Mike Hahn – hahnbytes@gmail.com

Sooperzone

Soopertree and Soopertags are new programming and text markup languages, respectively. Soopertree and other languages such as Python and Java can be used for desktop and web-based software development. Soopertree and Soopertags are implemented in Java, and are open source. Sooperzone.com hosts the desktop and web-based software, and TwindoBoard.com hosts a closed source whiteboard which is used to teach math, coding and web design.

Business Model

End-users and developers who wish to use applications based on Soopertags which support fancy font mode (variable-width fonts, multiple font sizes) must pay a Soopertree subscription fee of $10/year. All web-based applications support fancy font mode, and end-users can use them without paying any fees. Applications which do not support fancy font mode use only monospaced text in a single font size. Developers of the open source Soopertree and Soopertags languages share the revenue from Soopertree subscriptions (and web hosting, see below) in proportion to their cumulative number of GitHub commits.

Developers who upload high-powered web-based applications must pay web hosting fees starting at $5 per month. All web-based applications are hosted by Sooperzone.com. Anyone can upload/download Soopertree desktop applications (as well as applications written in Python, Java, and other languages) to/from the server. Users of the desktop whiteboard and tutor smartphone app (see TwindoBoard.com) must pay a subscription fee of $20/year. A multi-user version of the desktop whiteboard (multiple machines) with a teacher smartphone app costs $5/machine/year.

Soopertags

Soopertags is a markup language used to lay out text, widgets, images and canvas objects on the display screen. Soopertags documents can also be used as configuration files, saving user preferences and other data. Canvas objects are writable images, whereas normal image files are read-only. A special type of canvas called a container can contain variable-width text in multiple font sizes (as opposed to monospaced text), as well as other containers, canvas objects, and images. Converters will be developed to convert Soopertags and Soopertree/Python into HTML and Javascript, respectively.

TwindoBoard

TwindoBoard.com hosts software used to teach math, coding and web design to clients (students) of nonprofit organizations, for free. The tutors use a smartphone app in landscape mode which syncs the 2 displays viewed by the tutor and the student. Bluetooth is used for connectivity between the student's desktop/laptop and the tutor's smartphone. All lessons are in monospaced mode, using the desktop whiteboard. Tutors and students who are not registered with the nonprofit organizations pay a subscription fee of $20/year.

Whiteboard

Math basics: Use the arrow keys to move the cursor. Type underscore(s) to underline the numerator of a fraction. Use the special character command (Ctrl+K) to insert special characters such as pi, square root, sum, and integral. Use Tab/Shift+Tab to display/undo the next step in the math problem being solved. Type question mark (?) to explain the current step or to break the current step down into lower-level steps. Click on Help after typing question mark to access the help system.

More commands: The optional default to upper case setting assumes that all letters entered are upper case (use the shift key to enter a lower case letter). Use asterisk and slash for multiply and divide. Fractions or matrices enclosed in brackets use tall brackets. Smart down/up arrow: press it after inserting a character moves the cursor beneath/above that character. Functions such as lines and parabolas can be plotted interactively on a graph.

Expression Language

Mathematical expressions are encoded (internally) using the Soopertree programming language. Each step in the math problem being solved manipulates this Soopertree expression. Even if the user enters steps in a different order than the default ordering, the simplification logic can handle that. The user can type Tab/Shift+Tab to redo/undo her previous step, as well as to redo/undo the computer's previous step.

Whiteboard Grid Commands

Use Shift+Arrow Key to highlight a rectangular block. Press Insert to insert a row or column of spaces before a highlighted block (insert blank line if no highlight). Press Shift+Insert/Delete to insert/delete an entire row/column when a block is highlighted. Press Enter at end of a line of text: insert blank line, back up on that line to line up with beginning of text on previous line. Press Enter on blank line to back up to line up with beginning of text on a previous line, or insert blank line if already at beginning of line. Press Ctrl+Tab to move forward to line up with beginning of first or next word on a previous line. Press Home to move to beginning of text on current line, press it again to toggle between beginning of line and beginning of text. This usage of Enter, Tab and Home is useful for editing program code with multiple indentation levels. The user doesn't have to memorize these commands: type question mark at any time to access the help system.

Superscripts

Superscripts and subscripts in monospaced mode are handled by employing a vertical offset of half a line per level of superscripting or subscripting. The caret symbol (^) is used as a superscript prefix, double-caret (^^) is used as a subscript prefix, and backslash (\) is used as an escape character (terminate super/subscript with a semicolon). Carets and double-carets cannot be mixed (exception: one level of superscript can be combined with one level of subscript).

Revenue

Let's assume that gross annual revenue is $200,000. Assume user counts are each 2000 for developers and end-users, and are each 3000 for tutors and students. Also, say teachers generate 20 percent of revenue: $40,000. So developers and end-users generate revenue of $40,000, and tutors and students generate revenue of $120,000. The number of machines (each teacher has multiple machines or student-spots) is 8000. Net revenue is $160,000 after subtracting Soopertree revenue which is distributed to the developers of open source Soopertree.

Soopermoves

Soopermoves.com is a domain which redirects to a web page included under Sooperzone.com. Soopermoves includes a tool used for developing 2D and 2.5D games which are written in Soopertree, Python, Java, and other languages. Graphics supported include 2D and 2.5D, but not 3D. A dimetric projection is used to support 2.5D graphics. Multiplayer mode is supported using WebSocket.

Dimetric Projection

All planes are parallel or at 90 degree angles with each other, the vantage point of the user is at a 45 degree angle, and all horizontal/vertical lines in the horizontal plane are rendered such that the slope of the line is +/- 0.5 (vertical lines in vertical planes are always vertical). Only horizontal, vertical, and diagonal lines at 45 degree angles are allowed. Since all planes are angled instead of directly facing the user due to the dimetric projection, diagonal lines are not rendered using a slope of 0.5, but have some other slope. Curves are limited to circular arcs in multiples of 45 degrees. Due to the dimetric projection they are rendered as elliptical arcs. Text is monospaced and appears skewed. Labels are allowed which contain a single line of normal text, bounded by a normal rectangle. Labels are always displayed in front of/on top of the dimetric projection.

Animation

Objects can move in 8 directions in 2D mode (90 degree angles and 45 degree angles) and 6 directions in 2.5D mode (up/down, left/right, forward/backward). Objects may include discs and balls. Support for collision detection functionality is provided. The parent object of an animated 2.5D object is assumed to be located on the ground or building directly beneath it. Objects can also dynamically change shape, incrementally or all at once.

Implementation Steps

1. Finish Soopertree syntax checker - ***done!***
2. Finish unit testing of syntax checker - ***done!***
3. Develop Soopertree code execution - **doing it**
4. Release Soopertree as console-based compiler on GitHub
5. Make Soopertree display grid-based with monospaced text
6. Add mouse support
7. Develop desktop whiteboard
8. Develop tutor's smartphone app
9. Develop teacher's smartphone app
10. Partner with Progress Place to beta test tutor app
11. Develop monetizing functionality
12. Implement user forums
13. Launch TwindoBoard website
14. Purchase Google AdWords advertising
15. Write Soopertags design specs
16. Develop Soopertags
17. Integrate Soopertree with Soopertags
18. Develop Soopertree code editor
19. Develop Web Converter
20. Implement Keyboard Aid (bells and whistles of editor)
21. Develop Web Editor: WYSIWYG Soopertags editor
22. Beta test Soopertree SDK
23. Implement Python and Java APIs for Soopertags
24. Develop Soopermoves 2D and 2.5D game engines
25. Port Soopertree to Android
26. Develop iOS version of Soopertree

Project Summary

Sooperzone, in a nutshell, consists of a new programming language, a new, simplified text markup language called Soopertags, a web hosting service (and app store) for websites and desktop apps built using Soopertags, and a closed source whiteboard which is used to teach math, coding and web design.

Soopertree and Soopertags

Soopertree is a new language inspired by Python, but like Lisp has no infix operators. Operators come before their operands, parentheses are used for grouping, and program statements are terminated with semicolons. Soopertags uses square brackets for grouping instead of the tags enclosed in angle brackets used by HTML. It works with Soopertree and other languages such as Java and Python. Soopertree and Soopertags are open source and implemented in Java.

Business Model

Users pay $10/year to enable multiple, variable-width fonts. Users of the whiteboard pay $20/year, except clients of participating nonprofit organizations pay no fees. Teachers pay $5/machine/year to make use of the multi-user whiteboard. Power users need to pay web hosting fees for resource-intensive website hosting, starting at $5/month. Soopertree will eventually run on smartphones.

TwindoBoard

TwindoBoard.com includes the whiteboard, a desktop application which is written in Java and used by the student, and which displays the current lesson. The tutor sits next to the student holding a smartphone. Bluetooth is used to synchronize the display of the whiteboard with the partial display of the tutor smartphone app. The whiteboard is integrated with Soopertree, making it extensible. The whiteboard is capable of teaching arithmetic and algebra out of the box. Soopertree integration makes it capable of teaching other subjects such as literacy and more advanced math.

Web Development

Soopertags files can contain embedded Soopertree code. Soopertags works with mainstream languages like Java and Python, not just Soopertree. Client-side and server-side code of websites is coded in Soopertree or other languages, and converted into Javascript (unless it was in Javascript to begin with) before the website code is uploaded to the server. Soopertags code is converted into HTML prior to being uploaded.

Soopertree

Soopertree (implemented in Java) is a Python dialect in which all operators precede their operands, and parentheses are used for all grouping (except string literals, which are delimited with double quotes, also statements are separated by semicolons). Soopertree source files have a .SPTR extension. Soopertags files (the sister language of Soopertree, a text markup language) have a .SPTG extension. SOOPERTagS: Sooperior Object Oriented Programming Environment and Run-Time System.

Special Characters

( ) grouping
- word separator
; end of stmt.
: dot operator
" string delimiter
\ escape char.
comment
_ used in identifiers
$ string prefix char.
{ } block comment

Op Characters

+ - * / %

= < >

& | ^ ~ ! ?

Differences from Python

- Parentheses, not whitespace
- Operators come before their operands
- Integration with Soopertags
- Information hiding (public/private)
- Single, not multiple inheritance
- Adds interfaces ("hedron" defs.)
- Drops iterators and generators
- Adds lambdas
- Adds quote and list-compile functions, treating code as data
- Adds cons, car and cdr functionality

Grammar Notation

- Non-terminal symbol: <symbol>
- Optional text in brackets: [*text*]
- Repeats zero or more times: [*text*]…
- Repeats one or more times: <symbol>…
- Pipe separates alternatives: *opt1* | *opt2*
- Comments in *italics*

Keyboard Aid

This optional feature enables hyphens, open parentheses, and close parentheses to be entered by typing semicolons, commas, and periods, respectively. When enabled, keyboard aid can be temporarily suppressed by using the Ctrl key in conjunction with typing semicolons, commas, and periods (no character substitution takes place). By convention, hyphens are used to separate words in multi-word identifiers, but semicolons are easier to type than hyphens. Similarly, commas and periods are easier to type than parentheses. Typing semicolon converts previous hyphen to a semicolon, and previous semicolon to a hyphen (use the Ctrl key to override this behaviour). Typing semicolon after close parenthesis simply inserts semicolon. Typing space after hyphen at end of identifier converts hyphen to underscore. The close delim switch automatically inserts a closing parenthesis/double quote when the open delimiter is inserted.

Soopertags

Soopertags is a simplified markup language used to replace HTML. Mock JSON files using Soopertags syntax have a .SPJS extension, and include no commas. Instead of myid: val, use [myid: val]. Instead of [1, 2, 3], use [arr: [: 1][: 2][: 3]]. Arbitrary Soopertags code can be embedded in the Soopertree echo statement. Soopertags syntax, where asterisk (*) means occurs zero or more times, is defined as follows:

Tags:
- [tag]
- [tag (fld val)*: body]
- [tag (fld val)*| body |tag]

Body:
- text
- [(fld val)*: text]*

Call: (Soopertree code)
- [expr: <expr>]
- [exec: <stmt>...]
- [sptr: <path>]

Soopertree Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space):

<source file>:
- do ([<imp>]... [<def glb>] [<def>]... [<class>]...)

<imp>:
  <import stmt> **;**

<import stmt>:
  import <module>...
  from <rel module> import <mod list>
  from <rel module> import all

<module>:
  <name>
  (**:** <name><name>...)
  (as <name><name>)
  (as (**:** <name><name>...) <name>)

<mod list>:
  <id as>...

<id as>:
  <mod id>
  (as <mod id><name>)

<mod id>:
  <mod name>
  <class name>
  <func name>
  <var name>

<rel module>:
  (**:** [<num>] [<name>]...)
  <name> *// ?*

<class>:
- <cls typ><name> [<base class>] [<does>] [<vars>] [<ivars>] do (<def>…) **;**
- abclass <name> [<base class>] [<does>] [<vars>] [<ivars>] do (<anydef>…) **;**
- <hedron><name> [<does>] [<const list>] do ([<abdef>]... [<defimp>]...) **;**
- enum <name><elist> **;**
- ienum <name><elist> **;**

<cls typ>:
  class
  iclass

<hedron>:
  hedron
  ihedron

<does>:
  (does <hedron name>...)

<hedron name>:
<base class>:
  <name>
  (**:** <name><name>…)

<const list>:
  (const <const pair>...)

<const pair>:
  (<name><const expr>)

<def glb>:
  gdefun [<vars>] [<ivars>] do <block> **;**

<def>:
- <defun> (<name> [<parms>]) [<vars>] [<gvars>] [<dec>] do <block> **;**

<defimp>:
- defimp (<name> [<parms>]) [<vars>] [<gvars>] [<dec>] do <block> **;**

<abdef>:
  abdefun (<name> [<parms>]) [<dec>] **;**

<defun>:
  defun
  idefun

<anydef>:
  <def>
  <abdef>

<vars>:
  (var [<id>]...)

<ivars>:
  (ivar [<id>]...)

<gvars>:
  (gvar [<id>]...)

<parms>:
  [<id>]... [<parm>]... [(* <id>)] [(** <id>)]

<parm>:
  (<set op><id><const expr>)

<dec>:
  (decor <dec expr>...)

<block>:
  ([<stmt-semi>]…)

<stmt-semi>:
    <stmt> **;**

<jump stmt>:
    <continue stmt>
    <break stmt>
    <return stmt>
    return <expr>
    <raise stmt>

<raise stmt>:
    raise [<expr> [from <expr>]]

<stmt>:
    <if stmt>
    <while stmt>
    <for stmt>
    <switch stmt>
    <try stmt>
    <asst stmt>
    <del stmt>
    <jump stmt>
    <call stmt>
    <print stmt>
    <bool stmt>

<call expr>:
- (<name> [<arg list>])
- (**:** <colon expr>… (<method name> [<arg list>]))
- (call <expr> [<arg list>])

<call stmt>:
- <name> [<arg list>]
- **:** <colon expr>… (<method name> [<arg list>])
- call <expr> [<arg list>]

<colon expr>:
    <name>
    (<name> [<arg list>])

<arg list>:
    [<expr>]... [(<set op><id><expr>)]...

<dec expr>:
    <name>
    (<name><id>...)
    (**:** <name><id>...)
    (**:** <name>... (<id>...))

<dot op>:
    dot | **:**

<del stmt>:
    del <expr>

<set op>:
    set | =

<asst stmt>:
    <asst op><target expr><expr>
    <set op> (tuple <target expr>...) <expr>
    <inc op><name>

<asst op>:
    set | addset | minusset | mpyset | divset |
    idivset | modset |
    shlset | shrset | shruset |
    andbset | xorbset | orbset |
    andset | xorset | orset |
    = | += | -= | *= | /= |
    //= | %= |
    <<= | >>= | >>>= |
    &= | ^= | '|=' |
    &&= | ^^= | '||='

<target expr>:
    <name>
    (**:** <colon expr>… <name>)
    (slice <arr><expr> [<expr>])
    (slice <arr><expr> all)
    (<crop><cons expr>)

<arr>: *// string or array/list*
    <name>
    <expr>

<if stmt>:
- if <expr> do <block> [elif <expr> do <block>]…
  [else do <block>]

<while stmt>:
    while <expr> do <block>
    while do <block> until <expr>

<for stmt>:
- for <name> [<idx var>] in <expr> do <block>
- for (<bool stmt>**;** <bool stmt>**;** < bool stmt>) do <block>

<try stmt>:
- try do <block> <except clause>… [else do <block>] [eotry do <block>]
- try do <block> eotry do <block>

<except clause>:
    except <name> [as <name>] do <block>

<bool stmt>:
    quest [<expr>]
    ? [<expr>]
    <asst stmt>

<switch stmt>:
    switch <expr><case body> [else do <block>]

<case body>:
    [case <id> do <block>]...
    [case <dec int> do <block>]...
    [case <str lit> do <block>]...
    [case <tuple expr> do <block>]...

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<paren stmt>:
    (<stmt>)

<qblock>:
    (quote [<paren stmt>]...)

<expr>:
    <keyword const>
    <literal>
    <name>
    (<unary op><expr>)
    (<bin op><expr><expr>)
    (<multi op><expr><expr>…)
    (<quest><expr><expr><expr>)
    <lambda>
    (quote <expr>...)
    <cons expr>
    <tuple expr>
    <list expr>
    <dict expr>
    <venum expr>
    <string expr>
    <bytes expr>
    <target expr>
    <call expr>
    <cast>

<quest>:
    quest | ?

<inc op>:
    incint | decint | ++ | --

<unary op>:
    minus | notbitz | not |
    - | ~ | !

<bin op>:
    <arith op>
    <comparison op>
    <shift op>
    <bitwise op>
    <boolean op>

<arith op>:
    div | idiv | mod | mpy | add | minus |
    / | // | % | * | + | -

<comparison op>:
    ge | le | gt | lt | eq | ne | is | in |
    >= | <= | > | < | == | !=

<shift op>:
    shl | shr | shru |
    << | >> | >>>

Note: some operators delimited with single quotes for clarity (quotes omitted in source code)

<bitwise op>:
    andbitz | xorbitz | orbitz |
    & | ^ | '|'

<boolean op>:
    and | xor | or |
    && | ^^ | '||'

<multi op>:
    mpy | add | strdo | strcat |
    and | xor | andbitz | xorbitz |
    or | orbitz |
    * | + | % | + |
    && | ^^ | & | ^ |
    '||' | '|'

<const expr>:
    <literal>
    <keyword const>

<literal>:
    <num lit>
    <str lit>
    <bytes lit>

<cons expr>:
    (cons <expr><expr>)
    (<crop><expr>)

<tuple expr>:
    (tuple [<expr>]…)
    (<literal> [<expr>]…)
    ( )

<list expr>:
    (jist [<expr>]…)

<dict expr>:
    (dict [<pair>]…)

<pair>:
    // expr1 is a string
    (**:** <expr1><expr2>)
    (**:** <str lit><expr>)

<venum expr>:
    (venum <enum name> [<elist>])
    (venum <enum name><idpair>...)

<elist>:
    <id>...
    <intpair>...
    <chpair>...

<intpair>
    // integer constant
    <int const>
    (**:** <int const><int const>)

<chpair>
    // one-char. string
    <char lit>
    (**:** <char lit><char lit>)

<idpair>
    <id>
    (**:** <id><id>)

<cast>:
    (cast <literal><expr>)
    (cast <class name><expr>)

<print stmt>: *// built-in func*
    print <expr>…
    println [<expr>]…
    echo <expr>…

<lambda>:
    (lambda ([<id>]...) <expr>)
    (lambda ([<id>]...) do <block>)
    (lambdaq ([<id>]...) do <qblock>)
    // must pass qblock thru compile func

No white space allowed between tokens, for rest of Soopertree Grammar

<white space>:
    <white token>...

<white token>:
    <white char>
    <line-comment>
    <blk-comment>

<line-comment>:
    # [<char>]... <new-line>

<blk-comment>:
    { [<char>]... }

<white char>:
    <space> | <tab> | <new-line>

<name>:
- [<underscore>]… <letter> [<alnum>]…
  [<hyphen-alnum>]… [<underscore>]…

<hyphen-alnum>:
    <hyphen><alnum>…

<alnum>:
    <letter>
    <digit>

In plain English, names begin and end with zero or more underscores. In between is a letter followed by zero or more alphanumeric characters. Names may also contain hyphens, where each hyphen is preceded and succeeded by an alphanumeric character.

<num lit>:
    <dec int>
    <long int>
    <oct int>
    <hex int>
    <bin int>
    <float>

<dec int>:
    [<hyphen>] 0
    [<hyphen>] <any digit except 0> [<digit>]…

<long int>:
    <dec int> L

<float>:
    <dec int><fraction> [<exponent>]
    <dec int><exponent>

<fraction>:
    <dot> [<digit>]…

<exponent>:
    <e> [<sign>] <digit>…

<e>:
    e | E

<sign>:
    + | -

<keyword const>:
    null
    true
    false

<oct int>:
    0o <octal digit>…

<hex int>:
    0x <hex digit>…
    0X <hex digit>…

<bin int>:
    0b <zero or one>…
    0B <zero or one>…

<octal digit>:
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<hex digit>:
    <digit>
    A | B | C | D | E | F
    a | b | c | d | e | f

<str lit>:
    " [<str item>]... "

<str item>:
    <str char>
    <escaped str char>
    <str newline>

<str char>:
    any source char. except "\", newline, or
    end quote

<str newline>:
    \ <newline> [<white space>] "

<escaped char>:
    \\ *backslash*
    \" *double quote*
    \} *close brace*
    \a *bell*
    \b *backspace*
    \f *formfeed*
    \n *new line*
    \r *carriage return*
    \t *tab*
    \v *vertical tab*
    \ooo *octal value = ooo*
    \xhh *hex value = hh*

<escaped str char>:
    <escaped char>
    \N{name} *Unicode char. = name*
    \uxxxx *hex value (16-bit) = xxxx*

<crop>:
    c <crmid>... r

<crmid>:
    a | d

Not implemented: string prefix and bytes data type (rest of grammar)

<str lit>:
    [$ <str prefix>] <quoted str>

<str prefix>:
    r | R

<quoted str>:
    " [<str item>]... "

<bytes lit>:
    $ <byte prefix><quoted bytes>

<byte prefix>: *// any case/order*
    b | br

<quoted bytes>:
    " [<bytes item>]... "

<bytes item>:
    <bytes char>
    <escaped char>
    <str newline>

<bytes char>:
    any ASCII char. except "\", newline, or
    end quote