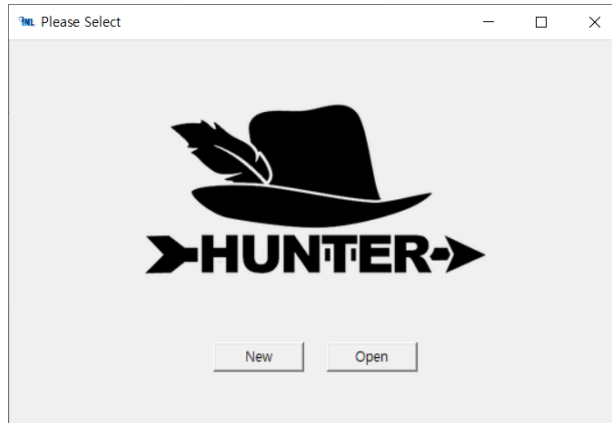
After the HUNTER software package is installed, the HUNTER application is started either by inputting `execute HUNTER.py` into the command line in Python, or by double-clicking on the HUNTER icon (see Figure 1). After launching the software, a splash screen pops up (see Figure 2) and remains until the HUNTER application is closed by clicking the appropriate close-window button in the titlebar, initiating a new HUNTER model, or opening an existing one.



**Figure 1: HUNTER Desktop Icon**

---

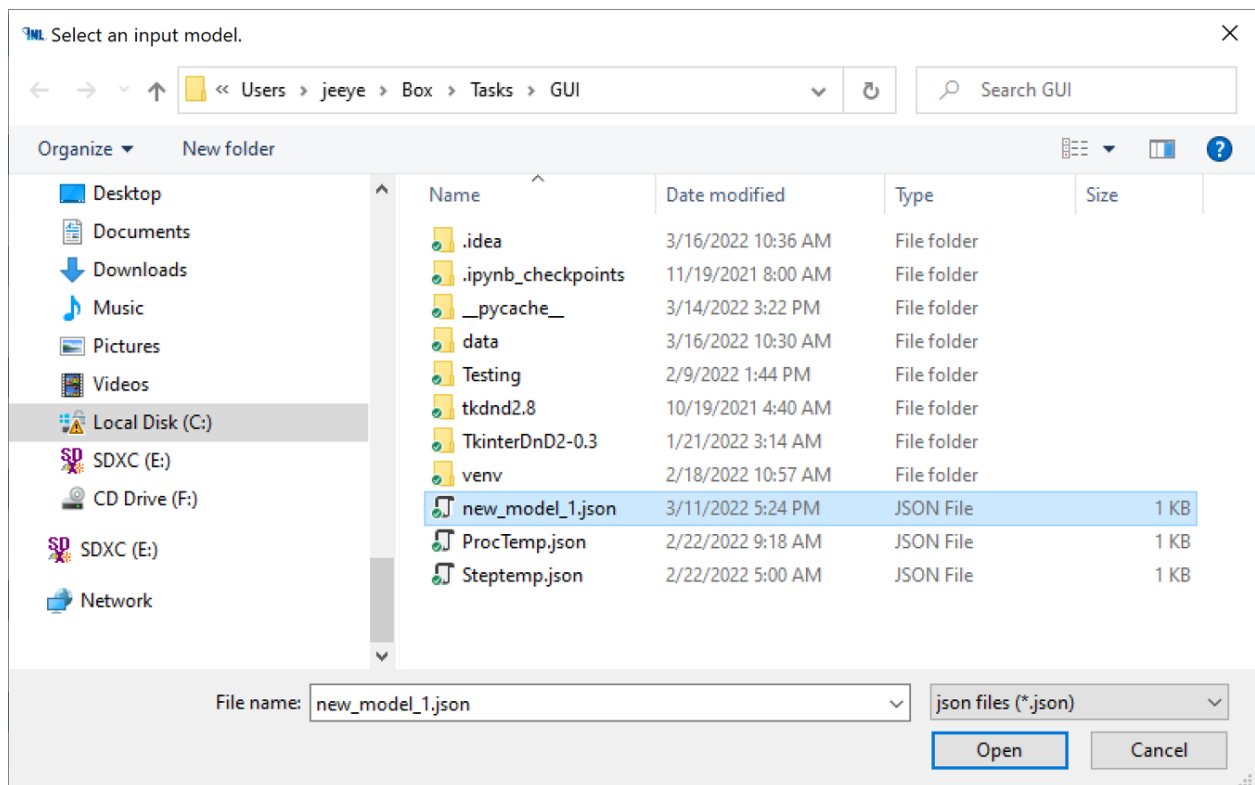
\* A limiting factor regarding HUNTER's cross-platform compatibility is the operating system used for the linked plant simulation code, which is currently Microsoft Windows.



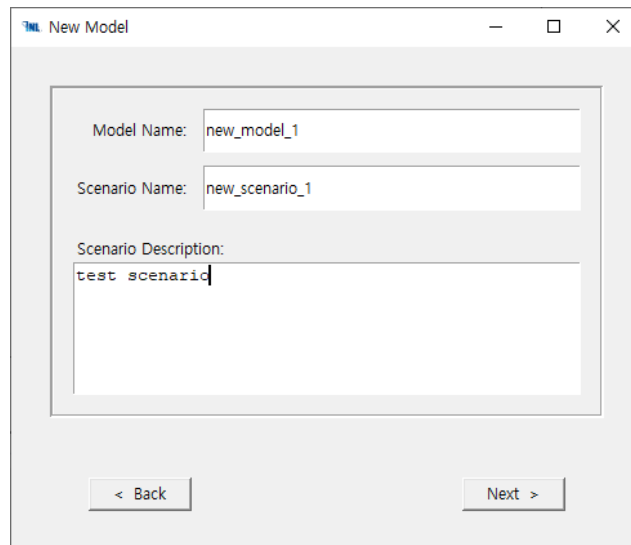
**Figure 2: HUNTER Splash Screen**

### 3. NEW OR EXISTING MODELS

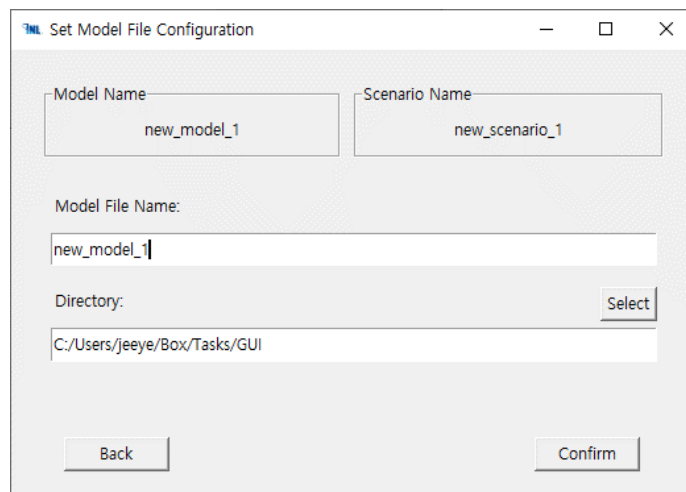
HUNTER scenarios are grouped into models, which link all the subsets files (e.g., procedures). If the user chooses to open an existing model, a dialog box appears, as shown in Figure 3, enabling the user to select a previously created model file. Models are stored as JavaScript Object Notation (JSON) files.



**Figure 3: HUNTER File Dialog for Loading an Existing Model**

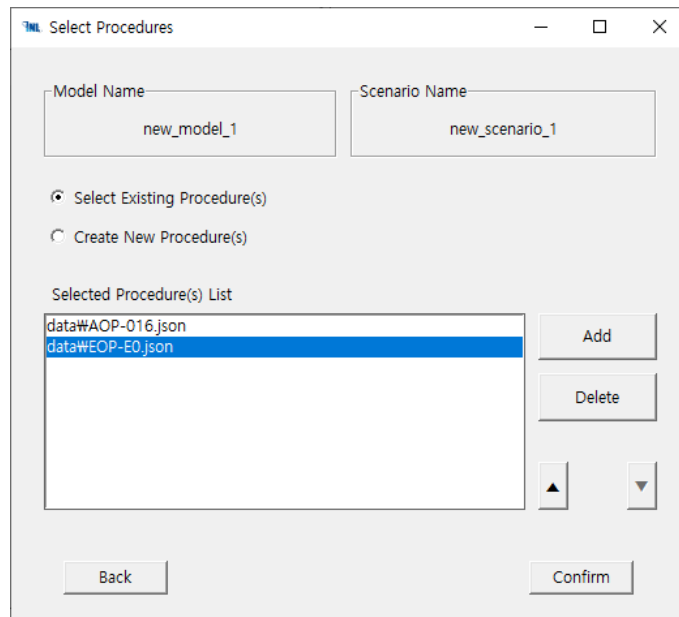


**Figure 4: HUNTER Screen for Creating a New Model**

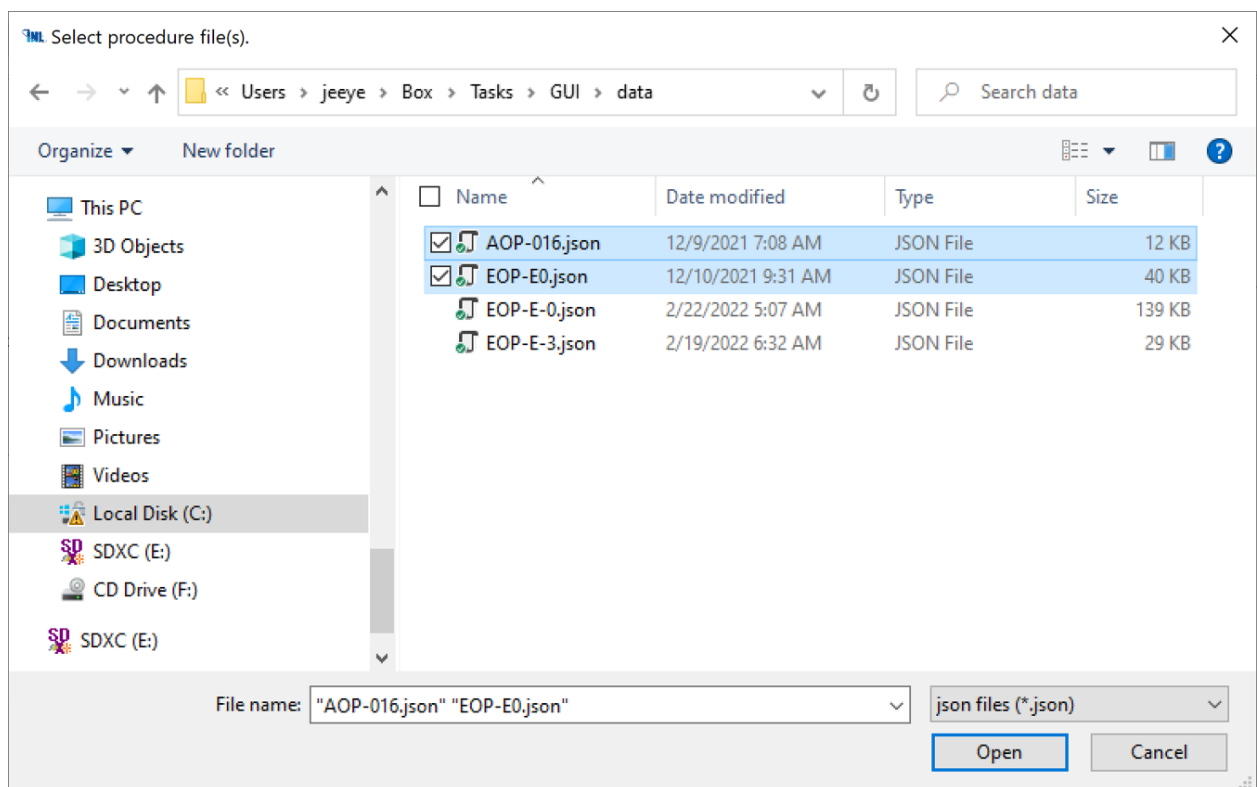


**Figure 5: HUNTER Dialog Window for Specifying a Model File Location**

If the user chooses to create a model, a window for entering the model name, scenario name, and description appears, as shown in Figure 4. A follow-on screen (see Figure 5) allows the user to specify the location of the model files so they can be saved for later reuse. After entering this required information and clicking on the Next button, the following window (see Figure 6) allows the user to add or delete existing procedures from the model, or to create new models. Figure 7 shows the dialog box for selecting procedure files to add to the model. Procedure files are also stored as JSON files. Multiple procedure files can be selected at once, and the order of the selected procedure files can be changed (using the arrow buttons) or deleted (using the Delete button).



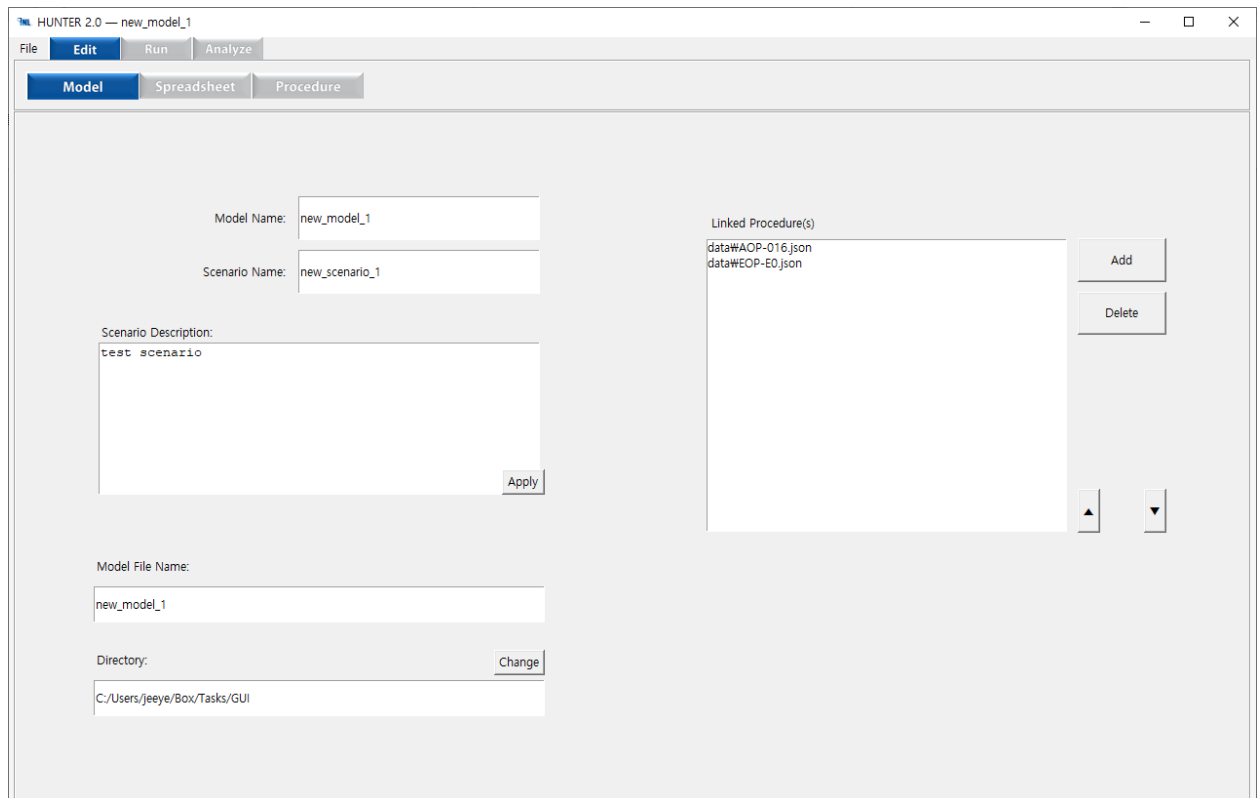
**Figure 6: HUNTER Dialog Window for Adding, Creating, or Deleting Existing Procedures from the Model**



**Figure 7: HUNTER File Dialog Window for Adding Procedures**

If the user opts to manually create a new procedure, they are prompted to name the procedure and are then taken to the procedure window (described in the next section), where they can build the new procedure in step-by-step fashion.






## 4. MAIN EDIT WINDOW



**Figure 8: HUNTER Model Window View**







HUNTER has three main modes: Edit, Run, and Analyze, as denoted by selectable items in the menu bar (see Figure 8). The currently displayed item is highlighted with a blue background. Currently unavailable options have a dark gray background. A fourth option (i.e., the File menu) is always available and allows the user to load, save, and print models. Edit mode allows the user to define and edit different elements of the HUNTER model, such that the end user should be able to create complete HUNTER models from within these windows. Within Edit mode are three windows: Model, Spreadsheet, and Procedure. These are represented as buttons (the Model window is shown in Figure 8). When the initial setup is finished, the model settings are imported to the Model window, where the user can see and change them. The Spreadsheet window (see Figure 9) gives an overall view that can be used to check and change all the input values for HUNTER analysis at once. The Spreadsheet window gives the user a full view of the entire HUNTER taxonomy and is recommended only for power users. Lastly, there is the Procedure window, as seen in Figure 10. Because HUNTER is built around operating procedures at nuclear power plants [4], this view contains much of the essential functionality needed to drive HUNTER.




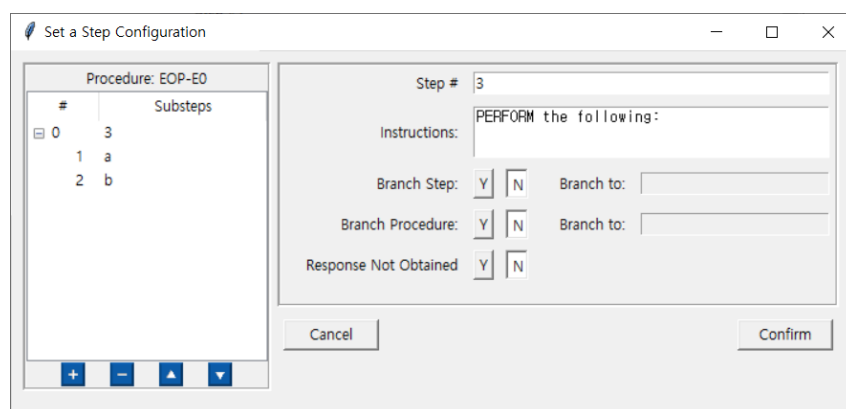
Symbol	Function
	Move Task Up
	Move Task Down
	Add Task
	Delete Task
	Edit Task

**Figure 11: HUNTER Procedure Editing Tools**

As can be seen in Figure 10, the Procedure window visualizes, in procedural step units, the procedure relevant to the model's scenario, and offers an interface that helps to easily define procedure step attributes. The Procedure window consists of the graphical procedure model area (on the left) and the input setting panel (on the right). The graphical area on the left includes a tab for each procedure, the name of which is displayed at the top. Each tab employs a flowchart-like format to visualize all the steps of the procedure. If the user needs to change the contents or the order of the steps, they can edit the steps via the button panel at the top left (see Figure 11 for a legend).

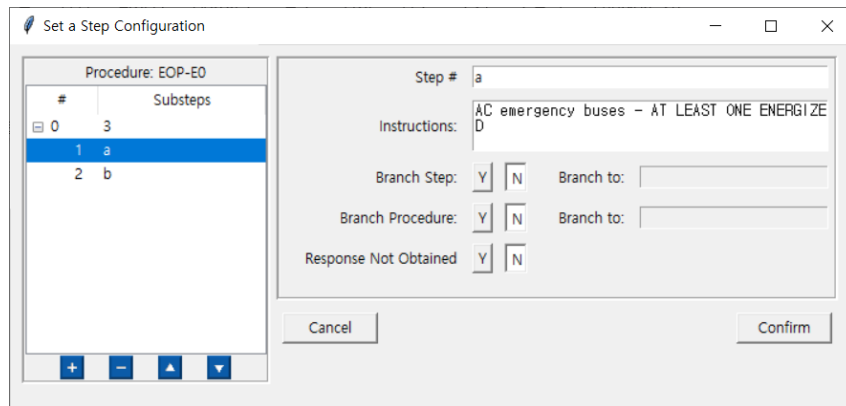
Each step box in this diagram can be clicked on to highlight the selected step. Information on the selected step is displayed in the panel on the right. The user can select a step box and then press the  or  button on the left to move the step up or down in the step order. To change the contents of the step, the user can use the pop-up window that appears by pressing the  button. The user can also select a step and then press the  button to delete it. However, due to step linkages, the user can only delete Response Not Obtained (RNO) steps via the window that pops up by pressing the edit button () and not by pressing the delete button ().

A user can add a new step by pressing the  button. A step configuration pop-up window appears, and the settings inputted to that window are used to create a new step. Each newly created step is, by default, positioned last in the procedure, but the order can be changed by using the step move buttons.

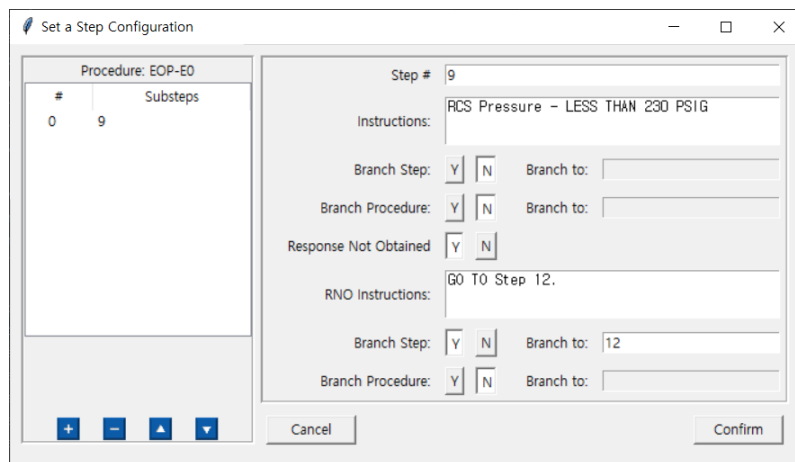


**Figure 12: HUNTER Procedure Step Configuration Editor: Normal Step**





**Figure 13: HUNTER Procedure Step Configuration Editor: Substep**



**Figure 14: HUNTER Procedure Step Configuration Editor: Response Not Obtained**

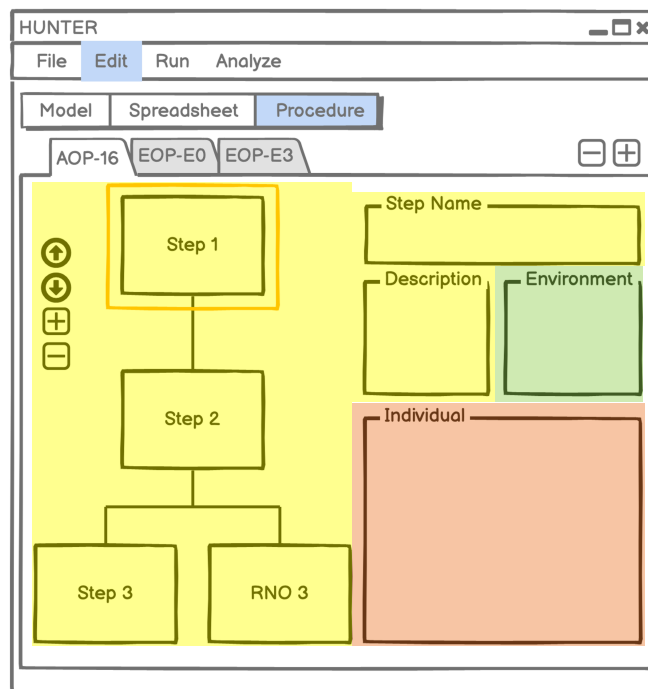
The window in Figure 12 pops up when the edit or add buttons are clicked. The user can add a substep by clicking the **+** button at the bottom of the window (see Figure 13). To make a substep of a substep (i.e., a nested substep), the user selects the parent substep from the left-hand list box and presses **+**. To delete a specific substep, the user selects the target substep in the left-hand list box and presses **-**. The user can operate the up or down buttons to change the substep order. Information on the selected step/substep is displayed on the right. When the edit button is pressed, information on the selected step is displayed. If the corresponding step is RNO (see Figure 14), the user can indent the Y for “Yes” button of RNO so that a space for adding RNO information appears. To delete an existing RNO step, the user clears the RNO option via this step configuration setting window.

The Procedure window (see Figure 10) contains information relevant to all aspects of HUNTER. Figure 15 shows a wireframe version of the Procedure window, in which the various types of HUNTER information are shown in different colors depicting the different modules. As noted, HUNTER has three main modules:

- The Task module contains information about procedure steps
- The Environment module links to external code like plant simulation
- The Individual module models aspects like PSFs that affect how the virtual operator performs tasks

Yellow areas correspond to information used for the Task module. The green area information is used for the Environment module and contains definitions of key plant parameters employed by the steps in order to monitor and control the plant simulation, when applicable. Finally, the red area information is used by the Individual module to set the PSF fields. Note that each of these areas is linked to a specific

procedure step or substep. Currently, there is no ready way of using the GUI functionality to set global parameters (e.g., shared PSF assignments) for all steps in the procedure. That functionality is best achieved by using the Spreadsheet window.



**Figure 15: Wireframe Rendering of the HUNTER Procedure Window, with Color-Coded Sections Depicting Task (Yellow), Environment (Green), and Individual (Red) Module Information**

## 5. CONCLUSIONS

Dynamic HRA—and, by extension, HUNTER—will succeed as risk tools only if they provide true benefits to the risk analysts who use them. Dynamic HRA offers the potential to provide deeper modeling fidelity; opportunities for exploring the ranges of human performance; the ability to extrapolate HRA to new scenarios, technologies, and domains; and the prospect to model output types beyond human error probabilities [2]. However, dynamic HRA does not accomplish these advantages over static HRA without costs. Dynamic HRA can be considerably more complex to set up and model. As such, HUNTER strives to strike a balance by creating a uniquely simple and adaptable software tool that may be readily used by risk analysts to model phenomena of interest. With the advent of a standalone software package under a single GUI, HUNTER promises to make dynamic modeling in HRA considerably easier for analysts. This is an ongoing task, which the development team (see Figure 16) is undertaking over multiple years. First steps toward greater simplicity include expanding the GUI beyond the command line interface for running and analyzing events. Beyond that, additional work is being undertaken to automate modeling from procedures and integrate with other risk modeling tools.



**Figure 16: The HUNTER Development Team (Right to Left—Jooyoung Park, Thomas Ulrich, Jeeyea Ahn, Ronald Boring, and Yunyeong Heo)**

### **Acknowledgements**

This work of authorship was prepared as an account of work sponsored by Idaho National Laboratory (under Contract DE- AC07-05ID14517), an agency of the U.S. Government. Neither the U.S. Government, nor any agency thereof, nor any of their employees makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

### **References**

- [1] Boring, R., Mandelli, D., Rasmussen, M., Herberger, S., Ulrich, T., Groth, K., & Smith, C. (2016). *Integration of Human Reliability Analysis Models into the Simulation-Based Framework for the Risk-Informed Safety Margin Characterization Toolkit*, INL/EXT-16-39015. Idaho Falls: Idaho National Laboratory.
- [2] Boring, R., Ulrich, T., Ahn, J., Heo, J., & Park, J. (2022). *Software Implementation and Demonstration of the Human Unimodel for Nuclear Technology to Enhance Reliability (HUNTER)*, INL/RPT-22-66564. Idaho Falls: Idaho National Laboratory.
- [3] Boring, R., Ulrich, T., Park, J., Heo, Y., & Ahn, J. (2022). “The HUNTER Dynamic Human Reliability Analysis Tool: Overview of the Enhanced Framework for Modeling Human Digital Twins.” Proceedings of the Probabilistic Safety Assessment and Management (PSAM 16) Conference.
- [4] Ulrich, T., Boring, R., Park, J., Heo, Y., & Ahn, J. (2022). “The HUNTER Dynamic Human Reliability Analysis Tool: Procedurally Driven Operator Simulation.” Proceedings of the Probabilistic Safety Assessment and Management (PSAM 16) Conference.