UVSim: A Python-Based Machine Language Simulator with GUI

Course: CS2450 - X01
Team Members: Collin Ross, Erich Zaugg, Tess Brian, Meaghan Barrett
Date: August 2025
Instructor: Professor Burtt

Table of Contents

Executive Summary / Introduction

UVSim is a Python-based application that simulates a simple virtual machine capable of interpreting and executing programs written in a low-level instruction set. The application features a graphical user interface (GUI) developed using tkinter, enabling users to load, edit, run, and manage machine language programs in an accessible environment.

The simulator supports both legacy (4-digit) and new (6-digit) formats, ensuring backward compatibility while encouraging modern best practices. A built-in file converter ensures seamless transitions between the two formats. Through the GUI, users can interact with memory, input/output operations, and directly manipulate program instructions — supporting a hands-on learning experience in systems-level programming and architecture concepts.

This project demonstrates not only fundamental understanding of CPU and memory simulation, file I/O, and GUI design, but also highlights clean modular programming with files like cpu.py, memory.py, convert.py, and uvsim.py, each with clear responsibilities. Additionally, a JSON-based configuration (colors.json) enables users to personalize the color scheme of the interface, adding an extra layer of usability.

Overall, UVSim serves as a teaching tool, a developer sandbox, and a launchpad for future system-level simulations.

Future Road Map

The UVSim project presents numerous opportunities for enhancement and expansion. As an educational tool and software simulator, future iterations can aim to improve functionality, user experience, educational alignment, and platform flexibility. The following roadmap outlines proposed directions for development if continued over the next 6–24 months.

I. Functional Enhancements

1. Instruction Stepping and Breakpoints
Introduce step-through execution and user-defined breakpoints to aid debugging and instruction flow analysis.

2. Real-Time Syntax Validation
Enhance the instruction editor with dynamic validation, offering immediate feedback on formatting and opcode errors.

3. Instruction Usage Analyzer
Generate post-execution analytics including opcode usage statistics, memory access patterns, and control flow diagrams.

II. User Interface and Usability

1. Dynamic Memory Visualization
Implement a more interactive grid to visually represent memory usage and instruction execution in real-time.

2. Enhanced Theme Management
Develop a visual theme editor for modifying color schemes directly through the GUI, linked to the existing colors.json configuration.

3. Resizable Panels and Layout Profiles
Allow users to rearrange the interface layout and save preferred panel arrangements for personalized workflows.

III. AI Integration

1. Instruction Summary Generator
Use AI or rules-based interpretation to produce plain-language descriptions of a given program’s logic and behavior.

2. Built-in Assistant / Chatbot
Incorporate an assistant trained on project documentation and opcode rules to assist users with common questions.

IV. Platform and Deployment Expansion

1. Web-Based UVSim
Port the simulator to a web-based environment to remove installation barriers and support browser-based learning.

2. Mobile Application (iOS/Android)
Develop a lightweight mobile version for code viewing, editing, and submission.

3. Cloud File Integration
Enable cloud-based file management using APIs for Google Drive or Dropbox, with support for version history and team collaboration.

V. Integration and Export Features

1. LMS Connectivity (Canvas, Blackboard)
Provide direct upload support for assignments, submissions, and grades into Learning Management Systems.

2. GitHub Integration
Allow students and users to version-control their work, submit assignments, or showcase projects directly from the application.

User Stories

User Story 1:

As an instructor, I want to provide CS students with a machine language simulation tool, so that they can experiment with low-level programming concepts without needing physical hardware.

User Story 2:

As a CS student, I want to run my BasicML program in the UVSim so I can understand how assembly-level instructions interact with CPU and memory.

Use Cases

Use Case 1: File Input by User

Actor: User
System: File handling and loader subsystem
Goal: Load a program file into memory for execution or editing
Steps:

1. Launch program.
2. Prompt user to select a file from any directory.
3. Receive file selection input.
4. Validate file existence and format (4-digit or 6-digit words).
5. Confirm file contains no more than 100 lines (old format) or 250 lines (new format).
6. Read file contents word by word (5 characters for old, 6 for new format).
7. Parse and store valid words into sequential memory addresses.
8. If invalid format or too many lines, raise error and re-prompt.
9. Display file contents in GUI for user inspection and editing.

Use Case 2: Execute BRANCH (Opcode 40 / 040 for new format)

Actor: Instruction execution unit
System: Program counter and control flow logic
Goal: Jump unconditionally to a specific memory address
Steps:

1. Parse opcode from instruction (2-digit or 3-digit opcode depending on format).
2. Identify operand (target address, supporting 2- or 3-digit addresses).
3. Verify target address within valid range (00-99 old, 000-249 new).
4. Set program counter to target address.
5. Continue execution from new memory location.

Use Case 3: Execute READ (Opcode 10 / 010 for new format)

Actor: Instruction execution unit
System: Input handling and memory writing logic
Goal: Store user input into specified memory location
Steps:

1. Parse opcode from instruction.
2. Identify target memory address from operand.
3. Prompt user for input.
4. Receive input from console.
5. Store input value in identified memory address.
6. Increment program counter.

Use Case 4: Execute HALT (Opcode 43 / 043 for new format)

Actor: Instruction execution unit
System: Program state and control logic
Goal: End program execution
Steps:

1. Parse opcode from instruction.
2. Set halted flag to True.
3. Output message to console showing halt location and accumulator value.
4. Stop instruction cycle.

Use Case 5: Execute BRANCHNEG (Opcode 41 / 041 for new format)

Actor: Instruction execution unit
System: Program counter and accumulator logic
Goal: Branch to a new address if accumulator is negative
Steps:

1. Parse opcode from instruction.
2. Identify operand (target address).
3. Check if accumulator < 0.
4. If true, set program counter to operand.
5. Else, increment program counter.

Use Case 6: Execute BRANCHZERO (Opcode 42 / 042 for new format)

Actor: Instruction execution unit
System: Program counter and accumulator logic
Goal: Branch to a new address if accumulator is zero
Steps:

1. Parse opcode from instruction.
2. Identify operand (target address).
3. Check if accumulator == 0.
4. If true, set program counter to operand.
5. Else, increment program counter.

Use Case 7: Execute LOAD (Opcode 20 / 020 for new format)

Actor: Instruction execution unit
System: Memory management and code processor
Goal: Successfully load a value into the accumulator
Steps:

1. Parse function code.

2. Identify target memory address from operand.
3. Fetch value from identified memory address.
4. Copy fetched value into accumulator register.
5. Increment program counter.

Use Case 8: Execute STORE (Opcode 21 / 021 for new format)

Actor: Instruction execution unit
System: Memory management subsystem
Goal: Store the value in the accumulator into memory
Steps:

1. Parse function code.
2. Identify target memory address from operand.
3. Copy value from accumulator to memory at identified address.
4. Increment program counter.

Use Case 9: Execute ADD (Opcode 30 / 030 for new format)

Actor: Instruction execution unit
System: Arithmetic logic unit (ALU)
Goal: Add a value from memory to the accumulator
Steps:

1. Parse function code.
2. Identify memory address from operand.
3. Fetch value from memory.
4. Add value to accumulator.
5. Store result in accumulator.
6. Increment program counter.

Use Case 10: Execute SUBTRACT (Opcode 31 / 031 for new format)

Actor: Instruction execution unit
System: Arithmetic logic unit (ALU)
Goal: Subtract a memory value from the accumulator
Steps:

1. Parse function code.
2. Identify memory address from operand.
3. Fetch value from memory.
4. Subtract value from accumulator.
5. Store result in accumulator.
6. Increment program counter.

Use Case 11: Execute DIVIDE (Opcode 32 / 032 for new format)

Actor: Instruction execution unit
System: Arithmetic logic unit (ALU)
Goal: Divide the accumulator by a value from memory
Steps:

1. Parse function code.
2. Identify memory address from operand.
3. Fetch value from memory.
4. If value ≠ 0, divide accumulator by value.
5. Store result in accumulator.
6. If value == 0, raise divide-by-zero error and halt.
7. Increment program counter unless halted.

Use Case 12: Execute MULTIPLY (Opcode 33 / 033 for new format)

Actor: Instruction execution unit
System: Arithmetic logic unit (ALU)
Goal: Multiply accumulator by a value from memory
Steps:

1. Parse function code.
2. Identify memory address from operand.
3. Fetch value from memory.
4. Multiply value by accumulator.
5. Store result in accumulator.
6. Increment program counter.

Use Case 13: Convert 4-Digit File to 6-Digit File Format

Actor: User
System: File conversion utility within the application
Goal: Convert an existing 4-digit word format file into the new 6-digit word format file for extended memory and functionality
Steps:

1. User opens the conversion tool from the application GUI or menu.
2. Prompt user to select a 4-digit format file from any directory.
3. Validate the selected file contains only valid 4-digit words and does not exceed 100 lines.
4. Read each line of the file sequentially.
5. For each line:
    - If the first two digits match a valid function opcode, convert using the 0XX0XX format (e.g., 1007 → 010007).

    - Otherwise, treat as numerical value and convert using the 00XXXX format (e.g., 5555 → 005555).
6. Reject the file if any invalid or ambiguous lines are detected.
7. Prompt user to save the new 6-digit file in a user-chosen directory with a new filename.
8. Save the converted file respecting the 250-line max and six-digit word format.
9. Confirm conversion success and optionally open the new file in the editor.

Use Case 14: Color Scheme Customization

Actor: User
Goal: Customize the color scheme of the application
Steps:

1. Launch the application.
2. Navigate to color settings (through GUI or a config file).
3. Select a primary and an off-color from a color picker or by entering RGB/Hex values.
4. Apply changes instantly or restart the app to see changes.
5. Ensure readability of text with the selected color scheme.

Use Case 15: Load and Save Files from Custom Directories

Actor: User
Goal: Load and save files from user-specified directories
Steps:

1. Launch the program.
2. Use the "Open" button to navigate and load files from any directory.
3. Edit the file as necessary.
4. Save the file to any directory or under a new name using the "Save" button.

Functional specifications

Functional:

1. The system shall display a ‘load program file’ button.

2. The system GUI shall exhibit a primary color, used as the main background color. The primary color shall default to UVU green (Hex# 4C721D).

3. The system GUI shall exhibit a secondary color, used for clickable buttons and text. The secondary color shall default to white (Hex# FFFFFF).

4. The system colors shall be user-configurable via a configuration file or in-app option without recompilation.

5. The system shall allow users to open a text file when the ‘load program file’ button is clicked. The text file shall be imported via a user-chosen directory.

6. The system shall raise an error if one or more words in the text file are not valid 4- or 6-digit signed integers.

7. The system shall load the contents of the chosen file into editable memory via the GUI, allowing user review before execution.

8. The system shall allow a user to make changes to their file inside the GUI including cut, copy, paste, add, delete, and edit operations.

9. The system GUI shall allow a user to save their file to a user-chosen directory with optional renaming.

10. The program shall begin operation at the first location in memory (000) when the ‘run program’ button is clicked.

11. The system’s memory access shall be restricted to valid address space (000–249).

12. The system shall display a ‘run program’ button, allowing an open file to be executed.

13. The system shall open a user-input popup when a READ command (010XXX) is encountered in the program.

14. The system shall read the contents of the input field in the input popup into memory when the user clicks the ‘submit’ button in the popup window.

15. The system’s execution shall support various 6-digit opcodes within the range 010–043:

15a. Opcode 010: Read a word from the keyboard into a specific location in memory

15b. Opcode 011: Write a word from a specific location in memory to screen

15c. Opcode 020: Load a word from a specific location in memory into the accumulator

15d. Opcode 021: Store a word from the accumulator into a specific location in memory

15e. Opcode 030: Add a word from a specific location in memory to the word in the accumulator

15f. Opcode 031: Subtract a word from a specific location in memory from the word in the accumulator

15g. Opcode 032: Divide the word in the accumulator by a word from a specific location in memory

15h. Opcode 033: Multiply a word from a specific location in memory to the word in the accumulator

15i. Opcode 040: Branch to a specific location in memory

15j. Opcode 041: Branch to a specific location in memory if the accumulator is negative

15k. Opcode 042: Branch to a specific location in memory if the accumulator is zero

15l. Opcode 043: HALT: stop the program

16. The system shall display an accumulator value, with the accumulator being a memory register with the same size and functions of the other memory registers.

17. The system shall reset the accumulator to 0 and memory to all zeros when a new file is loaded in.

18. The system shall end program execution when a HALT command (043XXX) is encountered.

19. The system shall display the contents of memory (up to 250 entries) in a scrollable, editable table format, confirming when a program is successfully loaded.

20. The system shall display output in a separate text area when the program executes a WRITE instruction (011XXX).

21. The system shall prevent the user from running the program if no file is loaded or if the file contains format errors.

22. The system shall allow the user to close the application in the GUI.

23. The system shall initialize all memory cells (000–249) and the accumulator to 0 when a new file is loaded in.

24. The system shall reject malformed instructions (such as: invalid characters, unsupported opcodes, or incorrect word length).

25. The system shall raise an error on division by zero.

26. The system's accumulator and register values exceeding ±999999 shall wrap using modulo 1,000,000.

27. The system shall display error messages coinciding with errors that might be encountered (ex. bad words in a file, missing words, trying to access out-of-range memory).

28. The system shall support multiple files open at once via GUI tabs or windows. 4-digit files shall be automatically converted to 6-digit format before editing or running.

29. The system shall only allow files with words of consistent size, either 4-digit or 6-digit words.

Non-functional:

1. The system shall be built using tiered architecture.

2. The system shall avoid crashes through input validation and exception handling.

3. The system shall make it clear to the user if the file chosen isn’t of the correct format (e.g., wrong length, illegal characters, too many lines).

4. The system shall be compatible with Python 3+.

5. The system shall be compatible with Windows and macOS.

6. The system shall execute the program and remain responsive (≤ 3 seconds for 250 instructions).

Class Diagram

```
+--------------------------------------------+
|              ColorConfig
+--------------------------------------------+
| - filename: str
| - primary: str
| - off: str
+--------------------------------------------+
| + __init__(filename="colors.json")
| + is_valid_hex(color: str): bool
| + load_colors(): void
+---------------------------------------------+
               ▲
               | used by
+--------------------------------------------------------+
|                   UVSimGUI
+--------------------------------------------------------+
| - root: Tk
| - sim: UVSim
| - load: Load_Program
| - colors: ColorConfig
| - memory_entries: list[Entry]
| - output_lines: list[str]
| - file_loaded: bool
| - run_button: Button
| - mem_canvas: Canvas
| - accumulator_var: StringVar
+----------------------------------------------------------+
| + __init__(root: Tk)
| + build_widgets(): void
| + load_file(): void
| + save_file(): void
| + save_memory_entry(index: int, value: str)
| + sync_entries_to_memory(): bool
| + run_program(): void
```

```
                    | + step_through_program(): void
                    | + show_read_popup(operand: int): int
                    | + update_memory_display(): void
                    | + update_output(): void
                    | + clear_output(): void
                    | + open_file_in_new_window(): void
                    +------------------------------------------------------------+
                    ▲                            ▲                              ▲
                    |                            |                              |
                uses|                          has-a                        uses|
                    |                            |                                 |
+-------------------------+   +----------------------------------+       +------------------------------------+
|          UVSim            |      Load_Program                      |           Convert
+-------------------------+    +------------------------------------+    +----------------------------------+
| - cpu: CPU                          | - memory: Memory                | + convert_to_int(word): int or False
| - memory: Memory       +------------------------------------------+   +--------------------------------+
+--------------------------+    | + __init__(): void
| + run(): void                  | + load_program(filename: str): void
+---------------------------+   +---------------------------------------------+

                 ▲
                 | has-a
                 |
+-------------------------------------------------------------+
|                              CPU
+-------------------------------------------------------------+
| - accumulator: int
| - program_counter: int
| - halted: bool
+---------------------------------------------------------------+
| + __init__()
| + execute(instruction: int, memory: Memory): void
+----------------------------------------------------------------+
                 ▲
               uses
                 |
+--------------------------------------------+
|                       Memory
+---------------------------------------------+
| - memory: list[MemoryRegister]
| - read_callback
| - waiting_for_input: bool
+----------------------------------------------+
| + __init__()
| + write(address: int, value: int): void
```

```
| + read(address: int): int
+-----------------------------------------------+
                 ▲
                has-a
                |
+---------------------------+
|    MemoryRegister
+----------------------------+
| - value: int
+---------------------------+
| + __init__(value=0)
| + set(value: int): void
| + get(): int
| + __repr__(): str
+--------------------------+
      ▲
      | called by
      |
+---------------+
|   main.py
+---------------+
| + main(): void
+----------------+
```

Class Descriptions

ColorConfig

- Loads GUI color themes from a colors.json file.

- Validates hex format.

- Allows customization of primary and off UI colors.

UVSimGUI

- Main GUI class that handles user interaction.

- Manages widgets, program loading, file handling, memory view, and execution.

- Coordinates between Load_Program, UVSim, and ColorConfig.

Load_Program

- Responsible for loading instruction files into memory.

- Parses .txt files and handles file format validation.

Convert

- Static class or helper for converting between formats.

- Changes strings into integers by striping the whitespace.

UVSim

- Simulator core.

- Ties together CPU and memory.

- Executes the program loaded in memory via CPU instructions.

CPU

- Core processor emulation.

- Handles opcodes, accumulator logic, branching, and program counter.

- Executes instructions by operating on Memory.

Memory

- Stores all instructions and values.

- Offers read and write operations with safety checks and callback support.

MemoryRegister

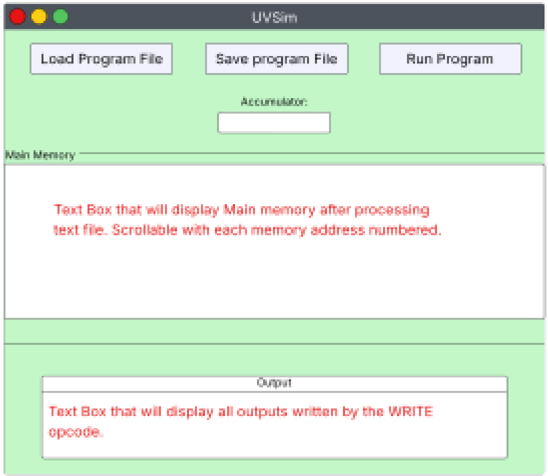
- Represents a single memory cell (integer value).

- Provides getter/setter and string conversion.

main.py

- Entry point for launching the GUI application.

GUI Wireframe



Text Box that will display Main memory after processing text file. Scrollable with each memory address numbered.

Text Box that will display all outputs written by the WRITE opcode.

Pop Up Window: Pops up to ask user for READ input.

Test Case Spreadsheet

UVSim Software Simulator User Manual

1. Introduction

UVSim is a Python-based software simulator designed to execute programs written in BasicML—a simplified educational machine language. This application provides a graphical user interface (GUI) using the tkinter library and supports both legacy and modern instruction formats. The simulator enables file loading, editing, execution, and debugging of machine code programs.

2. System Requirements

- **Python Version:** Python 3.x (Tkinter is typically included)
- **Operating System:** Cross-platform (Windows, macOS, Linux)
- **Libraries:** Only tkinter (included with Python)

3. Launching the Application

To launch UVSim:

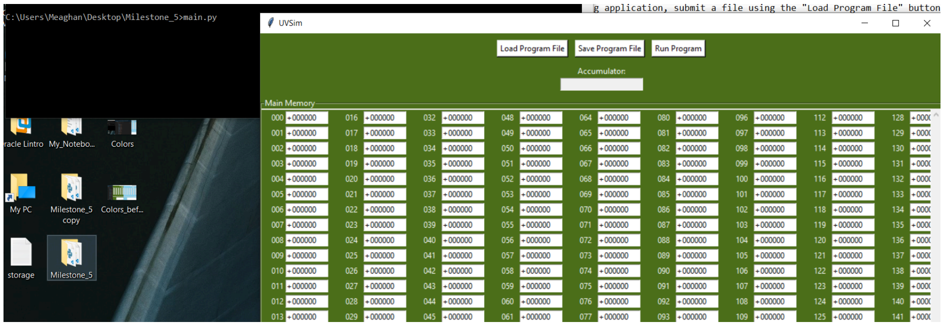
1. Open a terminal or command prompt.
2. Navigate to the directory where main.py is located.
3. Run the following command:

```
python3 main.py
```

4. Application Overview

Upon launching, the main interface presents several buttons and a program editor.

- **Load Program File** – Load a .txt file with BasicML instructions.
- **Run Program** – Execute the loaded instruction set.
- **Save Program File** – Save current instructions to a file.
- **Instruction Editor** – Edit instructions directly in the GUI.
- **Popup Windows** – Separate file tabs allow multiple files to be open at once.
- **Accumulator –** A separate register used for mathematical operations. *Note: the accumulator will be truncated by dropping the left-most digit if the value exceeds the 6-digit limit.*

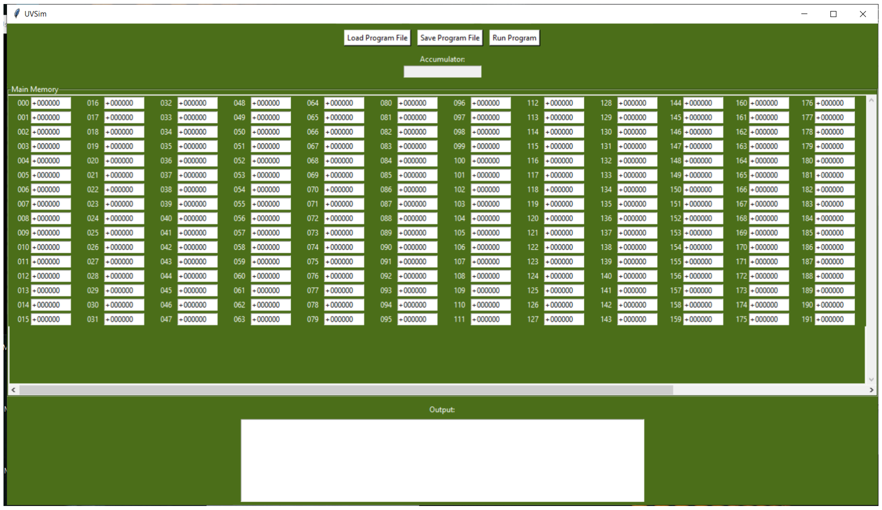


```
C:\Users\Meaghan\Desktop\Milestone_5>main.py
g application, submit a file using the "Load Program File" button
```

Accumulator:

Main Memory

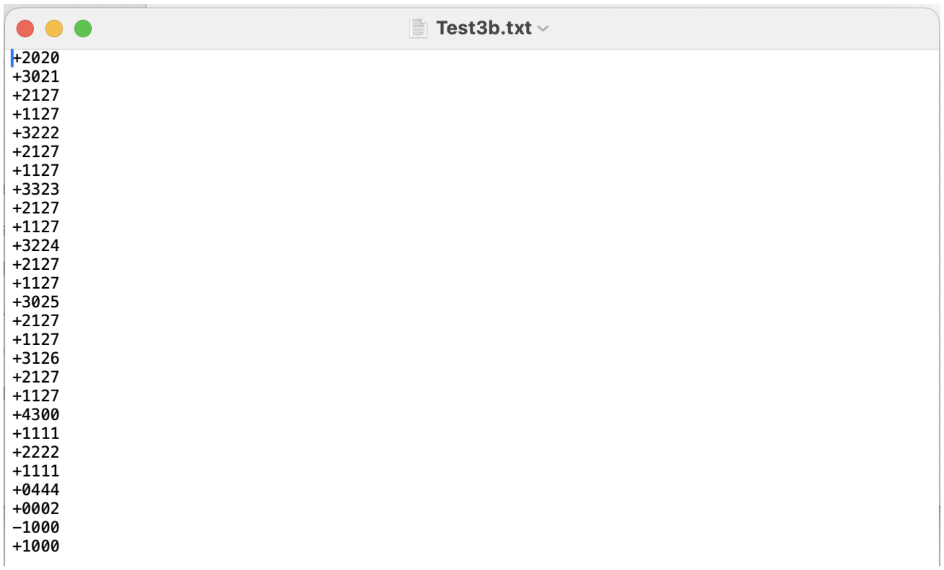
000	+000000	016	+000000	032	+000000	048	+000000	064	+000000	080	+000000	096	+000000	112	+000000	128	+000
001	+000000	017	+000000	033	+000000	049	+000000	065	+000000	081	+000000	097	+000000	113	+000000	129	+000
002	+000000	018	+000000	034	+000000	050	+000000	066	+000000	082	+000000	098	+000000	114	+000000	130	+000
003	+000000	019	+000000	035	+000000	051	+000000	067	+000000	083	+000000	099	+000000	115	+000000	131	+000
004	+000000	020	+000000	036	+000000	052	+000000	068	+000000	084	+000000	100	+000000	116	+000000	132	+000
005	+000000	021	+000000	037	+000000	053	+000000	069	+000000	085	+000000	101	+000000	117	+000000	133	+000
006	+000000	022	+000000	038	+000000	054	+000000	070	+000000	086	+000000	102	+000000	118	+000000	134	+000
007	+000000	023	+000000	039	+000000	055	+000000	071	+000000	087	+000000	103	+000000	119	+000000	135	+000
008	+000000	024	+000000	040	+000000	056	+000000	072	+000000	088	+000000	104	+000000	120	+000000	136	+000
009	+000000	025	+000000	041	+000000	057	+000000	073	+000000	089	+000000	105	+000000	121	+000000	137	+000
010	+000000	026	+000000	042	+000000	058	+000000	074	+000000	090	+000000	106	+000000	122	+000000	138	+000
011	+000000	027	+000000	043	+000000	059	+000000	075	+000000	091	+000000	107	+000000	123	+000000	139	+000
012	+000000	028	+000000	044	+000000	060	+000000	076	+000000	092	+000000	108	+000000	124	+000000	140	+000
013	+000000	029	+000000	045	+000000	061	+000000	077	+000000	093	+000000	109	+000000	125	+000000	141	+000



000	+000000	016	+000000	032	+000000	048	+000000	064	+000000	080	+000000	096	+000000	112	+000000	128	+000000	144	+000000	160	+000000	176	+000000
001	+000000	017	+000000	033	+000000	049	+000000	065	+000000	081	+000000	097	+000000	113	+000000	129	+000000	145	+000000	161	+000000	177	+000000
002	+000000	018	+000000	034	+000000	050	+000000	066	+000000	082	+000000	098	+000000	114	+000000	130	+000000	146	+000000	162	+000000	178	+000000
003	+000000	019	+000000	035	+000000	051	+000000	067	+000000	083	+000000	099	+000000	115	+000000	131	+000000	147	+000000	163	+000000	179	+000000
004	+000000	020	+000000	036	+000000	052	+000000	068	+000000	084	+000000	100	+000000	116	+000000	132	+000000	148	+000000	164	+000000	180	+000000
005	+000000	021	+000000	037	+000000	053	+000000	069	+000000	085	+000000	101	+000000	117	+000000	133	+000000	149	+000000	165	+000000	181	+000000
006	+000000	022	+000000	038	+000000	054	+000000	070	+000000	086	+000000	102	+000000	118	+000000	134	+000000	150	+000000	166	+000000	182	+000000
007	+000000	023	+000000	039	+000000	055	+000000	071	+000000	087	+000000	103	+000000	119	+000000	135	+000000	151	+000000	167	+000000	183	+000000
008	+000000	024	+000000	040	+000000	056	+000000	072	+000000	088	+000000	104	+000000	120	+000000	136	+000000	152	+000000	168	+000000	184	+000000
009	+000000	025	+000000	041	+000000	057	+000000	073	+000000	089	+000000	105	+000000	121	+000000	137	+000000	153	+000000	169	+000000	185	+000000
010	+000000	026	+000000	042	+000000	058	+000000	074	+000000	090	+000000	106	+000000	122	+000000	138	+000000	154	+000000	170	+000000	186	+000000
011	+000000	027	+000000	043	+000000	059	+000000	075	+000000	091	+000000	107	+000000	123	+000000	139	+000000	155	+000000	171	+000000	187	+000000
012	+000000	028	+000000	044	+000000	060	+000000	076	+000000	092	+000000	108	+000000	124	+000000	140	+000000	156	+000000	172	+000000	188	+000000
013	+000000	029	+000000	045	+000000	061	+000000	077	+000000	093	+000000	109	+000000	125	+000000	141	+000000	157	+000000	173	+000000	189	+000000
014	+000000	030	+000000	046	+000000	062	+000000	078	+000000	094	+000000	110	+000000	126	+000000	142	+000000	158	+000000	174	+000000	190	+000000
015	+000000	031	+000000	047	+000000	063	+000000	079	+000000	095	+000000	111	+000000	127	+000000	143	+000000	159	+000000	175	+000000	191	+000000

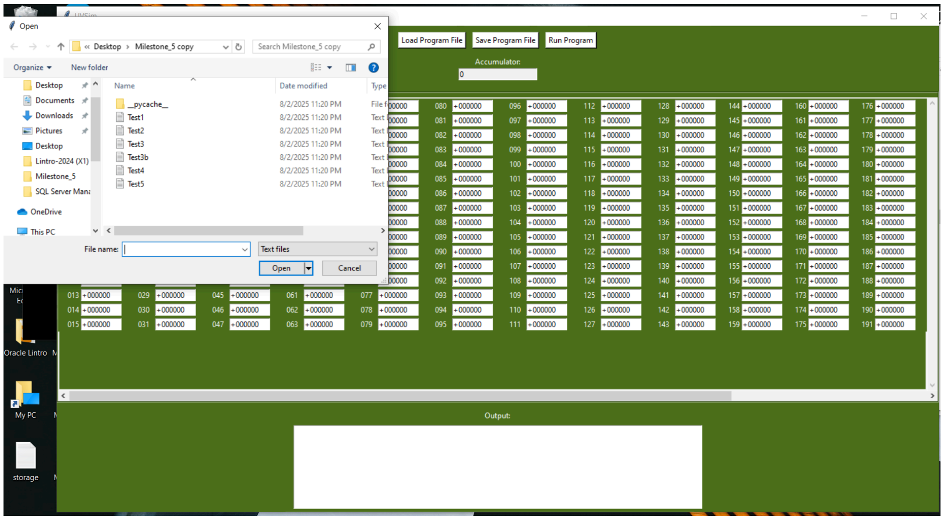
5. How to Use UVSim

5.1 Loading and Running Programs



```
+2020
+3021
+2127
+1127
+3222
+2127
+1127
+3323
+2127
+1127
+3224
+2127
+1127
+3025
+2127
+1127
+3126
+2127
+1127
+4300
+1111
+2222
+1111
+0444
+0002
-1000
+1000
```

1. Click **Load Program File** to import a .txt file.
2. Click **Run Program** to begin execution.
3. For input instructions (opcode READ), a pop-up will request user input. Enter the value and click **Submit**.
4. Upon HALT, the program stops and returns output.
5. To execute a new file, repeat the above process.



Accumulator: 0

Accumulator: -4444

000	+001115	016	+002222	032	+000000	048	+000000	064	+000000	080	+000000	096	+000000	112	+000000	128	+000000	144	+000000	160	+000000	176	+000000	192	+000000	208	+0
001	+004003	017	+003333	033	+000000	049	+000000	065	+000000	081	+000000	097	+000000	113	+000000	129	+000000	145	+000000	161	+000000	177	+000000	193	+000000	209	+0
002	+004300	018	+004444	034	+000000	050	+000000	066	+000000	082	+000000	098	+000000	114	+000000	130	+000000	146	+000000	162	+000000	178	+000000	194	+000000	210	+0
003	+001116	019	-004444	035	+000000	051	+000000	067	+000000	083	+000000	099	+000000	115	+000000	131	+000000	147	+000000	163	+000000	179	+000000	195	+000000	211	+0
004	+004206	020	+005555	036	+000000	052	+000000	068	+000000	084	+000000	100	+000000	116	+000000	132	+000000	148	+000000	164	+000000	180	+000000	196	+000000	212	+0
005	+004300	021	+000000	037	+000000	053	+000000	069	+000000	085	+000000	101	+000000	117	+000000	133	+000000	149	+000000	165	+000000	181	+000000	197	+000000	213	+0
006	+001117	022	+000000	038	+000000	054	+000000	070	+000000	086	+000000	102	+000000	118	+000000	134	+000000	150	+000000	166	+000000	182	+000000	198	+000000	214	+0
007	+002015	023	+000000	039	+000000	055	+000000	071	+000000	087	+000000	103	+000000	119	+000000	135	+000000	151	+000000	167	+000000	183	+000000	199	+000000	215	+0
008	+004205	024	+000000	040	+000000	056	+000000	072	+000000	088	+000000	104	+000000	120	+000000	136	+000000	152	+000000	168	+000000	184	+000000	200	+000000	216	+0
009	+001118	025	+000000	041	+000000	057	+000000	073	+000000	089	+000000	105	+000000	121	+000000	137	+000000	153	+000000	169	+000000	185	+000000	201	+000000	217	+0
010	+002019	026	+000000	042	+000000	058	+000000	074	+000000	090	+000000	106	+000000	122	+000000	138	+000000	154	+000000	170	+000000	186	+000000	202	+000000	218	+0
011	+004113	027	+000000	043	+000000	059	+000000	075	+000000	091	+000000	107	+000000	123	+000000	139	+000000	155	+000000	171	+000000	187	+000000	203	+000000	219	+0
012	+004300	028	+000000	044	+000000	060	+000000	076	+000000	092	+000000	108	+000000	124	+000000	140	+000000	156	+000000	172	+000000	188	+000000	204	+000000	220	+0
013	+001120	029	+000000	045	+000000	061	+000000	077	+000000	093	+000000	109	+000000	125	+000000	141	+000000	157	+000000	173	+000000	189	+000000	205	+000000	221	+0
014	+004300	030	+000000	046	+000000	062	+000000	078	+000000	094	+000000	110	+000000	126	+000000	142	+000000	158	+000000	174	+000000	190	+000000	206	+000000	222	+0
015	+001111	031	+000000	047	+000000	063	+000000	079	+000000	095	+000000	111	+000000	127	+000000	143	+000000	159	+000000	175	+000000	191	+000000	207	+000000	223	+0

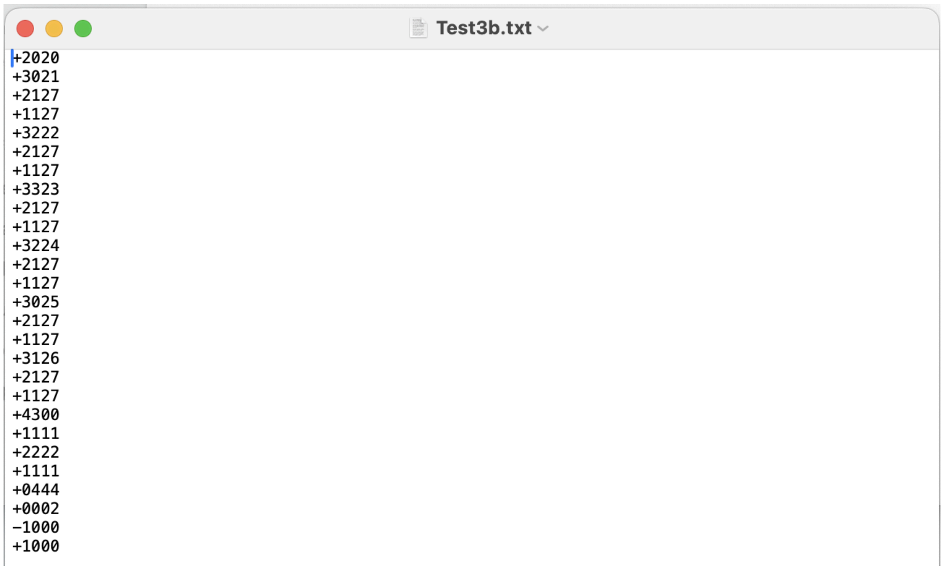
```
1111
2222
3333
4444
5555
HALT instruction encountered at memory [14]
```

5.2 Editing Instructions in GUI

- Add, delete, cut/copy/paste, and modify lines directly.
- File must not exceed the memory limit:
  - Legacy format: max 100 lines (00–99)
  - New format: max 249 lines (000–249)

6. File Format Specifications

6.1 BasicML File Structure



```
+2020
+3021
+2127
+1127
+3222
+2127
+1127
+3323
+2127
+1127
+3224
+2127
+1127
+3025
+2127
+1127
+3126
+2127
+1127
+4300
+1111
+2222
+1111
+0444
+0002
-1000
+1000
```

- Plain text (.txt) file
- One signed integer instruction per line
- Must include a HALT instruction:
    - Legacy: +4300
    - New: +043000

6.2 Formats

- **Legacy (4-digit)**: +1020, -3001, etc.
- **New (6-digit)**: +010035, +030249, etc.
- File must be **consistently formatted** with all lines of equal digit length.

7. BasicML Instruction Set

7.1 Legacy Opcodes (4-digit)

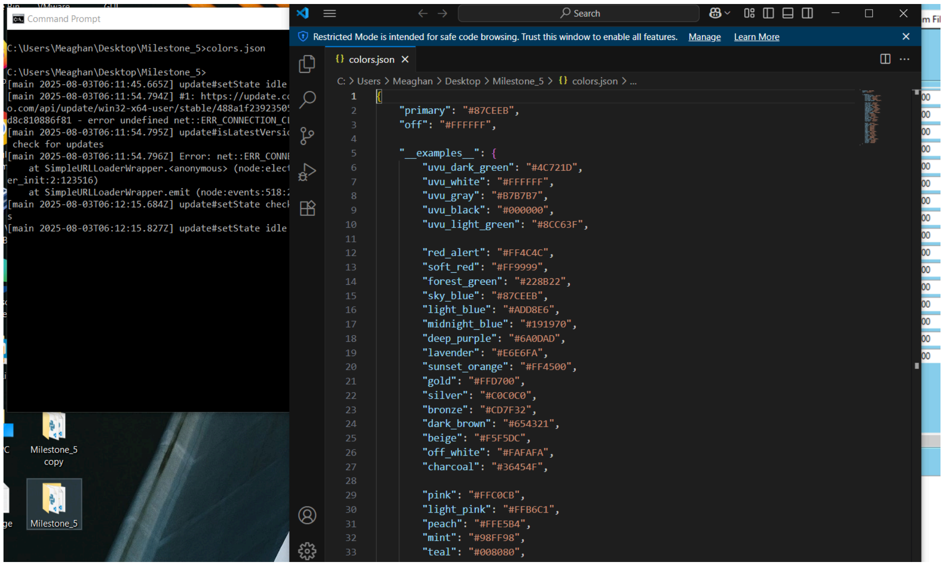
Opcode	Operation	Description
10	READ	Input a value to memory
11	WRITE	Output a value from memory
20	LOAD	Load memory value into accumulator
21	STORE	Store accumulator into memory
30	ADD	Add memory value to accumulator
31	SUBTRACT	Subtract memory value from accumulator
32	DIVIDE	Divide accumulator by memory value

33	MULTIPLY	Multiply accumulator by memory value
40	BRANCH	Jump to memory location
41	BRANCHNEG	Jump if accumulator is negative
42	BRANCHZERO	Jump if accumulator is zero
43	HALT	End program execution

7.2 New Opcodes (6-digit)

Opcode	Operation
010	READ
011	WRITE
020	LOAD
021	STORE
030	ADD
031	SUBTRACT
032	DIVIDE
033	MULTIPLY
040	BRANCH
041	BRANCHNEG
042	BRANCHZERO
043	HALT

8. Customizing the Interface Colors



```
C:\Users\Meaghan\Desktop\Milestone_5>colors.json

C:\Users\Meaghan\Desktop\Milestone_5>
[main 2025-08-03T06:11:45.665Z] update#setState idle
[main 2025-08-03T06:11:54.794Z] #1: https://update.co
o.com/api/update/win32-x64-user/stable/488a1f2392350
d8c810886f81 - error undefined net::ERR_CONNECTION_CL
[main 2025-08-03T06:11:54.795Z] update#isLatestVersio
 check for updates
[main 2025-08-03T06:11:54.796Z] Error: net::ERR_CONNE
    at SimpleURLLoaderWrapper.<anonymous> (node:elect
er_init:2:123516)
    at SimpleURLLoaderWrapper.emit (node:events:518:
[main 2025-08-03T06:12:15.684Z] update#setState check
s
[main 2025-08-03T06:12:15.827Z] update#setState idle
```

Restricted Mode is intended for safe code browsing. Trust this window to enable all features.

```json
{
    "primary": "#87CEEB",
    "off": "#FFFFFF",

    "__examples__": {
        "uvu_dark_green": "#4C721D",
        "uvu_white": "#FFFFFF",
        "uvu_gray": "#B7B7B7",
        "uvu_black": "#000000",
        "uvu_light_green": "#8CC63F",

        "red_alert": "#FF4C4C",
        "soft_red": "#FF9999",
        "forest_green": "#228B22",
        "sky_blue": "#87CEEB",
        "light_blue": "#ADD8E6",
        "midnight_blue": "#191970",
        "deep_purple": "#6A0DAD",
        "lavender": "#E6E6FA",
        "sunset_orange": "#FF4500",
        "gold": "#FFD700",
        "silver": "#C0C0C0",
        "bronze": "#CD7F32",
        "dark_brown": "#654321",
        "beige": "#F5F5DC",
        "off_white": "#FAFAFA",
        "charcoal": "#36454F",

        "pink": "#FFC0CB",
        "light_pink": "#FFB6C1",
        "peach": "#FFE5B4",
        "mint": "#98FF98",
        "teal": "#008080",
```

The program's color scheme can be customized by editing the colors.json file:

1. Navigate to the file location and open colors.json.
2. Edit the primary and off color hex codes:

```
{
        "primary": "#4C721D",
        "off": "#FFFFFF"
}
```

3. Hex code format must start with # followed by six characters (0–9, A–F).
4. Save changes and re-launch the program.
5. If errors appear, ensure the hex codes are valid.

```
{
    "primary": "#4C721D",
    "off": "#FFFFFF",

    "__examples__": {
        "uvu_dark_green": "#4C72
        "uvu_white": "#FFFFFF",
        "uvu_gray": "#B7B7B7",
        "uvu_black": "#000000",
        "uvu_light_green": "#8C

        "red_alert": "#FF4C4C",
        "soft_red": "#FF9999",
        "forest_green": "#228B22
        "sky_blue": "#87CEEB",
        "light_blue": "#ADD8E6",
        "midnight_blue": "#19197
        "deep_purple": "#6A0DAD"
        "lavender": "#E6E6FA",
        "sunset_orange": "#FF45
        "gold": "#FFD700",
        "silver": "#C0C0C0",
        "bronze": "#CD7F32",
        "dark_brown": "#654321",
        "beige": "#F5F5DC",
        "off_white": "#FAFAFA",
        "charcoal": "#36454F",

        "pink": "#FFC0CB",
        "light_pink": "#FFB6C1",
        "peach": "#FFE5B4",
        "mint": "#98FF98",
        "teal": "#008080",
        "cyan": "#00FFFF",
```

9. File Conversion and Compatibility

- Legacy 4-digit files are automatically converted to 6-digit instructions internally.
- The simulator supports up to **249** memory addresses for 6-digit files.
- Avoid mixing instruction formats within a single file.

10. Error Handling and Debugging

If errors occur:

- The application halts and reports the issue.
- A detailed error log is saved in a .txt file.
- Example issues:
  - Invalid opcode
  - Improper formatting
  - Instruction exceeding memory limit

11. Example Instruction Breakdown

4-Digit Format

- +1235
    - 12 → Opcode
    - 35 → Memory Location

6-Digit Format

- +010035
    - 010 → Opcode
    - 035 → Memory Location

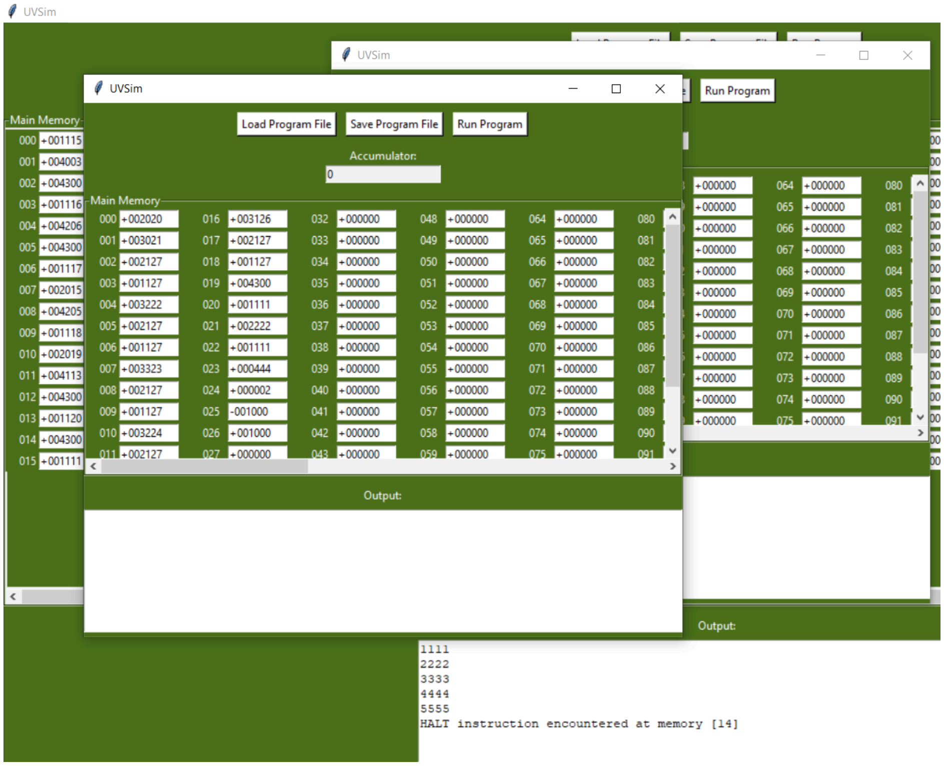
12. Testing Instructions

To test your code:

1. Create a .txt file with valid instructions.
2. Load it in UVSim.
3. If no errors occur, the code executes.

13. Notes

- You can load multiple program files simultaneously via GUI tabs.
- Only **one program** can be executed at a time.



Main Memory

000	+001115
001	+004003
002	+004300
003	+001116
004	+004206
005	+004300
006	+001117
007	+002015
008	+004205
009	+001118
010	+002019
011	+004113
012	+004300
013	+001120
014	+004300
015	+001111

Accumulator: 0

Main Memory

000	+002020	016	+003126	032	+000000	048	+000000	064	+000000	080	
001	+003021	017	+002127	033	+000000	049	+000000	065	+000000	081	
002	+002127	018	+001127	034	+000000	050	+000000	066	+000000	082	
003	+001127	019	+004300	035	+000000	051	+000000	067	+000000	083	
004	+003222	020	+001111	036	+000000	052	+000000	068	+000000	084	
005	+002127	021	+002222	037	+000000	053	+000000	069	+000000	085	
006	+001127	022	+001111	038	+000000	054	+000000	070	+000000	086	
007	+003323	023	+000444	039	+000000	055	+000000	071	+000000	087	
008	+002127	024	+000002	040	+000000	056	+000000	072	+000000	088	
009	+001127	025	-001000	041	+000000	057	+000000	073	+000000	089	
010	+003224	026	+001000	042	+000000	058	+000000	074	+000000	090	
011	+002127	027	+000000	043	+000000	059	+000000	075	+000000	091	

Output:

```
1111
2222
3333
4444
5555
HALT instruction encountered at memory [14]
```

14. Support

For bug reports or feature requests, please submit issues to the GitHub Repository.