Class: UVSim

Purpose:

This class creates a BasicML graphical interface for users to enter opcodes and execute them. There are buttons for load program, execute, load from file, and help.

Attributes:

- `memory`: A 100 word memory list.
- `accumulator`: Integer for current accumulator value.
- `accumulator_label`: A Tkinter Label widget to show the current value of the accumulator.
- `memory_label`: A Tkinter Label widget for memory display.
- `memory_listbox`: A Tkinter Listbox widget to display memory contents.
- `instruction_counter`: Integer for current instruction pointer.
- `instruction_label`: A Tkinter Label to display instruction input prompts.
- `instruction_input`: A Tkinter ScrolledText widget for user instruction entry.
- `load_button`: A Tkinter Button widget to load instructions into memory.
- `execute_button`: A Tkinter Button widget to execute loaded instructions.
- `file_button`: A Tkinter Button widget to load instructions from a file.
- `status_frame`: A Tkinter Frame widget for accumulator and instruction counter display.
- `instruction_counter_label`: A Tkinter Label widget to show the current instruction counter.
- `output_label`: A Tkinter Label widget for output display.
- `output_text`: A Tkinter ScrolledText widget for output messages.
- `help_button`: A Tkinter Button widget that opens a help dialog.

Methods:

`__init__(self, root)`

Purpose: Initializes the UVSim GUI.

Class: `UVSim`

Purpose: This class takes BasicML opcodes and executes them through file input or user input.

Attributes:

- `memory`: A 100 word memory list.
- `accumulator`: Holds integers for operations.
- `program_counter`: Keeps track of the current instruction being executed.
- `running`: Boolean to control program execution.

Methods:

`__init__()`

Purpose: Initializes memory, accumulator, program counter, and execution state.

Parameters: None.

Post-conditions: Initializes memory, sets `accumulator` and `program_counter` to zero, and sets `running` to `True`.

`load_program_from_file(filename)`

Purpose: Reads a program file and loads it into memory.

Parameters: `filename` (str): Name of the file containing BasicML instructions.

Return Value: `True` if the program loads successfully. `False` if an error occurs.

Pre-conditions: File must exist and contain valid instructions.

Post-conditions: Memory is filled with instructions, up to a maximum of 100.

`get_input()`

Purpose: Handles user input, allowing for easy overriding in tests.

Return Value: Integer input from the user.

Pre-conditions: User must provide a valid integer input.

Post-conditions: Returns a valid integer.

`execute()`

Purpose: Executes the loaded program based on given BasicML instructions.

Pre-conditions: Program must be loaded into memory.

Post-conditions: Executes instructions until a halt command is encountered or an error occurs.

Instruction Set:

- `10XX`: Read input into memory location `XX`.
- `11XX`: Write value from memory location `XX`.
- `20XX`: Load value from memory location `XX` into the accumulator.
- `21XX`: Store accumulator value into memory location `XX`.
- `30XX`: Add value from memory location `XX` to the accumulator.
- `31XX`: Subtract value from memory location `XX` from the accumulator.
- `32XX`: Divide accumulator by value from memory location `XX`.
- `33XX`: Multiply accumulator by value from memory location `XX`.
- `40XX`: Unconditional branch to memory location `XX`.
- `41XX`: Branch to `XX` if accumulator is negative.
- `42XX`: Branch to `XX` if accumulator is zero.
- `43XX`: Halt execution.

Class: `TestUVSim`

Purpose: This class checks for correct output from `UVSim` through unit tests.

Test Cases:

`test_read_valid_input()`

Purpose: Tests reading valid user input into memory.

Setup: Stores a read instruction (`1005`) in memory.

Execution: Mocks user input to simulate reading `1234`.

Expected Result: Memory at index 5 stores `1234`.

`test_read_invalid_input()`

Purpose: Tests handling of invalid input.

Setup: Stores a read instruction (`1005`) in memory.

Execution: Simulates non-numeric input (`'abc'`).

Expected Result: Raises a `ValueError`.

`test_write_output()`

Purpose: Tests writing memory value to output.

Setup: Stores `5678` in memory at index `10`.

Execution: Runs the write instruction (`1110`).

Expected Result: Output displays `5678`.

`test_addition_valid()`

Purpose: Tests accumulator addition.

Setup: Accumulator contains `10`, memory at index 5 contains `15`.

Execution: Runs add instruction (`3005`).

Expected Result: Accumulator holds `25`.

`test_divide_by_zero()`

Purpose: Tests division by zero error handling.

Setup: Accumulator contains `10`, memory at index 5 contains `0`.

Execution: Runs divide instruction (`3205`).

Expected Result: Prints "Error: Division by zero".

`test_branch_valid()`

Purpose: Tests unconditional branch instruction.

Setup: Stores branch instruction (`4020`) in memory.

Execution: Runs `execute()`.

Expected Result: `program_counter` is `20`.

`test_halt_execution()`

Purpose: Tests program halt instruction.

Setup: Stores halt instruction (`4300`) in memory.

Execution: Runs `execute()`.

Expected Result: Prints "*** Program terminated normally ***".