**The Unbin Visual Interpreter — Project Plan**

1. Overview.
2. The Unbin programming language was invented to be simple enough to have a parser understandable to a novice Python programmer. And to be aid in learning to trace the execution of a simple programming language.
3. The Unbin Visual Interpreter (UVI) is an interpreter of the Unbin programming language that visually portrays the interpreting of the Unbin program line-by-line, as it is executed.
4. The Unbin language is specified in Appendix A. One or more sample programs are shown in Appendix B.
5. The UVI has at least four windows:
   * One showing the program being interpreted
   * One showing the variables currently stored in memory
   * One graphically depicting the dependency of variables upon others
   * One showing the final output of the program
6. Mockups.

File: escape\_velocity.unb

— □ X

**File**

<<Pane graphically displaying dependencies>>

<<Input file pane>>

(This pane displays one or more trees, showing the dependencies of variables upon other variables. Nodes defined using single terms or unary operators are displayed as having one child, while nodes defined using binary operators are displayed as having two children.)

<<Memory pane>>

<<Pane displaying text output of script>>

1. Stretch goals.
2. Warn the user about trivial assignments (e.g., "x = x"), and re-assignments where the earlier value is not used before the follow-on assignment (e.g., "y = x" followed immediately by "y = 2 \* x").
3. Support the definition of dynamic variable assignment in the Unbin language. For example, the statement

y ~ 3 \* x

would not only assign the value of 3 \* x to the variable y, but would also repeat such an assignment any time the value in x is changed.

1. Support the ability to show the state of execution at an intermediate point in the script, and vary that intermediate point at will. (User interface details TBD.)
2. Implementation notes.
3. The Unbin programming language will be interpreted via a Python script.
4. The visual interpreter will be implemented using Python's Tkinter package.
5. The Graphical Display pane may be implemented using Python's turtle package.
6. Issues.
7. Identifier re-use. Unbin allows for an identifier (variable name) to be used multiple times in a script. This should not interfere with any of the other supported features, including dynamic variable assignment, if supported.
8. Timeline.
9. Unit #2: Project plan (this document).
10. Unit #3: Basic (command-line) Unbin interpreter, followed by the visual interpreter implementation.
11. Unit #4: Possibly implement one or more stretch goals.

**Appendix A: Language Specification for the Unbin Programming Language**

The name "Unbin" comes from the fact that the operators are **UN**ary or **BIN**ary.

1. Overview.
   * Unbin is a simple line-oriented programming language. Each non-blank line (as defined below) is an assignment of a value to a variable. The value is the value of the expression, and there are three such types of expressions:
     + single term (using either a floating point constant or a single variable)
       - **Form:** id = t
       - **Examples:** "x = 7", "gravity = 32.0", or "width = height".
     + values from expressions involving a single unary operator
       - **Form:** id = op1 t
       - **Examples:** "root = sqrt 25.0"
     + values from expressions involving a single binary operator
       - **Form:** id = tLeft op2 tRight
       - **Example:** "c = a + b"

where

* + - id is an identifier;
    - t, tLeft, and tRight are single terms (such as x, height, or 7.0);
    - op1 is a unary operator;
    - op2 is a binary operator.

1. Files. By convention, Unbin filenames end with the suffix "unb", as in "escape\_velocity.unb".
2. Whitespace
   * Text from a pound sign until the end of the current line are considered to be comments.
   * Lines that have only white space once comments are removed are considered to be "blank", and have no semantic impact.

* There must be white space on each side of each operator.

1. Supported numeric types.
   * All numeric values in Unbin are floats (i.e., floating-point numbers).
2. Supported unary operators.
   * Unbin supports only one unary operator, "sqrt", which returns the square root of its argument.
3. Supported binary operators.
   * Unbin supports four binary operators: +, -, \*, /. They each have their standard function.
4. Unsupported functionality.
   * Unbin does not support any loop functionality, function definitions, or class definitions.
5. Running an Unbin script.
   * Unbin scripts are run by an Unbin interpreter. If there is a command-line interpreter called unbin.py, then the command to run an Unbin script called escape\_velocity.unb would be:

unbin.py escape\_velocity.unb

* + The Unbin interpreter evaluates every line of the script in sequence.
  + Per-line output for blank lines. For each blank line of the script, the interpreter prints

linenum: <empty>

where linenum is the line number of the script, with numbering starting at 1.

**Note:** Per-line output is suppressed by default, and only shown in verbose mode.

* + Per-line output for non-blank lines. For each non-blank line of the script, the interpreter prints

linenum: id = value

where "id" is the identifier of the variable assigned to on line #linenum, and "value" is the value of the expression to the right of the equality operator.

**Note:** Per-line output is suppressed by default, and only shown in verbose mode.

* + Output of the script's final result. After the last line of the script has been evaluated and the per-line results printed, the Unbin interpreter prints one final line.
  + If there are any assignments in the script, then the final line printed is

Final result = value

where "value" is the value of the last assignment.

* + If there are no assignments in the script, then the final line printed is

No final result

**Appendix B: Sample Unbin script(s)**

1. Example script: h\_at\_t\_eq\_3.unb

# After a ball is thrown upward at t = 0 (seconds),

# from an initial height of 5 feet, and an upwards velocity of 40 (feet/sec),

# find the height of a ball at t = 3 (seconds).

#

# Use the formula h(t) = (1/2) \* g \* t\*\*2 + v\_0 \* t + h\_0

h\_0 = 5 # feet

v\_0 = 50 # feet/sec

g = -32 # gravity (negative = downward)

t = 3

half\_g = g / 2

t\_squared = t \* t

h\_acc = half\_g \* t\_squared

h\_vel = v\_0 \* t

h\_acc\_vel = h\_acc + h\_vel

h\_at\_t\_eq\_3 = h\_acc\_vel + h\_0

Output of sample script h\_at\_t\_eq\_3.unb

1: <empty>

2: <empty>

3: <empty>

4: <empty>

5: <empty>

6: h\_0 = 5.000000

7: v\_0 = 50.000000

8: g = -32.000000

9: t = 3.000000

10: <empty>

11: half\_g = -16.000000

12: t\_squared = 9.000000

13: h\_acc = -144.000000

14: <empty>

15: h\_vel = 150.000000

16: <empty>

17: h\_acc\_vel = 6.000000

18: h\_at\_t\_eq\_3 = 11.000000

Final result = 11.000000