UVSim Software Simulator

PREPARED FOR

Kevin Burtt

Utah Valley University

PREPARED BY

Taj Poulsen, Oliver Britos, Preston Little, Tyler Hilbig

CS 2450

Table of Contents

1. Executive Summary

UVSim is an innovative educational project aimed at revolutionizing the learning experience for computer science students, particularly in machine language and computer architecture. Developed as a software simulator, UVSim offers students a hands-on environment in which to experiment with executing BasicML programs on a virtual machine. The project aims to bridge the gap between theoretical knowledge and practical understanding by providing a platform that facilitates active learning and experimentation.

The core content of UVSim revolves around its virtual machine environment, which operates on signed four-digit decimal numbers and features a 100-word memory capacity. Through an intuitive graphical user interface (GUI), students can interact with the program, manage multiple simulators, load and edit program files, and customize the interface with predefined or custom themes. UVSim's significance lies in its ability to empower students to deepen their understanding of low-level programming concepts through practical experimentation and simulation, ultimately preparing them for advanced studies and real-world applications in computer science.

2. Application Instructions

Description

- UVSim is a simple virtual machine designed to help computer science students learn about machine language and computer architecture. This software simulator allows students to execute machine language programs written in BasicML on the UVSim. The UVSim works with words, which are signed four-digit decimal numbers (e.g., +1234, -5678). It has a 100-word memory, and these words are referenced by location numbers 00, 01, ..., 99.

Prerequisites

- Download the latest version of Python from https://www.python.org/downloads/ with your corresponding OS and run the installer.

- Download any files you would like to use with the UVSim program.

Installation

- Download the UVSim project file. This can be done by clicking "<>code", "Download ZIP", and then unzipping the file.

How to Start the Program

1. Ensure all the required modules (Standard Python Library) are accessible in the Python environment where you intend to run this program.
2. Execute the Python file using a Python interpreter. To do this, run python main.py in your terminal or command prompt.
3. GUI Window: After running the command, you will be taken to the UVSim GUI application interface.
4. Interacting with the GUI: You can interact with the program within the GUI window by clicking the provided buttons.

Running the Program and Simulators

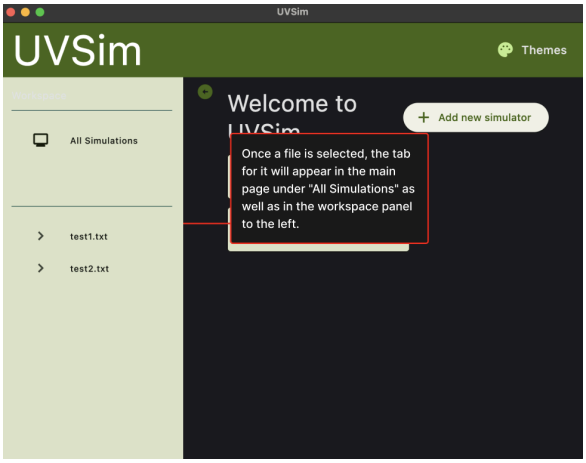
The UVSim Emulator allows you to execute programs and simulate computer operations by doing the following:

1. Add a simulator to the screen by creating a new simulator. (See Adding Simulators section)
2. Click on the tab created by the "Add new simulator" button in step 1.
3. Your selected file will appear in the text editor and simultaneously be loaded into the program (you can click the "Save and Load File" button to check that the file is loaded into the program. By doing so, you will see "File read successfully" in your console).
4. Click on the "Run" button to execute the program.
5. Monitor the output in the output console.
6. Use the "Stop" button to halt program execution.

Adding Simulators

1. To begin adding simulators, click the "Add new simulator" button. This will open a tab in which you can select files to test.

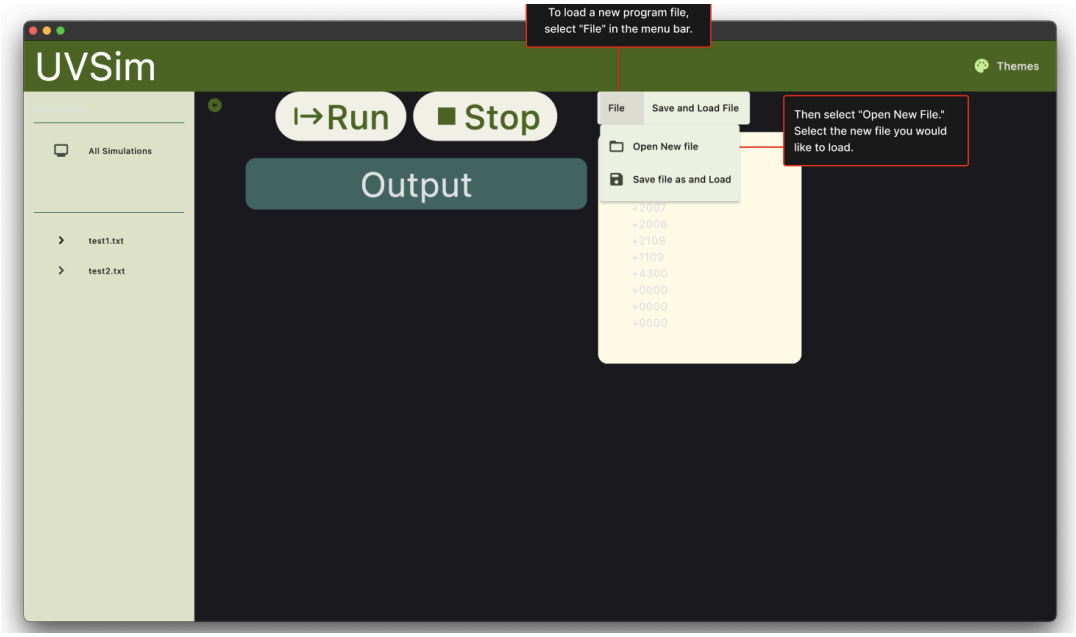
2. Once a file is selected, its tab will appear on the main page under "All Simulations" and in the workspace panel to the left.



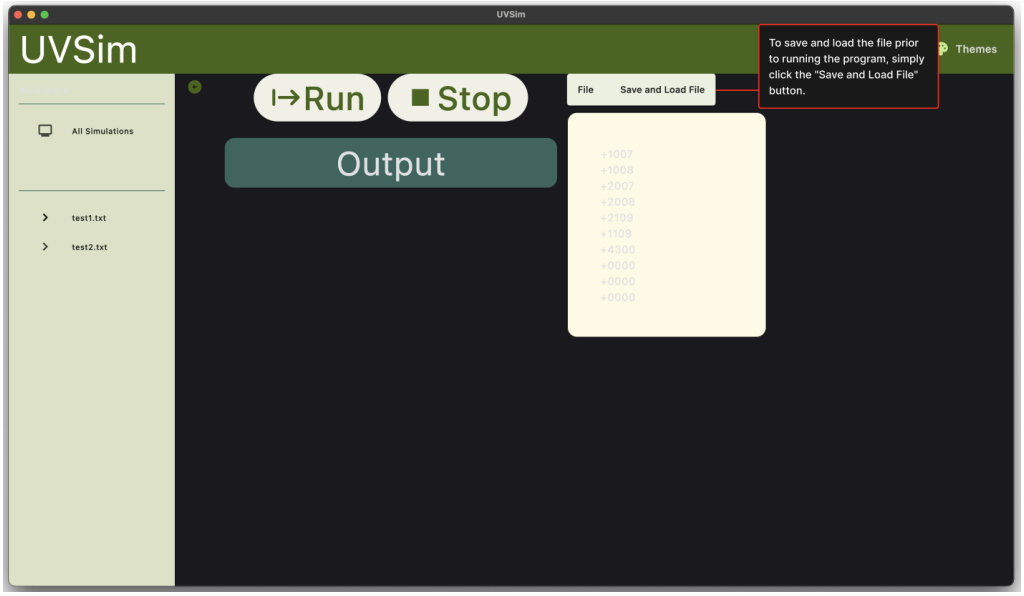
3. You can repeat steps 1 and 2 to create multiple simulations.

File Management

1. Select "File" in the menu bar to load a new program file and then "Open New File." Select the new file you would like to load.



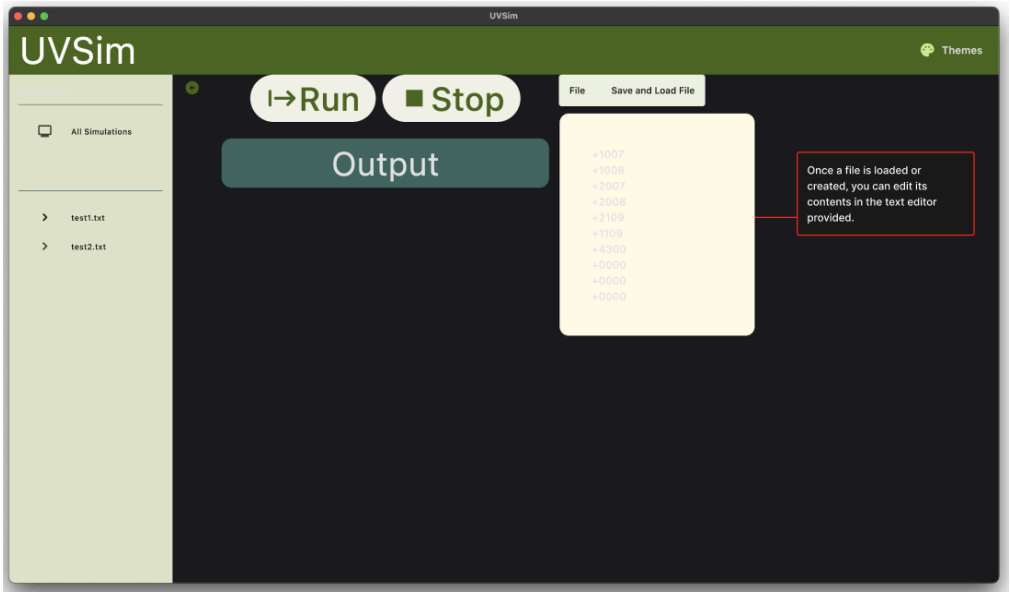
2. To save and load the file before running the program, click the "Save and Load File" button.



Editing Selected Files

Once a file is loaded or created, you can edit its contents in the text editor provided by doing the following:

1. Make your desired changes to the code.



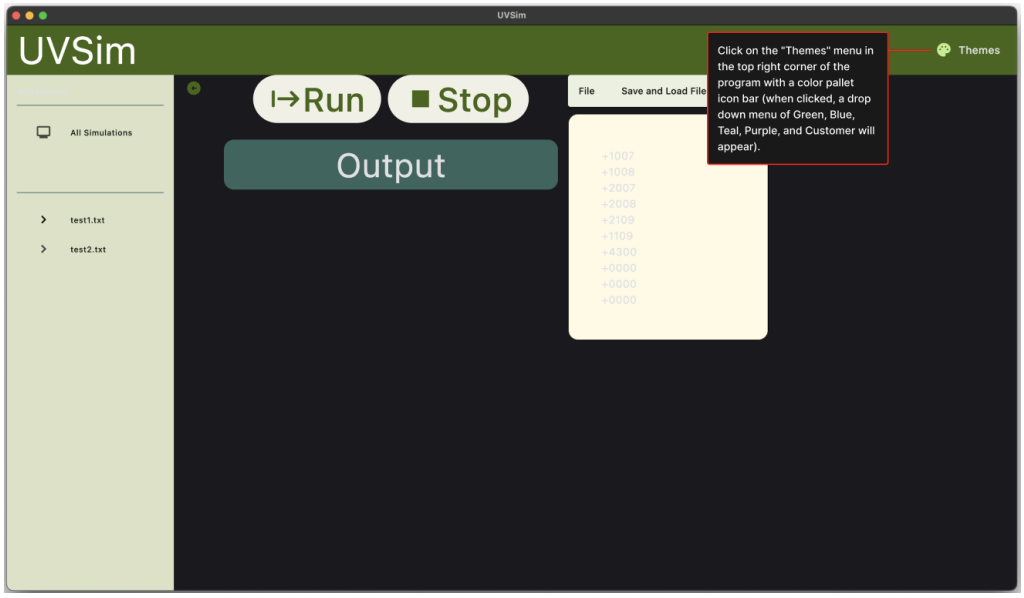
2. Use the "Save and Load File" button to save your changes in preparation for running the program.

Predefined Theme Customization

The UVSim Emulator provides options for customizing the user interface by doing the following:

1. Click on the "Themes" menu in the top right corner of the program, which has a color pallet icon bar (when clicked, a dropdownGreen, Blue, Teal, Purple, and

Customer dropdown menu will appear).

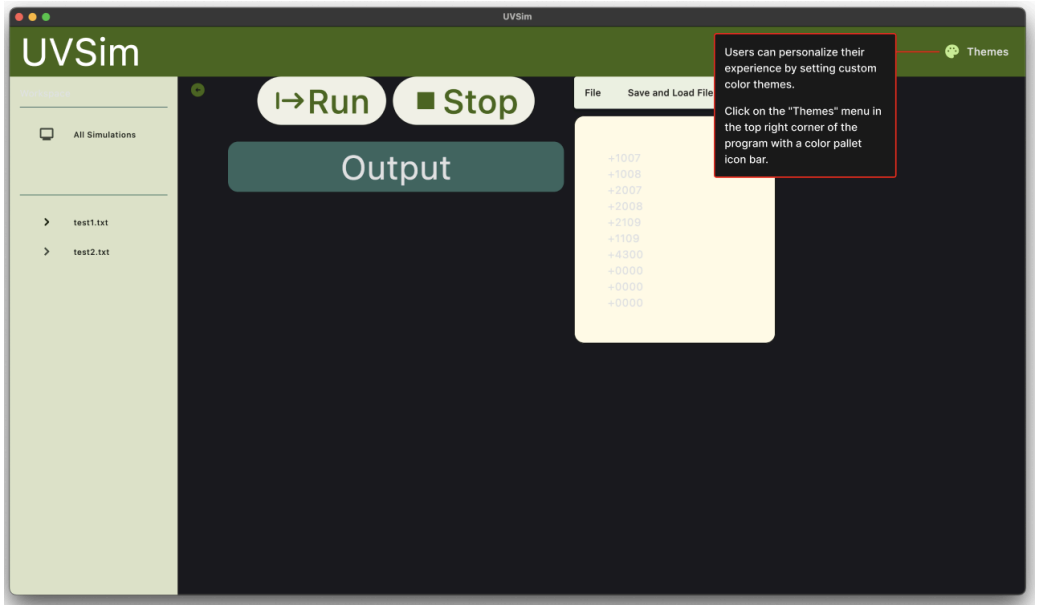


2. Select green, blue, teal, or purple from the dropdown menu to customize the interface with a predefined color theme.

Custom Theme

Users can personalize their experience by setting custom color themes:

1. Click on the "Themes" menu in the top right corner of the program with a color pallet icon bar.

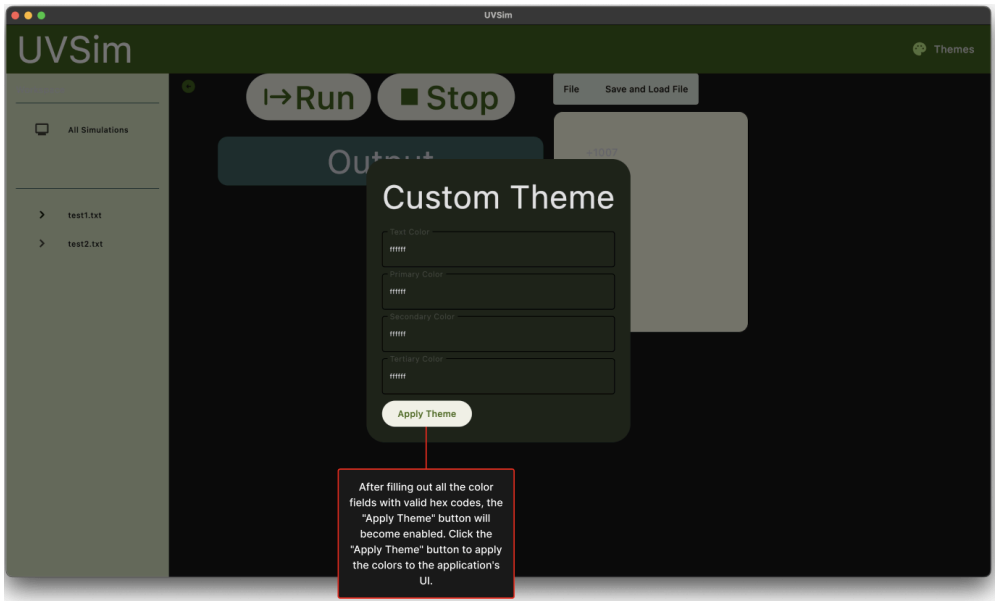


2. Select "Custom" in the dropdown menu.

3. Enter Hex Color Codes:

- Users will be presented with a dialog box containing four input fields, each corresponding to different aspects of the application's UI:

    - Text Color: This sets the color of the text throughout the application (ensuring it contrasts nicely with the background for readability).
    - Primary Color: This is used for most interactive elements, like buttons and links (choose a color that stands out but is pleasant to the eye).
    - Secondary Color: This color is used for elements that are not in the foreground but need distinction.
    - Tertiary Color: Used for accents and less dominant elements.
- Each field requires a valid hex color code. Hex color codes start with a # followed by six hexadecimal digits (e.g., #FFFFFF for white). Invalid entries will prompt an error and will not be accepted.



4. After filling out all the color fields with valid hex codes, the "Apply Theme" button will become enabled. Click the "Apply Theme" button to apply the colors to the application's UI.

Exiting the UVSim Emulator

The program can be exited in several ways depending on your operating system and how the application is running:

1. Close the Web Browser: If you are running the UVSim Emulator in a web browser, you can close the browser window or tab containing the application. This will terminate the program.
2. Terminate the Python Process: If you started the UVSim Emulator using a Python script (main.py), you can terminate the program by stopping the Python process.
    - On Windows: Press the Ctrl + C in the command prompt window where the Python process runs.
    - On macOS or Linux: Press the Ctrl + Z or Ctrl + C keyboard shortcuts to send a termination signal to the Python process.

3. User Stories / Use Cases

Description

- This section outlines user stories and a use case related to the UVSim machine language learning program.

User Stories

User Story #1

As an educator, I want to help my students learn machine language and computer architecture to further their education in computer science.

User Story #2

As a computer science student, I want to use an interactive and user-friendly machine language learning program to gain hands-on experience in machine learning.

Use Cases

Use Case #1 - File Loading (old format):

- **Actor:** user
- **System:** file management module
- **Goal:** load a file in the old four-digit word format
- **Steps:**
    1. User selects the option to load a file.
    2. User selects a file in the old format.
    3. The system converts the file to the new six-digit word format with leading zeroes.
    4. The converted file is loaded into the application.
    5. The user can now edit or execute the file in the new format.

Use Case #2 - File Loading (new format):

- **Actor:** user
- **System:** file management module
- **Goal:** load a file in the new six-digit word format
- **Steps:**
    1. User selects the option to load a file.
    2. User selects a file in the new format.
    3. The file is loaded into the application without conversion.
    4. The user can now edit or execute the file in the new format.

Use Case #3 - Store Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** successfully store a value in an address
- **Steps:**
    1. Retrieve the current value stored in the accumulator
    2. Store the retrieved value in a specified address
    3. Print a message indicating that the store operation was successful

Use Case #4 - Load/Store Operation (invalid address)

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** test if an exception is raised when an invalid address is input
- **Steps:**
    1. Parse function code
    2. Identify the target memory address from the last two digits of the word
    3. If the address is invalid, print a message indicating that the operation failed

Use Case #5 - Load/Store Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** successfully load a value into the accumulator and store it in an address
- **Steps:**
    1. Parse function code
    2. Identify the target memory address from the last two digits of the word
    3. Fetch value from identified memory address
    4. Copy the fetched value into the accumulator register
    5. Retrieve the current value stored in the accumulator
    6. Store the retrieved value in a specified address
    7. Print a message indicating that the store operation was successful

Use Case #6 - Add Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** add a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator)
- **Steps:**
    1. Parse function code
    2. Identify the target memory address from the last two digits of the word

3. Fetch value from identified memory address
4. Add the fetched value to the current value in the accumulator
5. Update the accumulator with the result

Use Case #7 - Subtract Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** subtract a word from a specific location in memory from the word in the accumulator (leave the result in the accumulator)
- **Steps:**
    1. Parse function code
    2. Identify the target memory address from the last two digits of the word
    3. Fetch value from identified memory address
    4. Subtract the fetched value from the current value in the accumulator
    5. Update the accumulator with the result

Use Case #8 - Divide Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** divide the word in the accumulator by a word from a specific location in memory (leave the result in the accumulator)
- **Steps:**
    1. Parse function code
    2. Identify the target memory address from the last two digits of the word
    3. Fetch value from identified memory address
    4. Check if the divisor (i.e., the fetched value) is not zero
    5. Divide the current value in the accumulator by the fetched value
    6. Update the accumulator with the result

Use Case #9 - Multiply Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor

- **Goal:** multiply a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator)
- **Steps:**
    1. Parse function code
    2. Identify the target memory address from the last two digits of the word
    3. Fetch value from identified memory address
    4. Multiply the fetched value by the current value in the accumulator
    5. Update the accumulator with the result

Use Case #10 - Read Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** read a word from the keyboard into a specific location in memory
- **Steps:**
    1. Prompt the user for input
    2. Validate the input format (+/- followed by a four-digit value)
    3. Store the validated input in the specified memory address
    4. Print a confirmation message if the input is successfully stored

Use Case #11 - Read Operation (invalid input)

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** test what would happen if the input caused an error
- **Steps:**
    1. Prompt the user for input
    2. Validate the input format (+/- followed by a four-digit value)
    3. If the input is bad, print a message indicating that the operation failed

Use Case #12 - Write Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** write a word from a specific location in memory to screen

- **Steps:**
    1. Retrieve the value stored in the specified memory address
    2. Print the retrieved value as output

Use Case #13 - Write Operation (invalid address)

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** test what would happen if the address caused an error
- **Steps:**
    1. Retrieve the value stored in the specified memory address
    2. If the address is bad, print a message indicating that the operation failed

Use Case #14 - Branch Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** branch to a specific location in memory
- **Steps:**
    1. Retrieve the target memory address from the instruction
    2. Update the program counter to point to the target memory address
    3. Print a message indicating the execution of the branch operation

Use Case #15 - BranchNeg Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** branch to a specific location in memory if the accumulator is negative
- **Steps:**
    1. Check if the sign of the value in the accumulator is negative
    2. If negative, update the program counter to point to the target memory address
    3. If not negative, proceed to the next instruction
    4. Print a message indicating the execution of the branchNeg operation

Use Case #16 - BranchZero Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** branch to a specific location in memory if the accumulator is zero
- **Steps:**
    1. Check if the value in the accumulator is zero
    2. If zero, update the program counter to point to the target memory address
    3. If not zero, proceed to the next instruction
    4. Print a message indicating the execution of the branchZero operation

Use Case #17 - Halt Operation

- **Actor:** program or algorithm
- **System:** memory management and code processor
- **Goal:** Stop the program
- **Steps:**
    1. Print a message indicating that the program execution has halted
    2. End the execution loop

Use Case #18 - File Saving

- **Actor:** user
- **System:** file management module
- **Goal**: save a file in the current format (either old or new)
- **Steps:**
    1. User selects the option to save a file.
    2. User specifies the filename and location.
    3. The file is saved in the current format (either old or new) based on its original format or user preference.

Use Case #19 - Multiple File Management:

- **Actor:** user
- **System:** application interface
- **Goal:** manage and interact with multiple open files simultaneously
- **Steps:**
    1. User opens multiple files within the application.
    2. The application displays tabs or sub-windows for each open file, allowing the user to switch between them.
    3. The user can edit or execute each file independently within the application instance.
    4. Changes made to each file are saved separately without interference from other open files.

4. Functional Specifications

Description

- This section outlines the functional and non-functional requirements for the UVSim application.

Functional Requirements

1. The system shall support a 100-word memory for BasicML words.
2. The system shall read a word from the keyboard into a specific location in memory if the first two digits of the BasicML word equals 10.
3. The system shall write a word from a specific location in memory to the screen if the first two digits of the BasicML word equals 11.
4. The system shall load a word from a specific location in memory into the accumulator if the first two digits of the BasicML word equals 20.
5. The system shall store a word from the accumulator in a specific location in memory if the first two digits of the BasicML equals 21.
6. The system shall add a word from a specific location in memory to the word in the accumulator, leaving the result in the accumulator if the first two digits of the BasicML word equals 30.

7. The system shall subtract a word from a specific location in memory from the word in the accumulator, leaving the result in the accumulator if the first two digits of the BasicML equals 31.
8. The system shall divide the word in the accumulator by a word from a specific location in memory, leaving the result in the accumulator if the first two digits of the BasicML equals 32.
9. The system shall multiply a word from a specific location in memory to the word in the accumulator, leaving the result in the accumulator if the first two digits of the BasicML equals 33.
10. The system shall branch to a specific location in memory if the first two digits of the BasicML equals 40.
11. The system shall branch to a specific location in memory if the accumulator is negative and the first two digits of the BasicML equals 41.
12. The system shall branch to a specific location in memory if the accumulator is zero and the first two digits of the BasicML equals 42.
13. The system shall stop the program if the first two digits of the BasicML equals 43.
14. The program must read the words in memory line-by-line unless a branch operation performs a jump.
15. The system shall halt program execution when a halt instruction is encountered, indicating the end of program execution.
16. During input operations, the system shall provide clear instructions to the user, guiding them on the expected format and type of input.
17. The system shall allow users to customize the color scheme of the application interface.
18. The system shall allow users to load program files into the GUI for inspection and editing before execution.
19. The system shall support loading program files from any user-specified directory.
20. The system shall allow users to save edited program files to a user-chosen directory.

21. The application shall support data files containing up to 250 lines, each corresponding to a memory register ranging from 000 to 249.
22. The application shall use three-digit memory addresses to reference lines within the expanded address space.
23. The application shall enforce the limitation of not allowing more than 250 lines in any loaded or edited file.
24. The application shall ensure that any command referencing a line number outside the range of 000 to 249 is considered invalid.
25. The application shall handle six-digit math operations correctly, including overflow handling, for the new word size.
26. The application shall append a zero to the beginning of each functional code to represent operations in the new six-digit format (e.g., 010 instead of 10 for a READ command).
27. The application shall support old and new file formats, allowing users to load, edit, and execute files with four or six-digit words.
28. The application shall not allow the mixing and matching of four-digit and six-digit words within an individual file, ensuring that each file is consistent with one format.

Non-functional Requirements

1. The system's user interface shall be designed according to the principles of minimalism, with intuitive controls and concise instructions to facilitate ease of use.
2. The system's reliability shall be demonstrated by achieving at least 99.9% uptime during continuous operation, ensuring consistent and accurate program execution.
3. The system's response time to user interactions shall be under 1 second, ensuring efficient execution of instructions and a seamless user experience.
4. The program shall take at most 20 minutes to be installed.
5. The application's user interface shall clearly distinguish between old and new file formats, allowing users to differentiate between them at load or runtime.

6. The application shall efficiently process and execute commands, providing responsive performance even when handling large files with the expanded address space.
7. The application shall maintain data integrity and reliability when handling multiple open files, ensuring that changes made to one file do not affect the integrity of others.

5. Class Diagrams & Definitions

Description

- This section provides a visual representation of each application feature outlined in the Application Instructions section. It serves as a guide to understanding how to utilize each feature effectively within the application's interface.

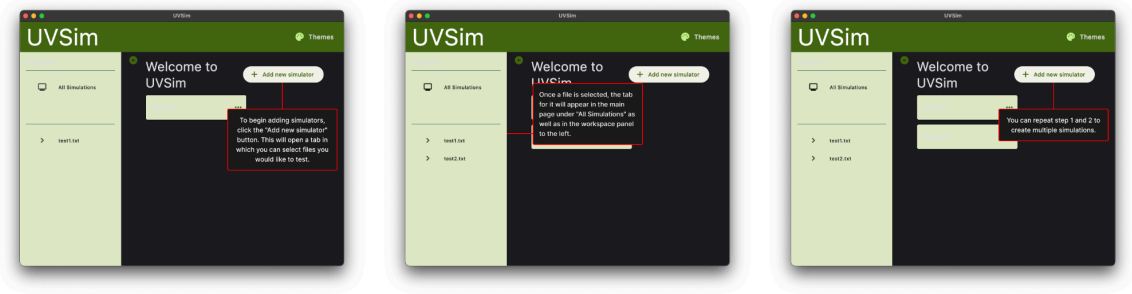
Class Diagrams

1. Running the Program and Simulators

Running the Program and Simulators

2. Adding Simulators

Adding Simulators

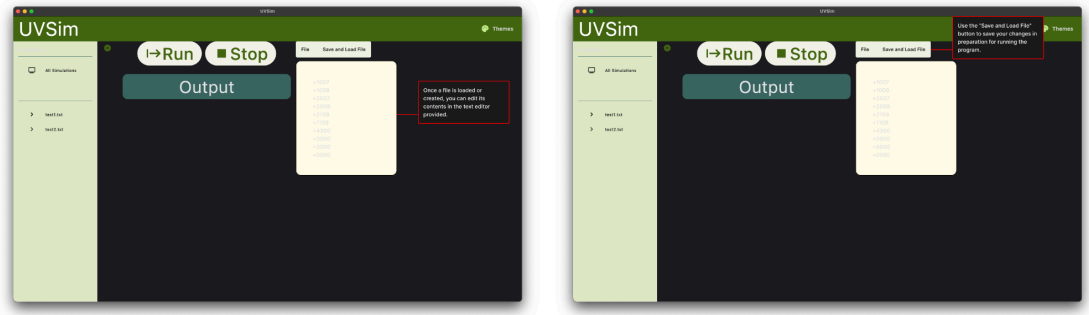


3. File Management

File Management

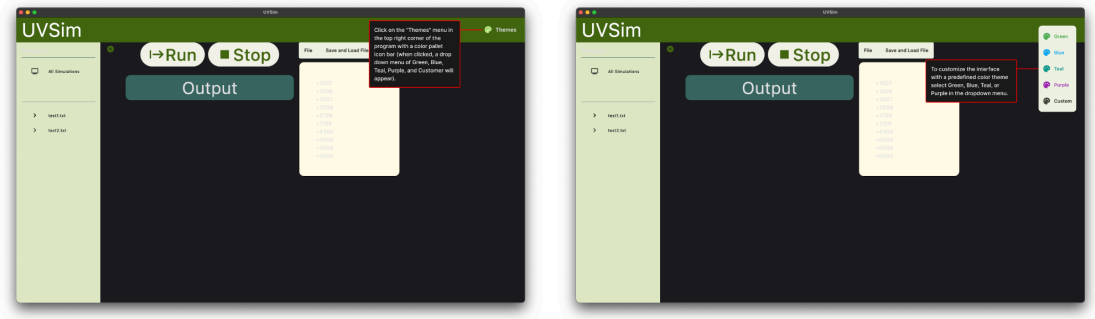
4. Editing Selected Files

Editing Selected Files



5. Predefined Theme Customization

Predefined Theme Customization



6. Custom Theme

Custom Theme

Class Definitions

main Function

- Purpose of function: The main function initializes and configures the main page of the UVSim application.
- Input: page (ft.Page) - The page object associated with the main page.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The main page of the UVSim application is initialized and configured.

route_change

- Purpose of function: Handles route changes in the application.
- Input: e (Event) - Event object representing the route change event.

- Return Value: None.
- Pre-conditions: UVSim application instance must be initialized.
- Post-conditions: Depending on the route, the appropriate view is displayed and updated.

view_pop

- Purpose of function: Pops a view from the page's views stack and navigates to the top view.
- Input: view - The view to be popped from the views stack.
- Return Value: None.
- Pre-conditions: UVSim application instance must be initialized.
- Post-conditions: The top view is displayed after removing the specified view from the views stack.

AppLayout Class

- Purpose of class: The AppLayout class represents the layout of the UVSim application, managing various UI elements and views.

home_page_view

- Purpose of function: Constructs the home page view with a welcome message and options to add new simulators.
- Return Value: ft.Column - The home page view.
- Pre-conditions: None.
- Post-conditions: The home page view is constructed and returned.

set_sim_view

- Purpose of function: Sets the active view to a simulator page based on the provided file name.
- Input: file_name (str) - The name of the file associated with the simulator.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The active view is set to the simulator page corresponding to the provided file name.

set_all_sim_view

- Purpose of function: Sets the active view to the home page view and hydrates it with all available simulators.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The active view is set to the home page view with all available simulators.

page_resize

- Purpose of function: Handles resizing of the page and adjusts the active view accordingly.
- Input: e (Event) - Event triggering the resizing.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The active view is resized based on the page dimensions.

hydrate_all_sim_view

- Purpose of function: Hydrates the home page view with all available simulators.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The home page view is populated with controls representing available simulators.

file_picker_result

- Purpose of function: Handles the result of file picker selection and updates the simulators accordingly.
- Input: e (ft.FilePickerResultEvent) - Event containing the selected files.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The simulators are updated based on the selected files.

go_to_simulator

- Purpose of function: Navigates to the simulator page associated with the clicked control.
- Input: e - Event containing information about the clicked control.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The page navigates to the simulator page corresponding to the clicked control.

toggle_nav_rail

- Purpose of function: Toggles the visibility of the navigation rail sidebar.
- Input: e - Event triggering the toggle action.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The visibility of the navigation rail sidebar is toggled.

EventHandler Class

- Purpose of class: The EventHandler class manages event handling for displaying output and getting user input.

display_output

- Purpose of function: Displays output.
- Input: Output value ("output").
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: Output is displayed using the OutputControl instance.

get_user_input

- Purpose of function: Retrieves user input.
- Input: None.
- Return Value: User input.
- Pre-conditions: None.
- Post-conditions: User input is retrieved using the InputControl instance.

FileHandler Class

- Purpose of class: The FileHandler class is responsible for managing file-handling functionality within the UVSim application.

build

- Purpose of function: Builds the UI layout for the file handler.
- Return Value: ft.Column - The column containing UI elements.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: The UI layout for the file handler is constructed and returned.

save_load_file

- Purpose of function: Saves or loads a file depending on the current state.
- Input: e (Event) - Event triggering the function call.
- Return Value: None.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: If a file path is available, the file is saved and then loaded into the operation.

file_picker_result

- Purpose of function: Handles the result of file picking operation.
- Input: e (ft.FilePickerResultEvent) - Result event from the file picker.
- Return Value: None.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: File path is retrieved and file content is displayed.

save_file_result

- Purpose of function: Handles the result of file saving operation.
- Input: e (ft.FilePickerResultEvent) - Result event from file saving.
- Return Value: None.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: File is saved to the specified path.

save_text

- Purpose of function: Saves text content to a file.
- Input: path (str) - Path to save the file, value (str) - Content to be saved.

- Return Value: None.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: Text content is saved to the specified file path.

open_file

- Purpose of function: Opens a file and reads its content.
- Input: user_file (str) - Path of the file to be opened.
- Return Value: str - Content of the file.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: File content is read and returned.

get_text_field_value

- Purpose of function: Retrieves the value of the text field.
- Return Value: str - Value of the text field.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: Value of the text field is returned.

run_program

- Purpose of function: Runs the program operation.
- Return Value: None.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: Program operation is executed.

stop_program

- Purpose of function: Stops the program operation.
- Return Value: None.
- Pre-conditions: FileHandler instance must be initialized.
- Post-conditions: Program operation is stopped.

InputControl Class

- Purpose of class: The InputControl class manages the input control functionality within the UVSim application.

build

- Purpose of function: Builds the UI layout for the input control.
- Return Value: None.

- Pre-conditions: InputControl instance must be initialized.
- Post-conditions: The UI layout for the input control is constructed and returned.

get_input

- Purpose of function: Retrieves user input.
- Return Value: str - User input.
- Pre-conditions: InputControl instance must be initialized.
- Post-conditions: User input is obtained and returned.

close_dlg

- Purpose of function: Closes the input dialog and updates the page.
- Return Value: None.
- Pre-conditions: InputControl instance must be initialized.
- Post-conditions: The dialog is closed, and the page is updated.

textfield_change

- Purpose of function: Handles text field changes and updates the send button state.
- Return Value: None.
- Pre-conditions: InputControl instance must be initialized.
- Post-conditions: The send button state is updated based on text field changes.

OutputControl Class

- Purpose of class: The OutputControl class represents a UI control for displaying output in the UVSim application.

display_output

- Purpose of function: Displays output on the UI.
- Input: output (str) - The output text to be displayed.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The output text is displayed on the UI.

Sidebar Class

- Purpose of class: The Sidebar class represents a user control for managing navigation within the UVSim application.

build

- Purpose of function: Constructs the UI layout for the sidebar.
- Return Value: ft.Container - The constructed UI layout for the sidebar.
- Pre-conditions: None.
- Post-conditions: The UI layout for the sidebar is constructed and returned.

sync_sim_destinations

- Purpose of function: Synchronizes the simulator destinations in the sidebar.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The simulator destinations in the sidebar are synchronized.

toggle_nav_rail

- Purpose of function: Toggles the visibility of the navigation rail sidebar.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The visibility of the navigation rail sidebar is toggled.

top_nav_change

- Purpose of function: Handles changes in the top navigation rail.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The selected index of the top navigation rail is updated.

bottom_nav_change

- Purpose of function: Handles changes in the bottom navigation rail.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The selected index of the bottom navigation rail is updated.

Topbar Class

- Purpose of class: The Topbar class represents the top app bar of the UVSim application, providing options to change themes.

build

- Purpose of function: Constructs the UI layout for the top app bar.

- Return Value: ft.AppBar - The constructed UI layout for the top app bar.
- Pre-conditions: None.
- Post-conditions: The UI layout for the top app bar is constructed and returned.

green_theme

- Purpose of function: Applies the green theme to the UVSim application.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The green theme is applied to the UVSim application.

blue_theme

- Purpose of function: Applies the blue theme to the UVSim application.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The blue theme is applied to the UVSim application.

Teal_theme

- Purpose of function: Applies the teal theme to the UVSim application.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The teal theme is applied to the UVSim application.

purple_theme

- Purpose of function: Applies the purple theme to the UVSim application.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The purple theme is applied to the UVSim application.

is_hex_color

- Purpose of function: Checks if the given string represents a valid hexadecimal color.
- Return Value: bool - True if the given string is a valid hexadecimal color, False otherwise.
- Pre-conditions: None.
- Post-conditions: The validity of the hexadecimal color string is determined.

textfield_change

- Purpose of function: Handles changes in the text fields for custom theme creation.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The state of the apply button is updated based on the text field values.

custom_theme

- Purpose of function: Allows the user to create a custom theme for the UVSim application.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The custom theme is created and applied to the UVSim application.

Operations Class

- Purpose of class: The Operations class manages the execution of operations within the UVSim application.

read_file

- Purpose of function: Reads instructions from a file and stores them in memory.
- Input: filename (str) - The name of the file to be read.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: Instructions are read from the file and stored in memory.

set_got_input

- Purpose of function: Sets the flag indicating if input is received.
- Input: got_input (bool) - Flag indicating if input is received.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: The flag indicating if input is received is set.

set_u_input

- Purpose of function: Sets the user input.
- Input: u_input (str) - The user input.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: The user input is set.

get_output

- Purpose of function: Retrieves the output value.
- Return Value: str - The output value.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: The output value is retrieved.

stop_execution

- Purpose of function: Stops the program execution.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: The program execution is stopped.

IO_op

- Purpose of function: Performs Input/Output operation.
- Input: op (int) - The operation code, address (int) - The memory address.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: Input/Output operation is performed.

load_store_op

- Purpose of function: Performs Load/Store operation.
- Input: op (int) - The operation code, address (int) - The memory address.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: Load/Store operation is performed.

arithmetic_op

- Purpose of function: Performs Arithmetic operation.

- Input: op (int) - The operation code, address (int) - The memory address.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: Arithmetic operation is performed.

branch_op

- Purpose of function: Performs Branch operation.
- Input: op (int) - The operation code, address (int) - The memory address.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: Branch operation is performed.

execute

- Purpose of function: Executes the program.
- Return Value: None.
- Pre-conditions: Operations instance must be initialized.
- Post-conditions: The program is executed.

OperationsError Class

- Purpose of class: The OperationsError class represents an error that occurs during operations within the UVSim application.

SimulatorPage Class

Purpose of class: The SimulatorPage class represents the page displaying the simulation controls and file handling for a specific simulator.

set_sim_id

- Purpose of function: Sets the identifier for the simulator.
- Input: sim_id (str) - Identifier for the simulator.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: The identifier for the simulator is updated.

run_button_result

- Purpose of function: Handles the result of clicking the run button.
- Input: e (Event) - Event triggering the function call.
- Return Value: None.

- Pre-conditions: None.
- Post-conditions: The file associated with the simulator is loaded into the register, and the simulation is executed.

stop_button_result

- Purpose of function: Handles the result of clicking the stop button.
- Input: e (Event) - Event triggering the function call.
- Return Value: None.
- Pre-conditions: None.
- Post-conditions: Execution of the simulation is stopped.

buttons_layout

- Purpose of function: Constructs the layout for the simulation control buttons.
- Return Value: ft.Column - The column containing simulation control buttons.
- Pre-conditions: None.
- Post-conditions: The layout for simulation control buttons is constructed and returned.

run_button

- Purpose of function: Creates the 'Run' button for the simulator.
- Return Value: ft.ElevatedButton - The 'Run' button.
- Pre-conditions: None.
- Post-conditions: The 'Run' button is created and returned.

stop_button

- Purpose of function: Creates the 'Stop' button for the simulator.
- Return Value: ft.ElevatedButton - The 'Stop' button.
- Pre-conditions: None.
- Post-conditions: The 'Stop' button is created and returned.

6. Unit Tests

Description

- This section provides the descriptions, case reference, input, expected output, and test success or failure for each unit test within the UVSim application.

Tests

Unit Test: test_load_operation

- Description: Tests data by placing a word at a specific memory address, then performs the load operation and checks if the accumulator contains the expected word.
- Case Reference: Load Operation
- Input: test_address = 1, test_word = "+1234"
- Expected Output: The accumulator should contain the word "+1234".
- Test Success/Failure: The test succeeds if self.assertEqual(self.operations.accumulator, test_word) passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_store_operation

- Description: Sets up test data by putting a word into the accumulator, then performs the store operation and checks if the memory at a specified address contains the expected word.
- Case Reference: Store Operation
- Input: test_address = 2, test_word = "-5678"

- Expected Output: The memory at address 2 should contain the word "-5678".
- Test Success/Failure: The test succeeds if self.assertEqual(self.operations.registers[test_address], test_word) passes without raising an exception. The test fails if the assertion raises an exception, indicating that the memory at the specified address did not contain the expected word.

Unit Test: test_invalid_address

- Description: Attempts to load an invalid address.
- Case Reference: Invalid Address Handling
- Input: Invalid operation code (opcode 7)
- Expected Output: No errors raised.
- Test Success/Failure: The test succeeds if no error is raised when an invalid address is attempted. The test fails if an error is raised, indicating that the expected error message was not printed.

Unit Test: test_load_store_op

- Description: Sets up test data for each operation and checks if the accumulator and memory are updated correctly. (Combines both load and store operations within a single test case.)
- Case Reference: Combined Load and Store Operations
- Input: test_address_load = 1, test_word_load = "+9876"
- Expected Output: Accumulator should contain the word "+9876" after the load operation. After the store operation, the memory at address 3 should contain the word "-5432".
- Test Success/Failure: Both assertions (self.assertEqual(self.operations.accumulator, test_word_load) and self.assertEqual(self.operations.registers[test_address_store],

test_word_store)) must pass without raising exceptions for the test to succeed. If either assertion fails, it indicates that the observed outcomes did not match the expected outcomes.

Unit Test: test_add_1

- Description: Tests whether "+1208" was added to the accumulator, which starts with "+0000".
- Case Reference: Add Operation
- Input: accumulator = "+0000", word to add = "+1208"
- Expected Output: The accumulator should contain the word "+1208".
- Test Success/Failure: The test succeeds if self.assertEqual("+1208", self.operations.accumulator, "Add operation failed") passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_add_2

- Description: Tests whether "+0000" was added to the accumulator, which starts with "+0000".
- Case Reference: Add Operation
- Input: accumulator = "+0000", word to add = "+0000"
- Expected Output: The accumulator should contain the word "+0000".
- Test Success/Failure: The test succeeds if self.assertEqual("+0000", self.operations.accumulator, "Add operation failed") passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_subtract_1

- Description: Tests whether "+1000" was subtracted from "+1208".
- Case Reference: Subtract Operation
- Input: accumulator = "+1208", word to subtract = "+1000"
- Expected Output: The accumulator should contain the word "+0208".
- Test Success/Failure: The test succeeds if self.assertEqual("+0208", self.operations.accumulator, "Add operation failed") passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_subtract_2

- Description: Tests whether "+0001" was subtracted from "+0000".
- Case Reference: Subtract Operation
- Input: accumulator = "+0000", word to subtract = "+0001"
- Expected Output: The accumulator should contain the word "-0001".
- Test Success/Failure: The test succeeds if self.assertEqual("-0001", self.operations.accumulator, "Add operation failed") passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_divide_1

- Description: Tests whether "+1208" was divided by "+0002".
- Case Reference: Divide Operation
- Input: accumulator = "+1208", word to divide = "+0002"
- Expected Output: The accumulator should contain the word "+0604".

- Test Success/Failure: The test succeeds if self.assertEqual("+0604", self.operations.accumulator, "Add operation failed") passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_divide_2

- Description: Tests whether "+9000" was divided by "+0002".
- Case Reference: Divide Operation
- Input: accumulator = "+9000", word to divide = "+0002"
- Expected Output: The accumulator should contain the word "+4500".
- Test Success/Failure: The test succeeds if self.assertEqual("+4500", self.operations.accumulator, "Add operation failed") passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_multiply_1

- Description: Tests whether "+1208" was multiplied by "+0002".
- Case Reference: Multiply Operation
- Input: accumulator = "+1208", word to multiply = "+0002"
- Expected Output: The accumulator should contain the word "+2416".
- Test Success/Failure: The test succeeds if self.assertEqual("+2416", self.operations.accumulator, "Add operation failed") passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_multiply_2

- Description: Tests whether "+9000" was multiplied by "+0000".

- Case Reference: Multiply Operation
- Input: accumulator = "+9000", word to multiply = "+0000"
- Expected Output: The accumulator should contain the word "+0000".
- Test Success/Failure: The test succeeds if self.assertEqual("+0000", self.operations.accumulator, "Add operation failed") passes without raising an exception. The test fails if the assertion raises an exception, indicating that the accumulator did not contain the expected word.

Unit Test: test_invalid

- Description: Tests whether using an opcode other than 30-33 would trigger an error or not.
- Case Reference: Invalid Operation Handling
- Input: accumulator = "+0000", opcode = "90", word = "+9500"
- Expected Output: The accumulator should contain the word "+0000" and the program will print "Invalid arithmetic operation code."
- Test Success/Failure: The test succeeds if no exception is raised when an invalid operation is attempted. The test fails if an exception is raised, indicating that the expected error message was not printed.

Unit Test: test_read

- Description: Tests whether "+2244" is successfully loaded into register 2.
- Case Reference: Read Operation
- Input: address = 2, user_input = "+2244"
- Expected Output: Register 2 should contain the value "+2244".

- Test Success/Failure: The test succeeds if self.assertEqual(self.operations.registers[2], expected_output) is true and fails if the value is not read into the address or the wrong number is read there.

Unit Test: test_read2

- Description: Tests whether "-1440" is successfully loaded into register 3.
- Case Reference: Read Operation
- Input: address = 3, user_input = "-1440"
- Expected Output: Register 3 should contain the value "-1440".
- Test Success/Failure: The test succeeds if self.assertEqual(self.operations.registers[3], expected_output) is true and fails if the value is not read into the address or the wrong number is read there.

Unit Test: test_read3

- Description: Tests a value ("+4466") to see if it is loaded into register 4.
- Case Reference: Read Operation
- Input: address = 4, user_input = "+4466"
- Expected Output: Register 4 should contain the value "+4466".
- Test Success/Failure: The test succeeds if self.assertEqual(self.operations.registers[4], expected_output) is true and fails if the value is not read into the address or the wrong number is read there.

Unit Test: test_bad_input2

- Description: Tests another invalid value ("+12345") to see if it is loaded into register 5.
- Case Reference: Read Operation

- Input: address = 5, user_input = "+12345"
- Expected Output: Register 5 should remain unchanged and contain the value "+0000".
- Test Success/Failure: The test succeeds if self.assertEqual(self.operations.registers[5], initial_value) is true, showing that the value in the register is still the initial value and therefore the input value was not read into it.

Unit Test: test_bad_input3

- Description: Tests another invalid value ("+00f0") to see if it is loaded into register 6.
- Case Reference: Read Operation
- Input: address = 6, user_input = "+00f0"
- Expected Output: Register 6 should remain unchanged and contain the value "+0000".
- Test Success/Failure: The test succeeds if self.assertEqual(self.operations.registers[6], initial_value) is true, showing that the value in the register is still the initial value and therefore the input value was not read into it.

Unit Test: test_write

- Description: Tests whether "+1111" is printed to the console.
- Case Reference: Write Operation
- Input: address = 0
- Expected Output: "+1111" should be printed to the console.

- Test Success/Failure: The test succeeds if self.assertEqual([expected_output1], printed_output) is true, showing that the value in the expected output (given register) matches the output.

Unit Test: test_write2

- Description: Tests whether "-1234" is printed to the console.
- Case Reference: Write Operation
- Input: address = 1
- Expected Output: "-1234" should be printed to the console.
- Test Success/Failure: The test succeeds if self.assertEqual([expected_output1], printed_output) is true, showing that the value in the expected output (given register) matches the output.

Unit Test: test_write_invalid_address

- Description: Tests whether an invalid address correctly prints an error message.
- Case Reference: Write Operation
- Input: address = 15
- Expected Output: "Error: Address {address} not found in registers." Should be printed to the console.
- Test Success/Failure: The test succeeds if the error message is printed to the console.

Unit Test: Test_branch_exec

- Description: Test the program jump to a specific address without executing any other store operation after the branch.
- Case Reference: Branch Operation

- Input: Memory = {0: "+4004", 1: "+2105", 2: "+2105", 3: "+2105", 4: "+4300", 5: "+0000"} Accumulator = "-1111"
- Expected Output: Memory at address 5 should remain +0000.
- Test Success/Failure: The test succeeds if no exception is raised by the memory at address 5 not being equal to +0000. If it fails, it's probably because the branch is not jumping to the specific address, and the simulator runs every line in memory before ending the program.

Unit Test: Test_branch_to_unknown_addr

- Description: Test the branch operation to an unknown address; the program should raise an error.
- Case Reference: Branch Operation
- Input: Memory = {0: "+4010", 1: "+2105", 2: "+2105", 3: "+2105", 4: "+4300", 5: "+0000"}
- Expected Output: OperationError with the message "Invalid memory address: 10."
- Test Success/Failure: The test succeeds if the program raises an OperationError with the expected message. Fails if no error is raised and the program runs normally.

Unit Test: test_branchNeg_exec

- Description: Test the program jump to a specific address when the accumulator is negative without executing any other store operation after the branch.
- Case Reference: BranchNeg Operation
- Input: Memory = {0: "+4104", 1: "+2105", 2: "+2105", 3: "+2105", 4: "+4300", 5: "+0000"} Accumulator = "-1111"
- Expected Output: Memory at address 5 should remain +0000.

- Test Success/Failure: The test succeeds if no exception is raised by the memory at address 5 not being equal to +0000. If it fails, it's probably because the branchNeg is not jumping to the specific address, and the simulator runs every line in memory before ending the program.

Unit Test: test_branchNeg_positive_accumulator

- Description: Test the branch negative operation with a positive accumulator; the program should run normally without executing the branch.
- Case Reference: BranchNeg Operation
- Input: Memory = {0: "+4104", 1: "+2105", 2: "+2105", 3: "+2105", 4: "+4300", 5: "+0000"} Accumulator = "+1111"
- Expected Output: Memory at address 5 should be +1111.
- Test Success/Failure: The test succeeds if no exception is raised by the memory at address 5 being equal to +1111. If it fails, it's probably because the branchNeg performs the jump even when the accumulator is positive.

Unit Test: test_branchNeg_to_unknown_addr

- Description: Test the branch operation to an unknown address; the program should raise an error.
- Case Reference: BranchNeg Operation
- Input: Memory = {0: "+4110", 1: "+2105", 2: "+2105", 3: "+2105", 4: "+4300", 5: "+0000"} Accumulator = "-1111"
- Expected Output: OperationError with the message "Invalid memory address: 10."
- Test Success/Failure: The test succeeds if the program raises an OperationError with the expected message. Fails if no error is raised and the program runs normally.

Unit Test: test_branchZero_exec

- Description: Test the program jump to a specific address if the accumulator is "0000" without executing any other store operation after the branch.
- Case Reference: BranchZero Operation
- Input: Memory = {0: "+4204", 1: "+2105", 2: "+2105", 3: "+2105", 4: "+4300", 5: "+0000"} Accumulator = "+0000"
- Expected Output: Memory at address 5 should remain +0000.
- Test Success/Failure: The test succeeds if no exception is raised by the memory at address 5 not being equal to +0000. If it fails, it's probably because the branchZero is not jumping to the specific address, and the simulator runs every line in memory before ending the program.

Unit Test: test_branchZero_non_zero_accumulator

- Description: Test the branch zero operation with a non-zero accumulator; the program should run normally without executing the branch.
- Case Reference: BranchZero Operation
- Input: Memory = {0: "+4204", 1: "+2105", 2: "+2105", 3: "+2105", 4: "+4300", 5: "+0000"} Accumulator = "+1111"
- Expected Output: Memory at address 5 should be +1111.
- Test Success/Failure: The test succeeds if no exception is raised by the memory at address 5 being equal to +1111. If it fails, it's probably because the branchZero performs the jump even when the accumulator is non-zero.

Unit Test: test_branchZero_to_unknown_addr

- Description: Test the branch operation to an unknown address; the program should raise an error.

- Case Reference: BranchZero Operation
- Input: Memory = {0: "+4210", 1: "+2105", 2: "+2105", 3: "+2105", 4: "+4300", 5: "+0000"} Accumulator = "+0000"
- Expected Output: OperationError with the message "Invalid memory address: 10."
- Test Success/Failure: The test succeeds if the program raises an OperationError with the expected message. Fails if no error is raised and the program runs normally.

Unit Test: test_halt_program

- Description: Test the program ends when the Halt word is executed.
- Case Reference: Halt
- Input: Memory = {0: "+4300", 1: "+2102", 2: "+0000"}
- Expected Output: Memory at address 5 should remain +0000.
- Test Success/Failure: The test succeeds if Memory at address 5 remains +0000. It means that the program stopped before reaching the store operation.

Unit Test: test_execution_run

- Description: Test the program runs without errors.
- Case Reference: Execution
- Input: Test1.txt file
- Expected Output: No errors Raised.
- Test Success/Failure: Test succeeds when no Error is raised.

Unit Test: test_bad_words

- Description: Test the programs fail when the file contains bad words; this test will create multiple files with bad words and run it.

- Case Reference: Execution
- Input: bad_words = ["+20458", "2054+", "+204", "-20458", "-204"]
- Expected Output: Error raised in every word on that list.
- Test Success/Failure: Test succeeds when an Error is raised. Each word on that list doesn't follow the expected word format. Therefore when reading the file it should raise an error on the word and the line it was found.

Unit Test: test_101_words

- Description: Test the program doesn't run when the file contains 101 or more words.
- Case Reference: Execution
- Input: Variable
- Expected Output: Error raised ="Error on line 101: Too many words in the file"
- Test Success/Failure: Since we only support 100-word memory, this test should raise an error for trying to run a file with 101 words on it.

Unit Test: test_100_words

- Description: Test the program will run with 100 words or less.
- Case Reference: Execution
- Input: Variable
- Expected Output: No errors Raised.
- Test Success/Failure: Since we only support 100-word memory, this test should not raise any error when trying to run a file with 100 words on it.

7. Future Road Map

Description

- Expanding the UVSim application beyond its current state could significantly enhance its usefulness and user experience. Below are potential future development directions, including new features and additional platform support.

New Features

- **Debugging tools:** Introduce features to aid students in debugging their BasicML programs. This could include breakpoints, step-by-step execution, variable inspection, and more.
- **Integrated help system:** Provide contextual help within the application, explaining BasicML instructions, CPU architecture, and programming concepts. You could also add video aids. This would serve as a valuable learning resource for students.
- **Integration with Learning Management Systems (LMS):** Enable integration with learning management systems (LMS) commonly used in educational institutions. This would streamline assignment submissions, grading, and tracking of student progress.
- **Accessibility features:** Ensure the application is accessible to users with disabilities by incorporating features such as screen reader support, keyboard navigation, and high-contrast mode.
- **Localization:** Translate the application into multiple languages to cater to a diverse user base worldwide. This would make the UVSim accessible to non-English-speaking students and educators.

Additional Platforms

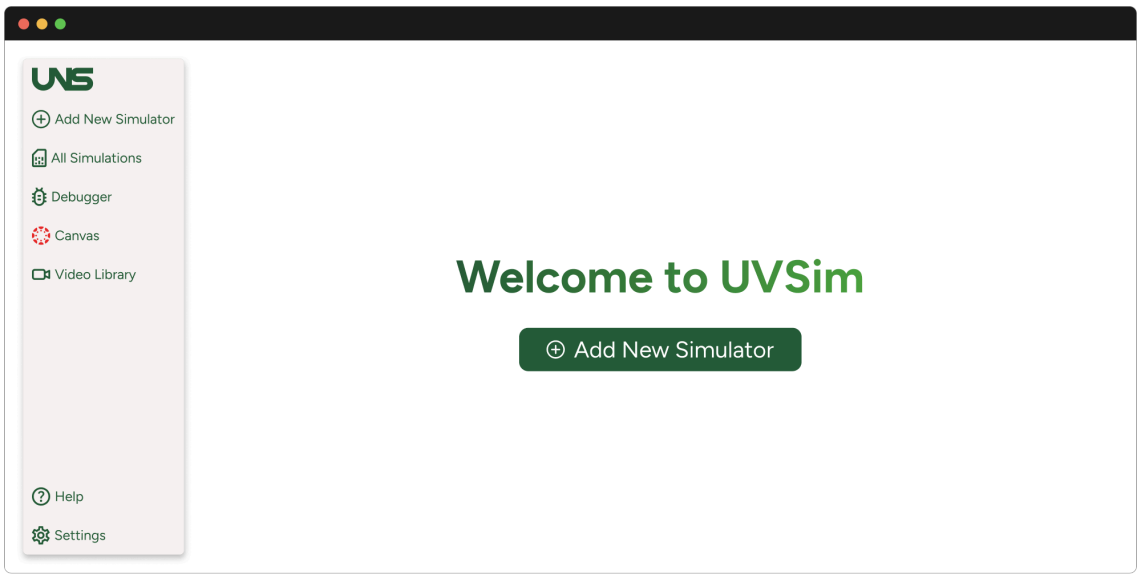
Along with being a console application, UVSim could expand to the following platforms:

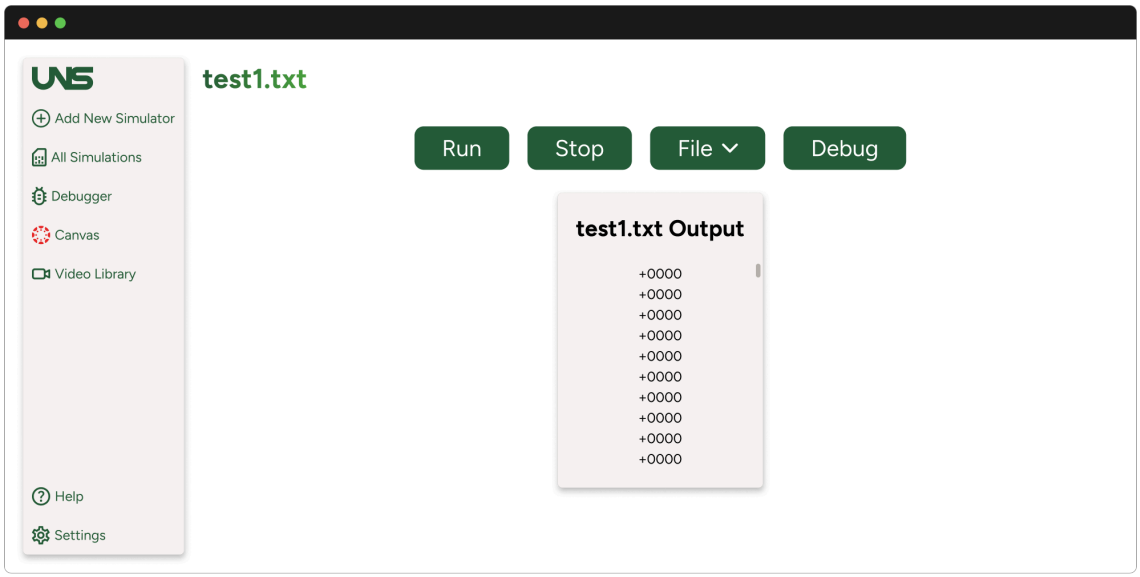
- Desktop app
- Mobile app
- Web app

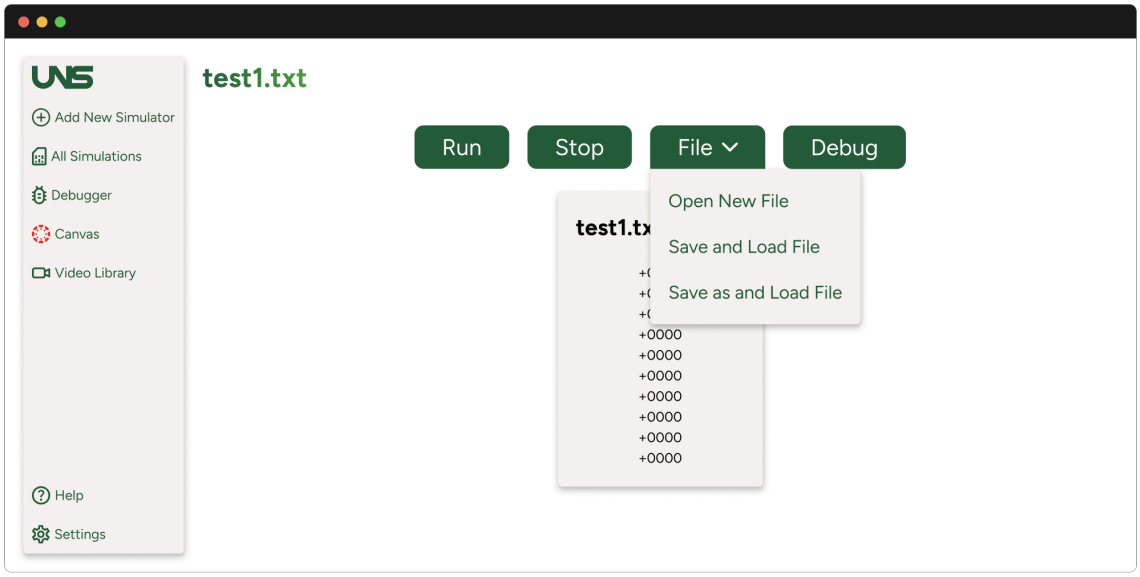
New UI Mockups

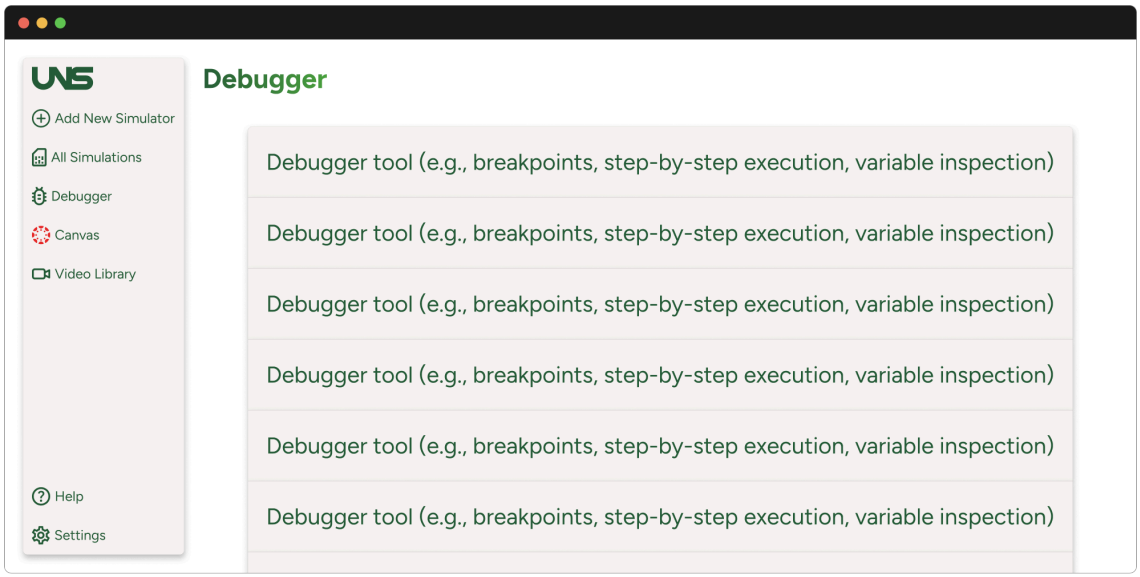
Below are UI mockups for the additional platforms and a redesign of the current program.

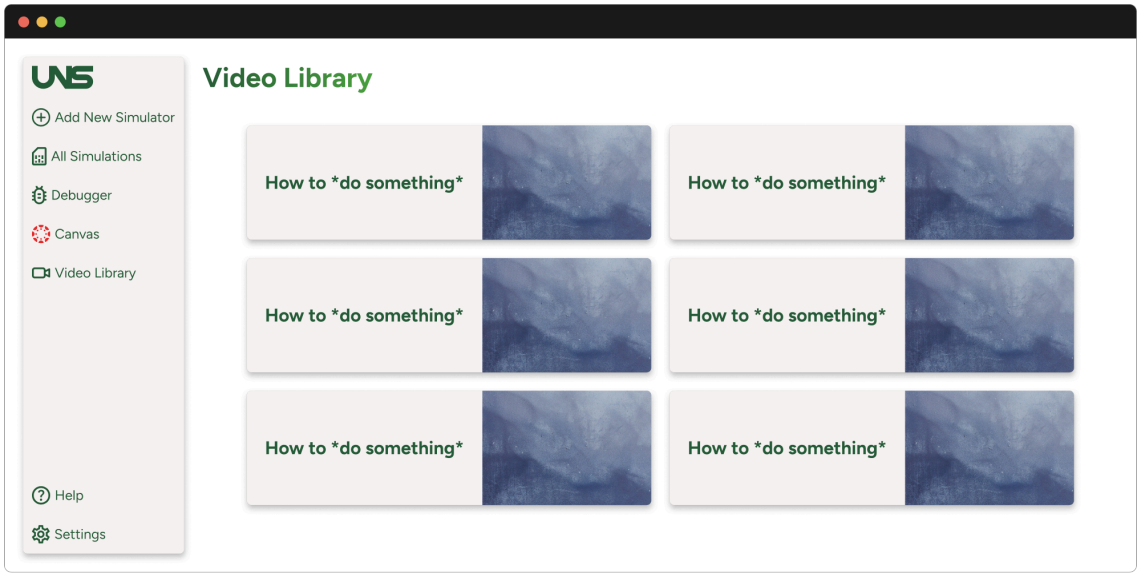
Desktop App

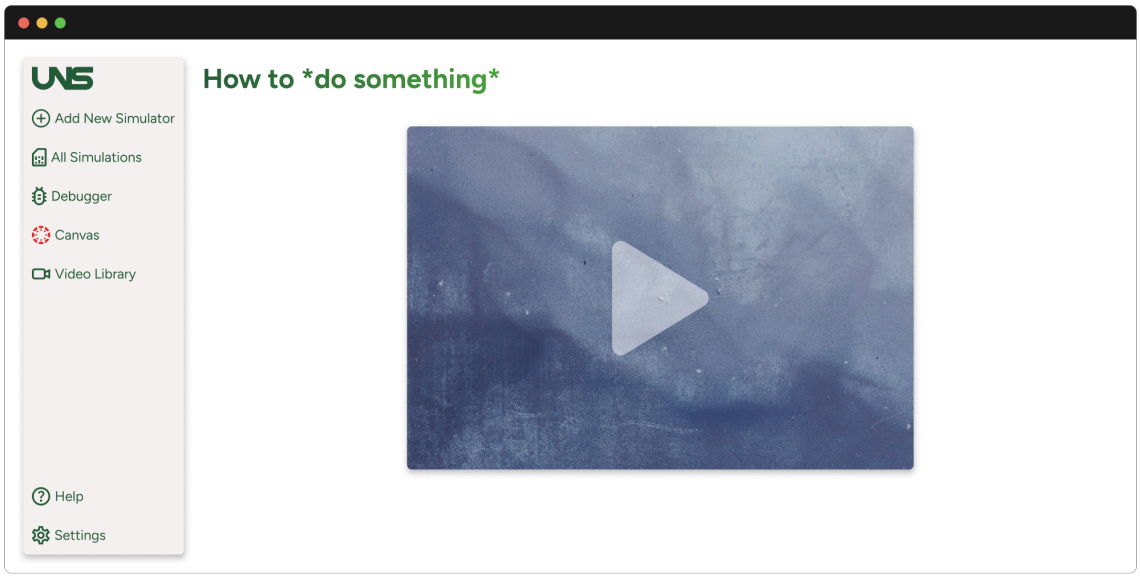


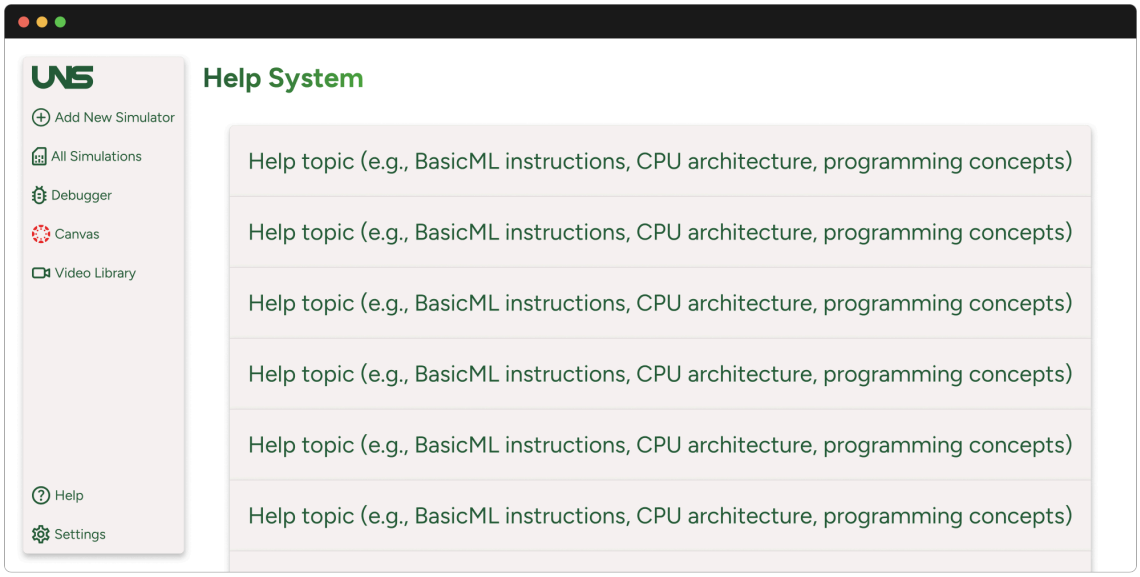

UNS
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
test1.txt
Run
Stop
File
Debug
test1.txt Output
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000

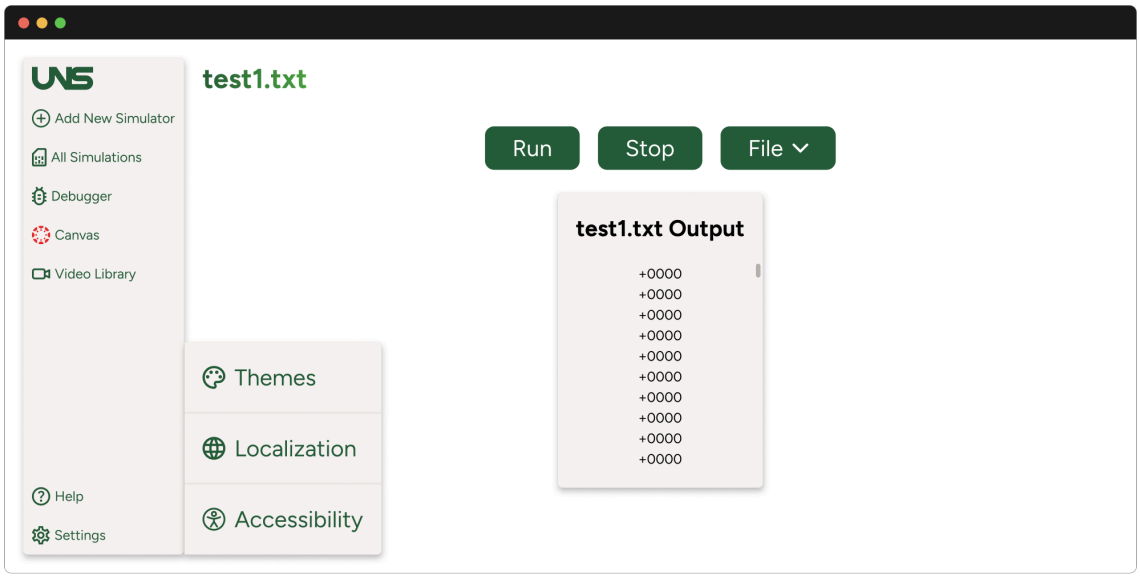

UNS
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
test1.txt
Run
Stop
File
Debug
Open New File
Save and Load File
Save as and Load File
+0000
+0000
+0000
+0000
+0000
+0000
+0000

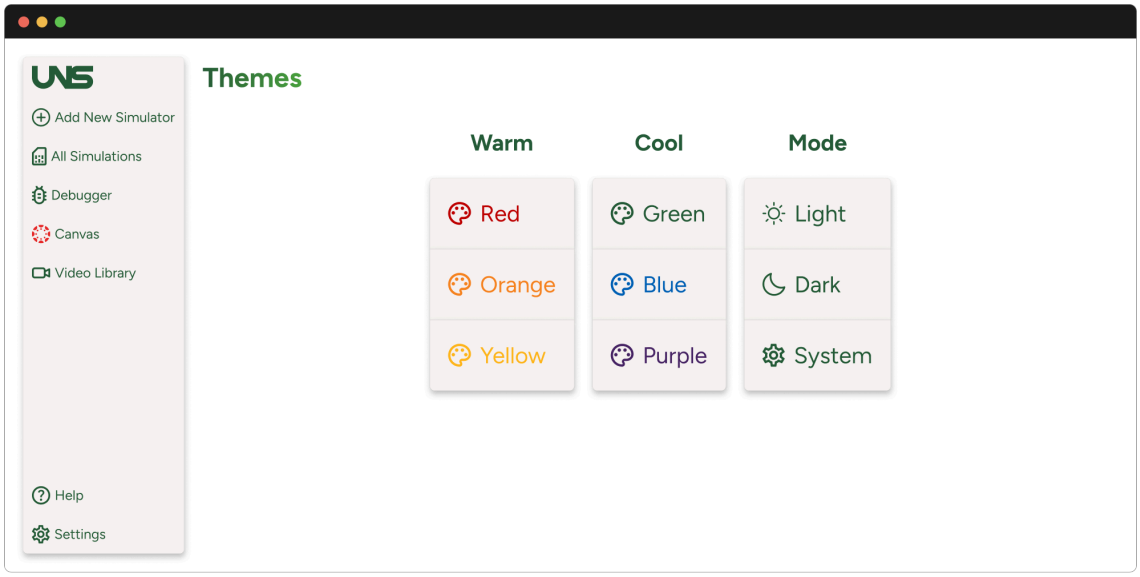

UNS
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Debugger
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)

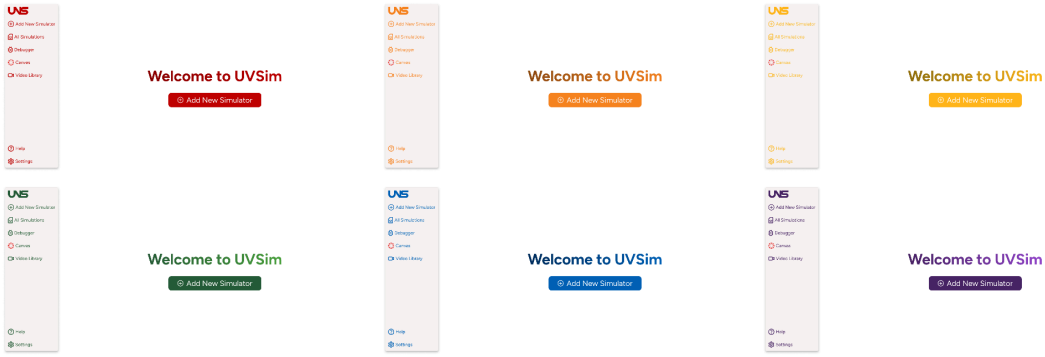

UNS
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Video Library
How to *do something*
How to *do something*
How to *do something*
How to *do something*
How to *do something*
How to *do something*

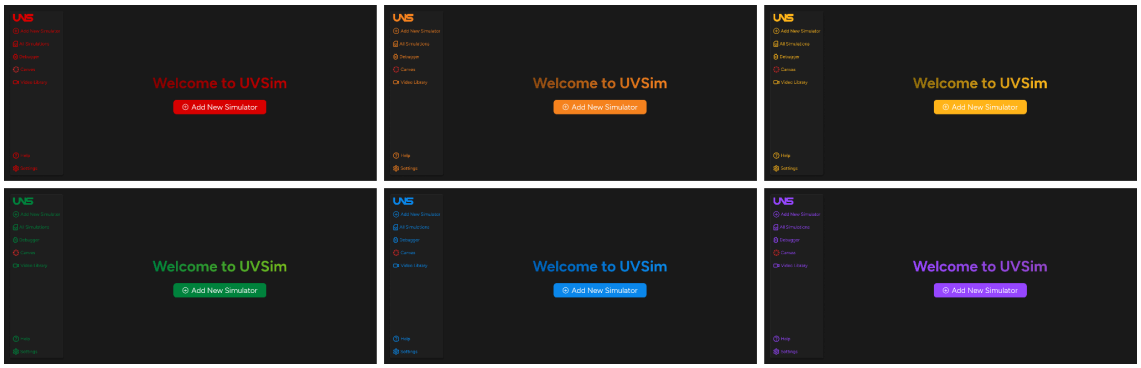

UNS
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
How to *do something*

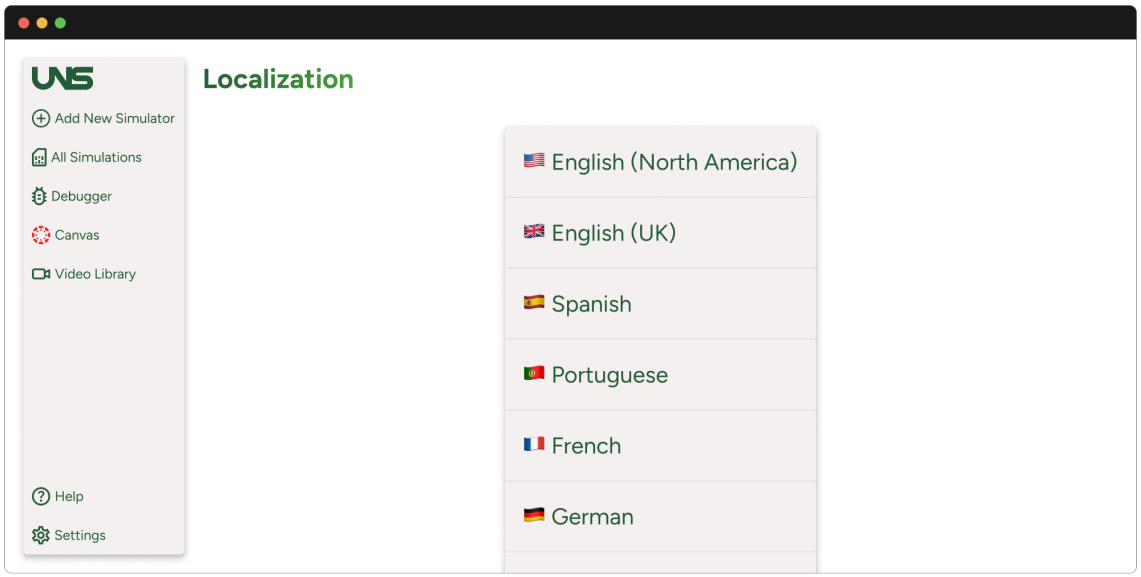

UNS
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Help System
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)

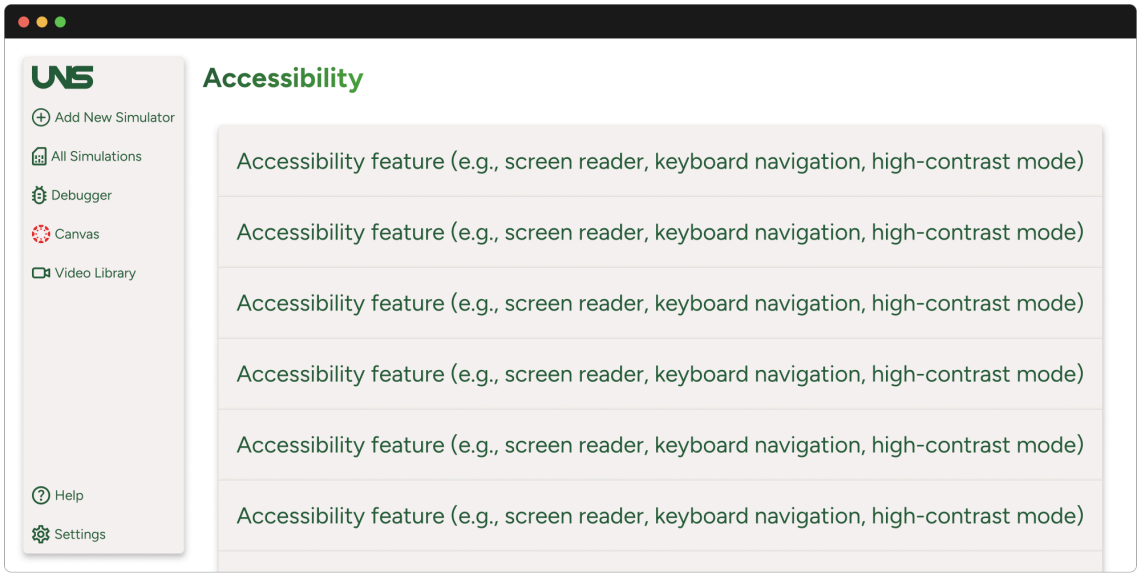

test1.txt
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Themes
Localization
Accessibility
Run
Stop
File
test1.txt Output
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000


Themes
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Warm
Cool
Mode
Red
Orange
Yellow
Green
Blue
Purple
Light
Dark
System


Welcome to UVSim
Add New Simulator


Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator


Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Localization
English (North America)
English (UK)
Spanish
Portuguese
French
German


Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Accessibility
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)

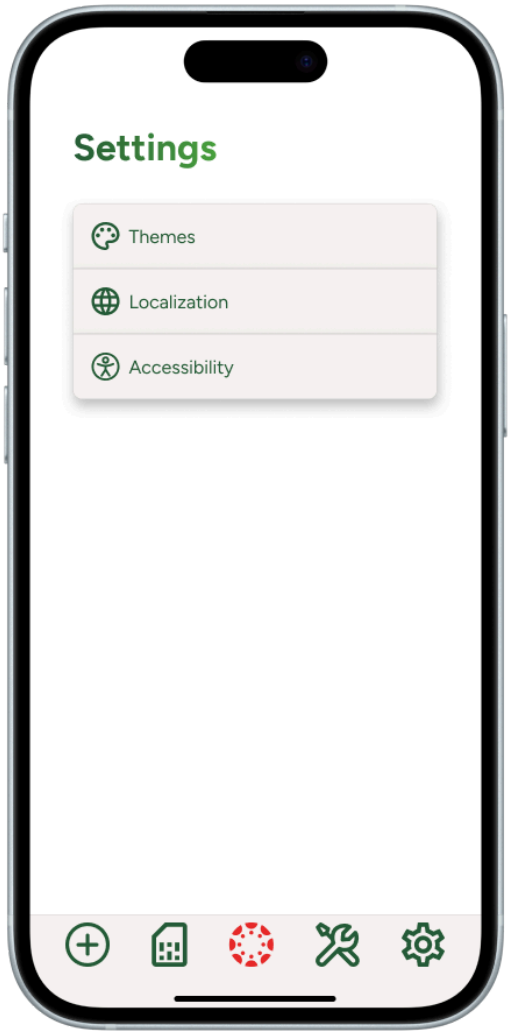
Mobile App

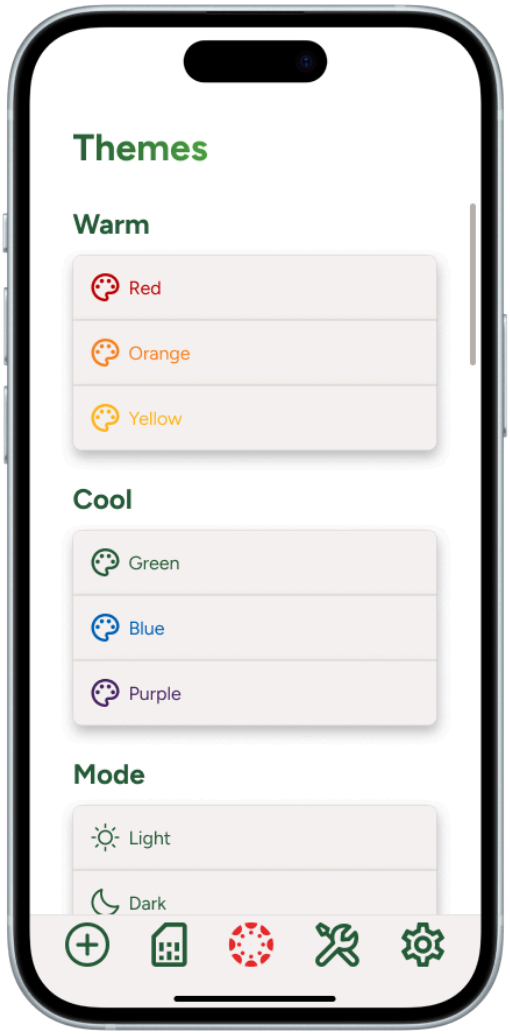

Welcome to UVSim
Add New Simulator
All Simulations
test1.txt
test2.txt
test3.txt
test4.txt
test5.txt
test6.txt
test7.txt
test8.txt
test9.txt
test10.txt
test11.txt
test12.txt
test13.txt
test14.txt


test1.txt
Run
Stop
File
Debug
test1.txt Output
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
test1.txt
Run
Stop
File
Debug
Open New File
Save and Load File
Save as and Load File
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000


Tools
Debugger
Video Library
Help System
Debugger
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)


Video Library
How to *do something*
How to *do something*
How to *do something*
How to *do something*
How to *do something*
How to *do something*
How to *do something*
How to *do something*
Help System
Help Topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help Topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help Topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help Topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help Topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help Topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help Topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help Topic (e.g., BasicML instructions, CPU architecture, programming concepts)


Settings
Themes
Localization
Accessibility

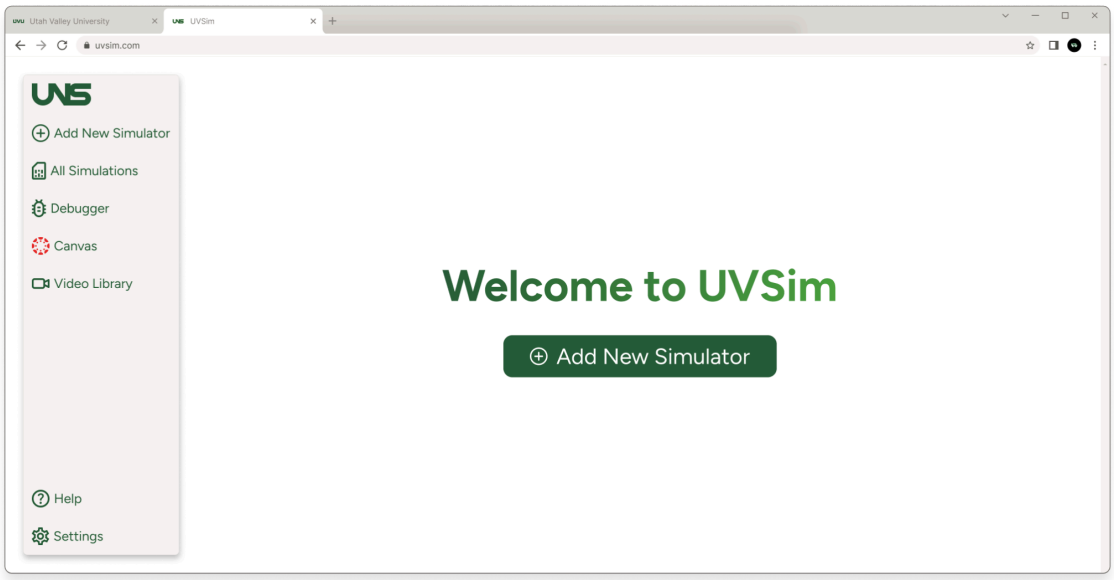

Themes
Warm
Red
Orange
Yellow
Cool
Green
Blue
Purple
Mode
Light
Dark

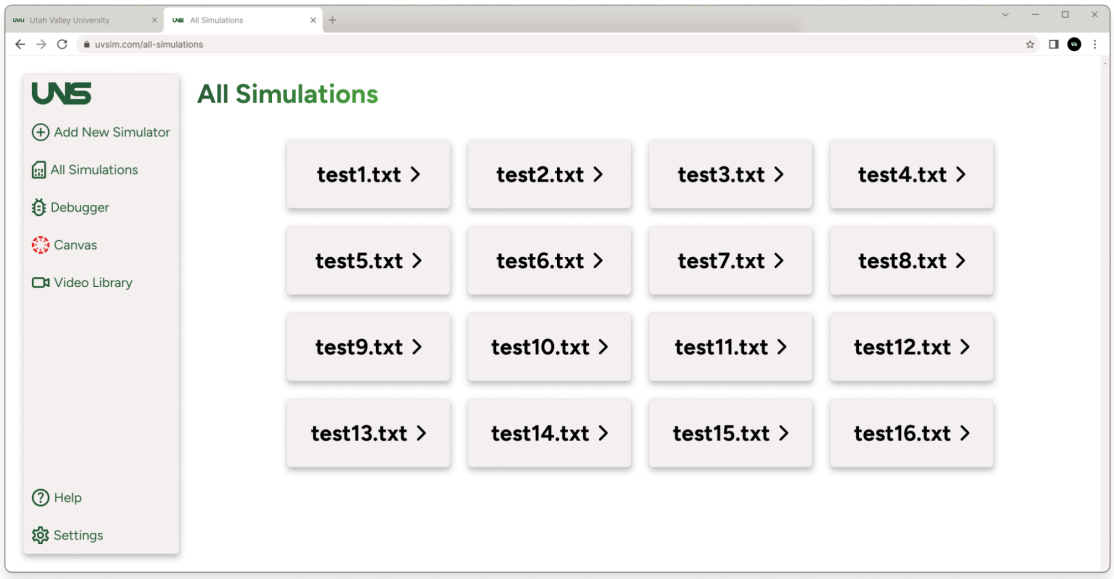

Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator

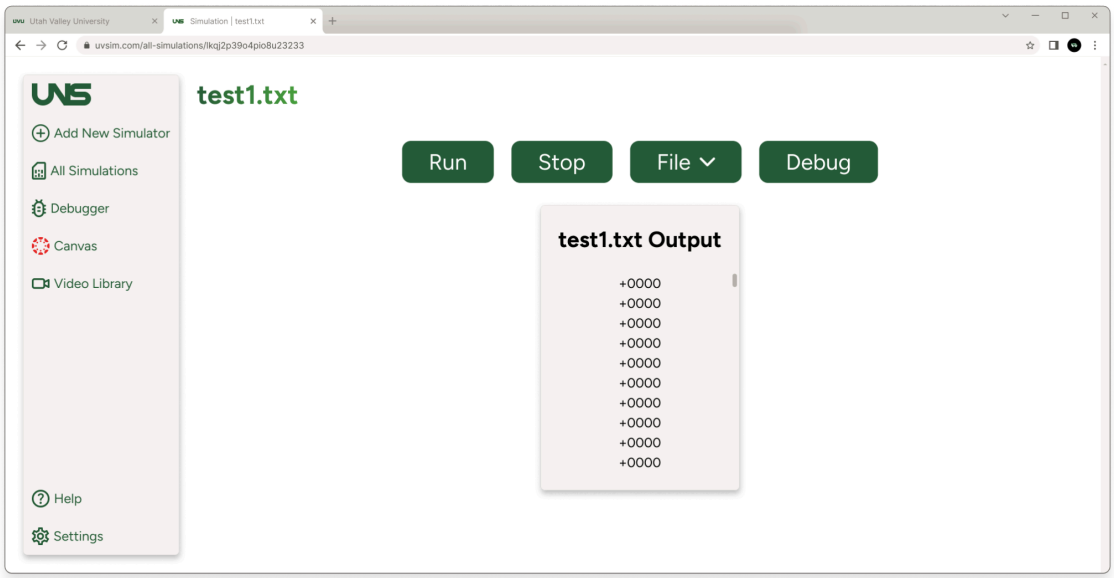

Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator
Welcome to UVSim
Add New Simulator

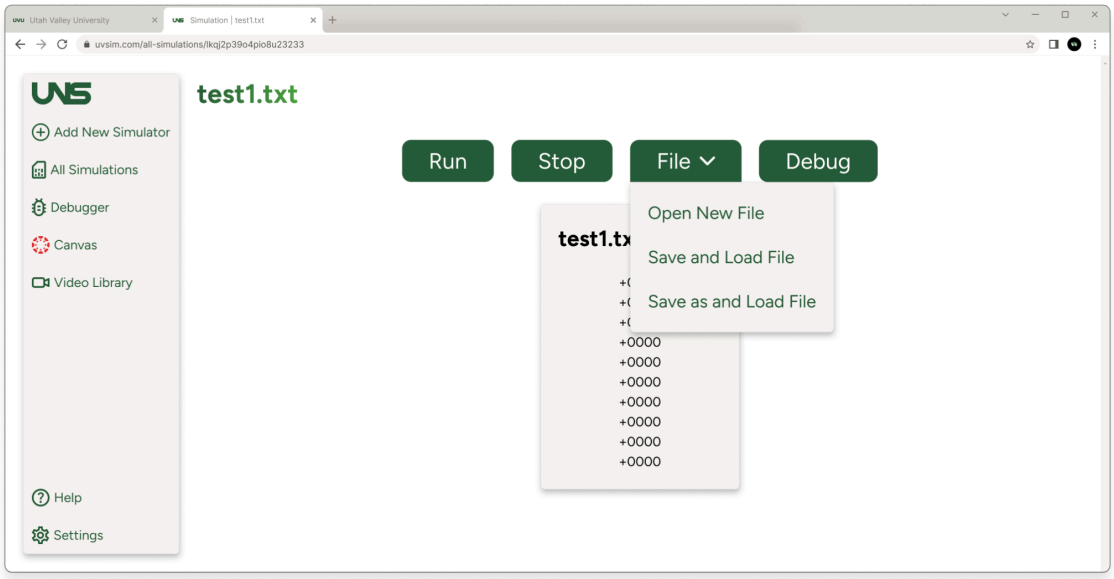
Web App

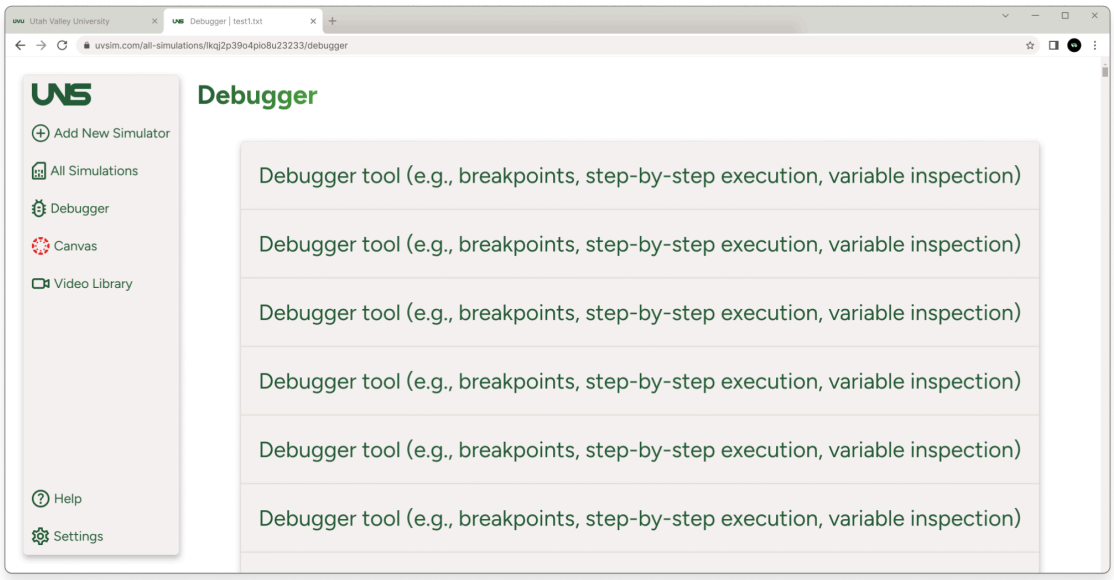
Note that the web app will have the same design and layout as the desktop app.

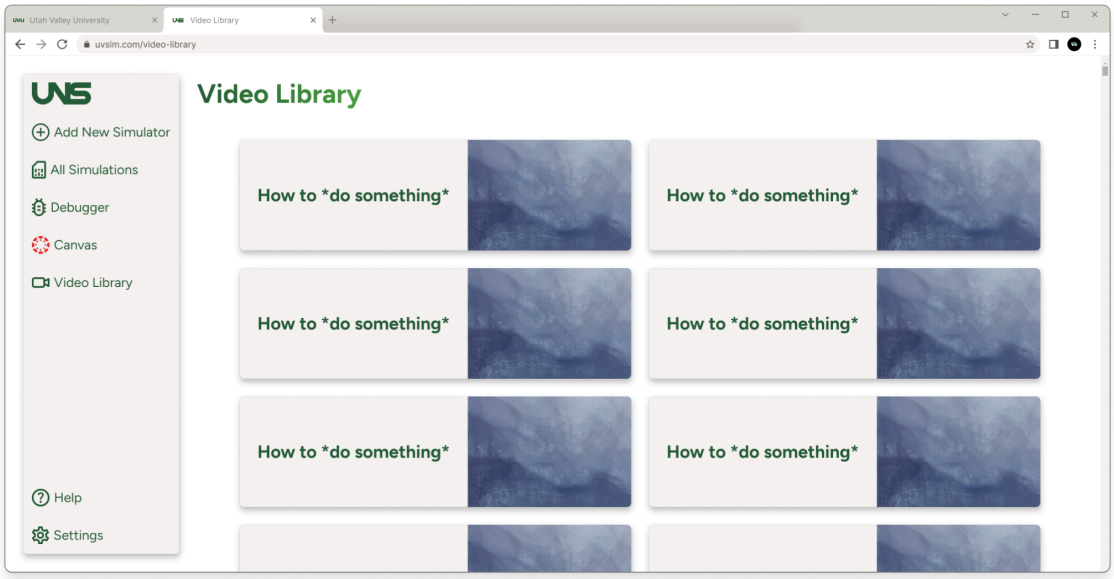

Utah Valley University
UVSim
uvsim.com
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Welcome to UVSim
Add New Simulator

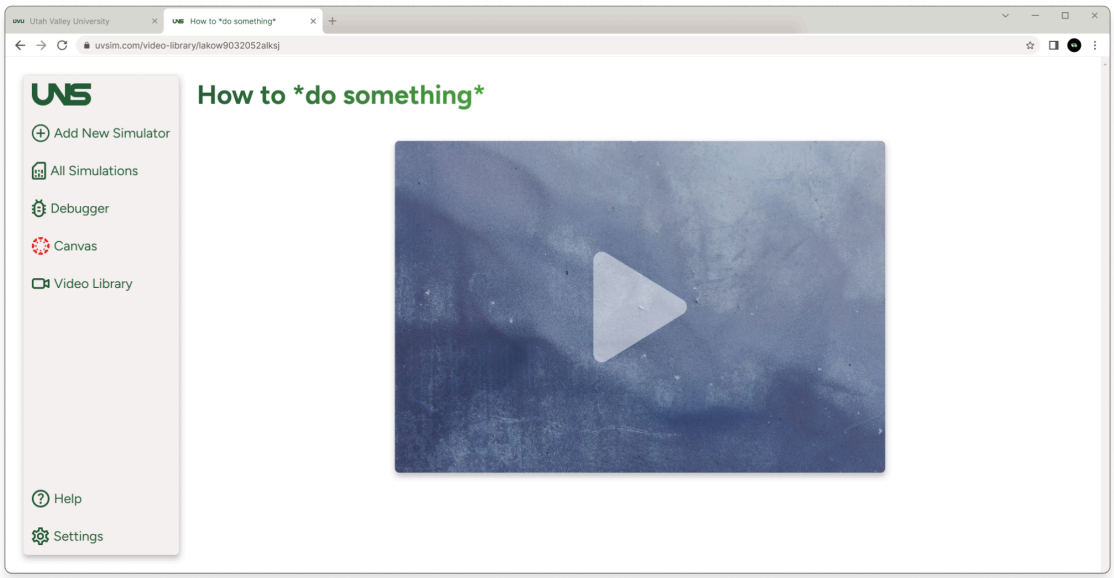

Utah Valley University
All Simulations
uvsim.com/all-simulations
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
All Simulations
test1.txt
test2.txt
test3.txt
test4.txt
test5.txt
test6.txt
test7.txt
test8.txt
test9.txt
test10.txt
test11.txt
test12.txt
test13.txt
test14.txt
test15.txt
test16.txt

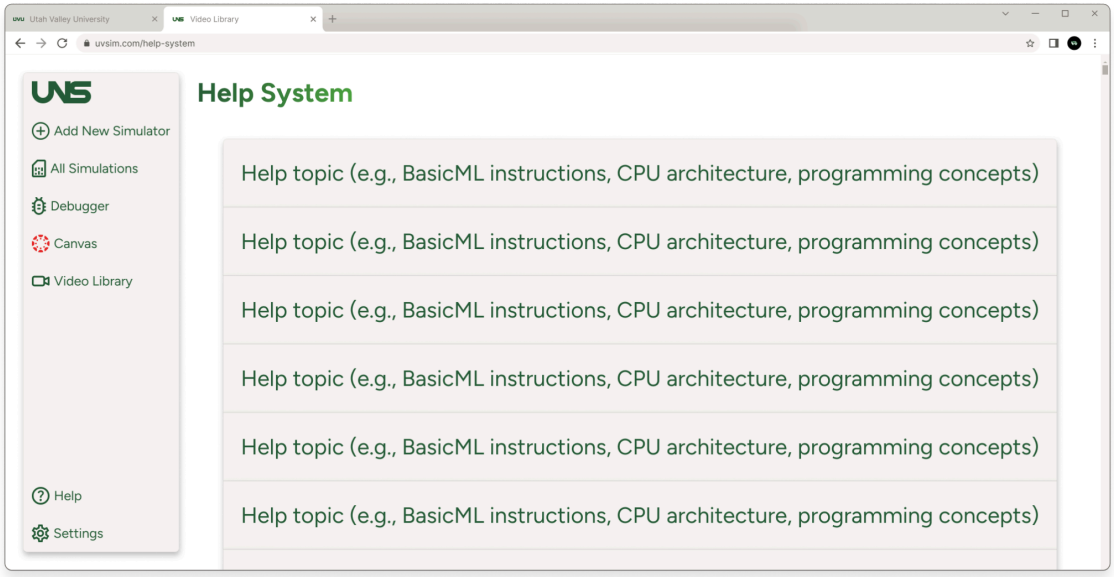

Utah Valley University
Simulation | test1.txt
uvsim.com/all-simulations/lkqj2p39o4pio8u23233
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
test1.txt
Run
Stop
File
Debug
test1.txt Output
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000

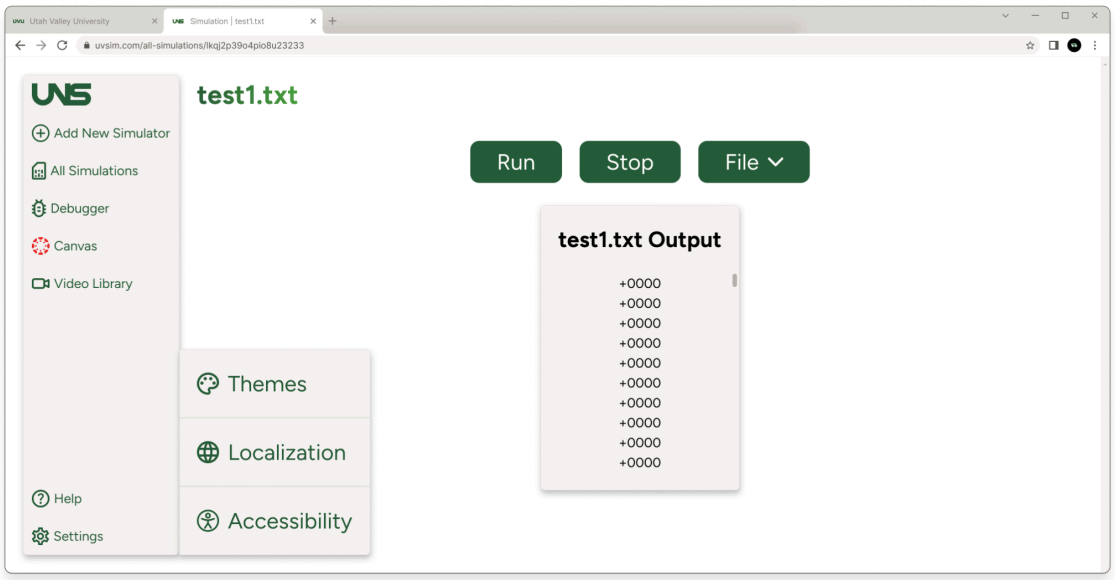

Utah Valley University
Simulation | test1.txt
uvsim.com/all-simulations/lkqj2p39o4pio8u23233
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
test1.txt
Run
Stop
File
Debug
Open New File
Save and Load File
Save as and Load File
+0000
+0000
+0000
+0000
+0000
+0000
+0000

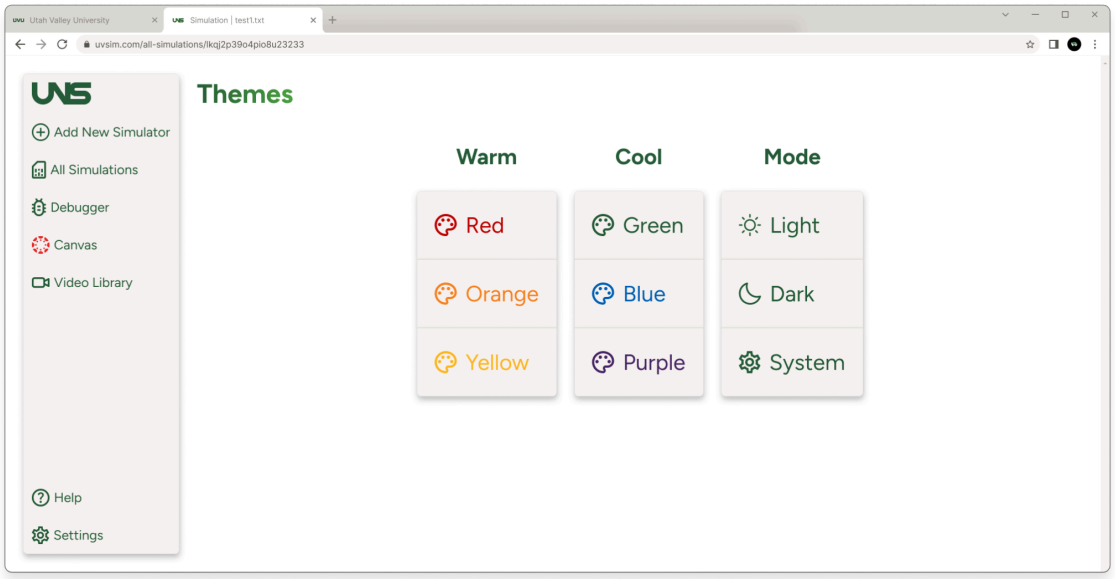

Utah Valley University
Debugger | test1.txt
uvsim.com/all-simulations/lkqj2p39o4pio8u23233/debugger
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Debugger
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)
Debugger tool (e.g., breakpoints, step-by-step execution, variable inspection)

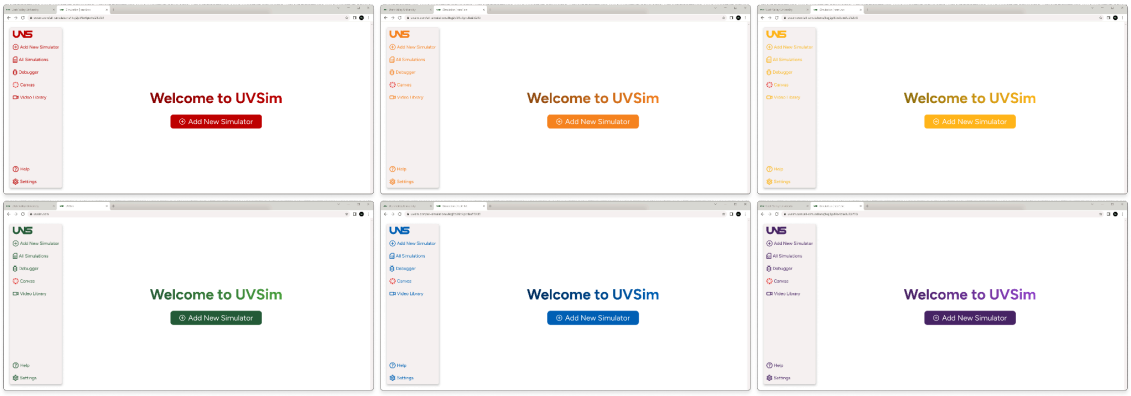

Utah Valley University
Video Library
uvsim.com/video-library
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Video Library
How to *do something*
How to *do something*
How to *do something*
How to *do something*
How to *do something*
How to *do something*

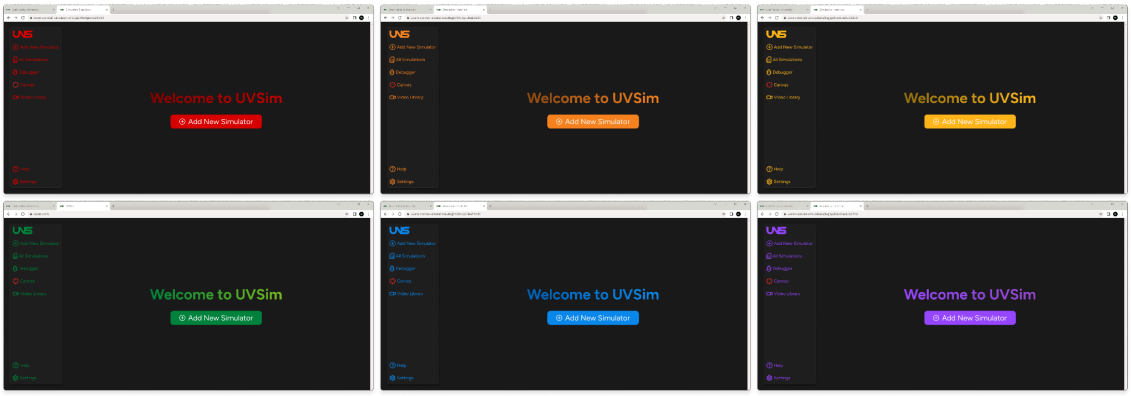

Utah Valley University
How to *do something*
uvsim.com/video-library/lakow9032052alksj
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
How to *do something*

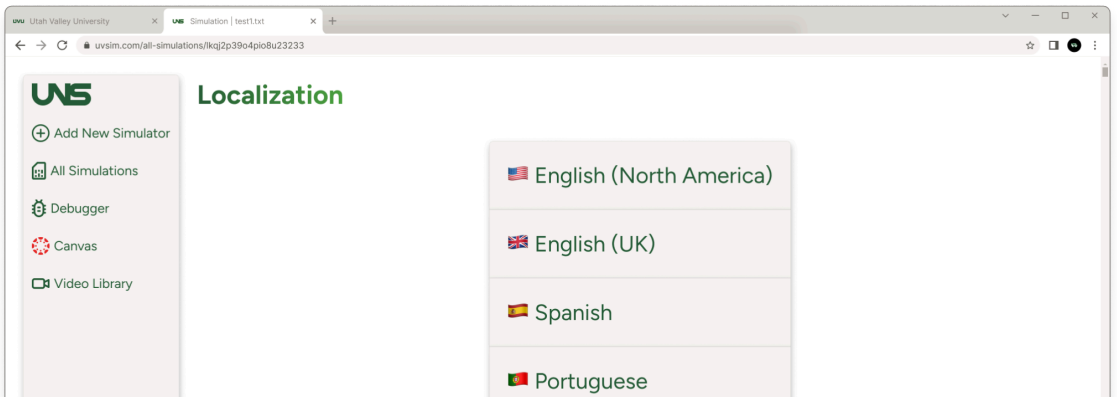

Utah Valley University
Video Library
uvsim.com/help-system
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Help System
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)
Help topic (e.g., BasicML instructions, CPU architecture, programming concepts)

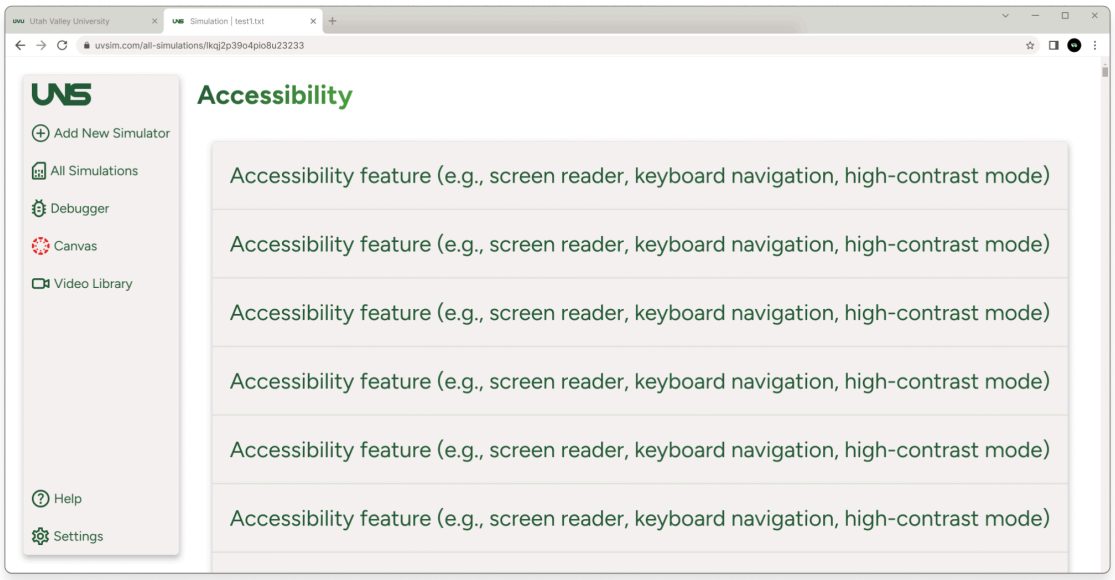

Utah Valley University
Simulation | test1.txt
uvsim.com/all-simulations/lkqj2p39o4pio8u23233
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
test1.txt
Run
Stop
File
test1.txt Output
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
+0000
Themes
Localization
Accessibility


Utah Valley University
Simulation | test1.txt
uvsim.com/all-simulations/lkqj2p39o4pio8u23233
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Themes
Warm
Cool
Mode
Red
Orange
Yellow
Green
Blue
Purple
Light
Dark
System


Welcome to UVSim
Add New Simulator


Welcome to UVSim
Add New Simulator


Localization
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
English (North America)
English (UK)
Spanish
Portuguese
French
German


Utah Valley University
Simulation | test1.txt
uvsim.com/all-simulations/lkqj2p39o4pio8u23233
Add New Simulator
All Simulations
Debugger
Canvas
Video Library
Help
Settings
Accessibility
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)
Accessibility feature (e.g., screen reader, keyboard navigation, high-contrast mode)