Mike Hahn – hahnbytes@gmail.com

Moborigin

Moborigin is a tool used to develop online communities accessed from Android and iOS mobile devices. Moborigin includes a new programming language called MoboLisp, along with a screen layout language called MoboTags, and a MoboLisp app store. MoboLisp also runs on Windows/Mac/Linux.

Revenue Sources

- Online Communities, user fees:
  - Members: $1/year (nonprofits)
  - Customers: $2/year
  - Employees: $10/year
  - Discount for >50 members/customers: 75% off
- Unrestricted Mode: $20 (one-time fee)
- Restricted Mode: FREE, pick one of the following
  1. Unichrome: only grayscale or gray mixed with a single primary color allowed
  2. Monospaced: grid of monospaced text fills screen
- Ads (optional):
  - Unrestricted mode apps for everyone
  - Online Communities: FREE
- TwindoBoard tutor app:
  - FREE for nonprofits
  - otherwise an unrestricted mode app
- Tutors: $20/year (directory listing, get paid hourly rate by students)
- Web hosting fees paid by developers of resource-hungry apps

Online Communities

- MoboLisp and MoboTags together form the simplest programming-language/layout-manager duo in existence, yet MoboLisp is almost as powerful as Python (though lacking its extensive libraries).
- Easily customized by adding or modifying MoboLisp and MoboTags code, or implemented from scratch.
- Members can share posts, comments, images, videos, web links, written material (plain text or formatted with MoboTags code), music/audio, and custom code.
- Tutors can teach math, coding and web design to the members.

Business Model

Non-profit organizations pay $1 per member per year, no charge for employees. For-profit companies pay $2 per customer per year and $10 per employee per year. Organizations having more than 50 members/customers enjoy a quantity discount of 75% off, paying just $0.25 or $0.50/year for each of the 51st and all subsequent members/customers, respectively. Members, customers and employees need not be Moborigin premium users.

Restricted Mode

Moborigin premium users pay an unlocking fee of $20. Those users have the option of using paid apps, and the ability to run MoboLisp apps in unrestricted mode. Restricted mode can be either unichrome or monospaced. In unichrome mode, all pixels are either in shades of gray or a mixture of zero or more gray and a single theme color. Six theme colors are available: red, green, blue, cyan, yellow, and magenta, but each app can only use one theme color. In monospaced mode, the entire screen consists of a grid of monospaced text, with embedded widgets and images.

Ads

Organizations making use of online communities have the option of displaying ads. Those communities are free: no fees are charged per member/customer/employee. Apps which display ads in unrestricted mode are accessible to all users, and the ad revenue is split evenly between the app authors and Moborigin.

Tutors

Moborigin provides a paid app, included with your unlocking fee, used for teaching math and other subjects. Tutors must pay a subscription fee of $20/year (starting after the first year), to be included in the tutor directory and to accept credit card payments from their students. This app is called TwindoBoard (free for the non-profit organizations).

Web Users

For those members of non-profit organizations who lack smartphones, a web-based interface will be provided. A conversion utility will be used to convert MoboTags and MoboLisp code into HTML and JavaScript, respectively.

Google's Cut

Both Google and Apple take a 30 percent cut of in-app purchases, which usually drops to 15 percent after the first year. The MoboLisp Runtime Environment (MRE) is just another Android/iOS app, so all payments from users are subject to the 30 percent commission to Google and Apple. Web hosting fees are charged by Google, so developers of resource-hungry apps pay extra.

Exit Strategy

In case neither the online communities, tutoring software, nor MoboLisp app store are profitable, the Java (and Swift) source code of the MRE will be released on GitHub. This can be used to create standalone Android and iOS mobile apps by bundling the MRE with the MoboLisp/tags source code of each app in the MoboLisp app store.

Implementation Steps

1. Finish MoboLisp syntax checker - ***done!***
2. Finish unit testing of syntax checker - ***done!***
3. Develop foundation of MoboLisp code execution - ***almost done!***
4. Develop rest of MoboLisp code execution
5. Release MoboLisp as console-based compiler on GitHub
6. Implement GUI: monospaced mode
7. Write MoboTags design specs
8. Develop MoboTags
9. Integrate MoboLisp with MoboTags
10. MoboLisp/tags: MoboLisp Runtime Environment (MRE)
11. Use Specialisterne to hire local Android programmer on spectrum
12. Port system to Android
13. Make pitch to DMZ tech incubator
14. Use Specialisterne to hire remote iOS programmer on spectrum
15. Convert MRE to Swift
16. Port system to iOS
17. Search for angel investor
18. Without angel investor, do not renew contracts of autistic programmers
19. Develop MoboLisp code editor
20. Develop TwindoBoard
21. Implement Keyboard Aid (bells and whistles of editor)
22. Develop WYSIWYG MoboTags screen editor
23. Implement online community using MoboLisp/tags

24. Perform beta testing
25. Develop monetizing functionality
26. Design website
27. Launch Moborigin website and app
28. Purchase Google AdWords advertising
29. Enable apps which display ads
30. Develop converters: MoboTags/MoboLisp to HTML/JavaScript
31. Develop game engine
32. Exit strategy: if necessary, release Java code of MRE on GitHub

Games

A game engine which supports multiplayer games (using Bluetooth) is written in Java. The games themselves are written in MoboLisp. Graphics supported include 2D and 2.5D, but not 3D. A dimetric projection is used to support 2.5D graphics.

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.

TwindoBoard

Moborigin.com hosts TwindoBoard 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 an unlocking fee of $20.

Math

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 MoboLisp programming language. Each step in the math problem being solved manipulates this MoboLisp 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

The next 2 paragraphs may be ignored, they are written in computerese. 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).

Founder Bio

I am Mike Hahn, the founder of Moborigin.com. I was previously employed at Brooklyn Computer Systems as a Delphi Programmer and a Technical Writer (I worked there between 1996 and 2013). At the end of 2014 I quit my job as a volunteer tutor at Fred Victor on Tuesday afternoons, where for 5 years I taught math, computers, and literacy, and became a volunteer math/computer tutor at West Neighbourhood House. I quit that job in mid-2019. I have a part-time job working for a perfume store. My hobbies are reading and I often go for walks. I don't read books very often, but on March 19, 2021 I started reading a biography of Steve Jobs which my brother gave me. I read the CBC news website, news/tech articles on my Flipboard app, and miscellaneous articles on my phone (same screen as my Google web page). I visit my brother once a month or more. For almost 30 years I was depressed on and off (I'm a rapid cycler), but it largely vanished after I ramped up development of my Moborigin project in early March 2021.

MoboLisp

MoboLisp (implemented in Java) is an open source 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). MoboLisp source files have a .MOOB extension. MoboTags files (the sister language of MoboLisp, a text markup language) have a .MBTG extension. MoboLisp sports a Lisp-like syntax.

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 MoboTags
- 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.

MoboTags

MoboTags is a simplified markup language used to replace HTML. Mock JSON files using MoboTags syntax have a .MBJS extension, and include no commas. Instead of myid: val, use [myid: val]. Instead of [1, 2, 3], use [arr: [: 1][: 2][: 3]]. Arbitrary MoboTags code can be embedded in the MoboLisp echo statement. MoboTags 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: (MoboLisp code)
- [expr: <expr>]
- [exec: <stmt>...]
- [moob: <path>]

MoboLisp 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 MoboLisp 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