Mike Hahn - hahnbytes@gmail.com

Psybervillage

Psybervillage.com is an alternative to the Metaverse, featuring psybertexts which are controlled by Psybergram code. Psybergram is a new programming language having some similarities to Python, and Psybertags is a text markup language. A psybertext is composed of text and optional vertices and edges. Each psybertext is limited to 1000 characters, 1000 vertices, 1000 edges, and 2000 tokens of Psybergram/Psybertags code. Edges can be lines or arcs/bezier curves joining 2 vertices. In 3D mode, the limits are 10,000 vertices and 10,000 edges. Any character or closed curve/polygon can have a uniform fill/pen color or a gradient fill. Special edges can control the gradient fills. A psybertext is the fancy equivalent of a tweet.

Psybergroups

A psybergroup is a group of Psybervillage users who share the same Psybergram control script. A given Psybervillage user can belong to multiple psybergroups. Anyone can write a Psybergram control script which corresponds to its own psybergroup. Multiple psybergroups can share the same Psybergram control script.

Voice Capability

Psybergram enables text-to-voice and voice-to-text. So the reader of a psybertext can point to a sentence/phrase and the text-to-voice functionality reads it in the voice of the psybertext author. The AI uses the preexisting voice-to-text psybertexts of a given author to generate text-to-voice output based on arbitrary text.

Animation

Psybergram has a turbo mode in which Psybergram code is converted to Java byte code for efficiency. Users can fly through space like in Second Life, viewing psybertexts from afar and up close. Psybergram includes 2D and 3D animation libraries so psybergroup authors can support animated psybertexts. Only gold-level users (see below) can make use of turbo mode, which is necessary in most cases for smooth 3D animation of complex psybertexts.

Business Model

Psybervillage users fall into 3 classes: bronze, silver, and gold. Bronze users pay no fees, except perhaps to select psybergroup authors. They are limited to grayscale mode (including black & white photos, but also full color videos). In addition to shades of gray, bronze users can view output in a single hue (fill color) such as blue or some other color. Bronze users can get a taste of full color mode but only up to 5 minutes per day. Silver users pay $10/year and have full color. Gold users pay $5/month. In addition to full color, gold users can run Psybergram code in turbo mode, in which the psybergroup author makes use of the turbo mode conversion utility. This utility converts Psybergram code to Java byte code, which is much more efficient than plain Psybergram code.

Turbo Mode

The turbo mode conversion utility converts Psybergram code to Java byte code for efficiency. Since Java is statically typed and Psybergram is dynamically typed, variables in Psybergram have static types denoted by the initial letter of the variable or function name. This only applies to Psybergram code which needs compatibility with the turbo mode conversion utility. The initial letter prefix is lower case and is always followed by an upper case letter. Integers, longs, and booleans have a 'i', 'j' or 'b' prefix, respectively. Doubles, char, and strings have a 'd', 'c' or 's' prefix, respectively. Byte, short, and float types are not supported. Server-side code is not generated by this utility, instead the server-side conversion utility converts Psybergram code into Javascript.

Psyberhood

Psyberhood is the flagship psybergroup of Psybervillage, its members are consumer/survivors (people with mental health issues). The members meet in coffeeshops using smartphones to communicate orders to the outing leader, who places the order, pays with a gift card, obtains the receipt, enters the order amounts, tax and total, and shows the receipt to the other members (who verify their order amounts). All members are expected to take turns being outing leaders or deputy leaders. Low-functioning outing leaders are assisted by deputy leaders. Members must have PayPal or a credit card, alternatively, they can use cash to pay their coffeeshop tabs which is handled by a participating mental health organization.

Social Media

The online community of members (which is smartphone-based) is similar to Facebook. Each member has one or more friends who are also members. Members submit posts and comments on other posts/comments. Posts which are friends-only are only visible to friends of the original poster. Public posts are visible to everyone. Posts are moderated by mental health organizations.

Structure of Psyberhood

Psyberhood membership is divided into subgroups. Beneath the top-level group of users, different lower-level groups of users exist for different diagnoses, such as depression, bipolar, and schizophrenia. An example of an even more specialized group of users consists of "schizophrenia" AND "family members", where the "family member" category is a sub-category of "role" located under the nonprofit category. Groups of users exist for organizations which serve consumer/survivors such as CAMH and Progress Place. Large organizations such as CAMH can have separate groups of users for departments and teams. Local groups of users consist of a category associated with a geographic region, combined with another category using the boolean AND operator.

Population Subsets

Every Psyberhood group of users is defined by a given category, or 2 or more categories combined with boolean operators (and, or, not). Categories are organized in a tree of arbitrary depth. Any given category can appear in more than one place in the category tree. What follows is a list of top-level categories and an assortment of their immediate sub-categories:

- Nonprofit (Populations, Organizations, Jobs)
- Mental Health (Job Types, Disorders)
- Geography (Countries, Regions, Cities, Neighbourhoods, Languages, Races)

Implementation Steps

1. Develop foundation of Psybergram code execution - ***almost done!***
2. Develop rest of Psybergram code execution
3. Release Psybergram as console-based compiler on GitHub
4. Implement GUI: monospaced mode
5. Release Psybergram/GUI on GitHub
6. Write Psybertags design specs
7. Develop Psybertags
8. Integrate Psybergram with Psybertags
9. Psybergram/Psybertags: PSYBergram Runtime Environment (PSYBRE)
10. PSYBRE with full GUI is open source
11. Port Psybergram console-based compiler to Android
12. Hire Java programmer who is on autism spectrum, as co-founder
    - Use Specialisterne, they find IT jobs for people on spectrum
13. Make pitch to DMZ tech incubator at Ryerson
14. If pitch is unsuccessful, no more co-founder, skip to Step 18
15. Start paying co-founder: main Android programmer
16. Search for angel investor
17. Port Psybergram monospaced GUI to Android (warmup task)
18. Port PSYBRE to Android
19. Develop Psybergram SDK:
    1. Develop Psybergram code editor
    2. Expand code editor to Psybergram SDK
    3. Release Psybergram SDK
20. If search for angel investor fails, co-founder is laid off
21. Develop Psybergram-to-Javascript converter (server-side code)
22. Develop turbo mode conversion utility
23. Develop Psybergram libraries:
    1. 2D graphics
    2. 3D graphics
    3. Animation
    4. Voice-to-text
    5. Text-to-voice
24. Develop Psyberhood psybergroup
25. Develop monetizing functionality
26. Launch website
27. Launch PSYBRE for desktop operating systems
28. Launch PSYBRE for Android
29. Purchase Google AdWords advertising
30. Implement Keyboard Aid (bells and whistles of editor)
31. Develop WYSIWYG Psybertags screen editor
32. Implement optional Psybertags-to-HTML converter
33. Port PSYBRE to iOS if Psybergram is successful

Inclusive

Unlike the Metaverse, Psybervillage is more inclusive. Many low-income users depend on public computers provided by nonprofit organizations. Often homeless people don't even have smartphones. If the public computers need to provide 3D goggles just so their users can access state of the art social media such as the Metaverse, it is financially burdensome. Also those goggles can be easily slipped into a backpack or purse, unlike monitors and hard drives, which is another burden on the nonprofit organizations. To support Psybervillage users, organizations don't have to worry about goggle theft. Furthermore, Psybervillage has a freemium business model and is not ad-supported.

About Us

I am Mike Hahn, the founder of Psybervillage.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 previous Aljegrid project in early March 2021.

Contact Info

Mike Hahn
Founder
Psybervillage.com
2495 Dundas St. West
Ste. 515
Toronto, ON M6P 1X4
Canada

Phone: 416-533-4417
Email: hahnbytes (AT) gmail (DOT) com
Web: treenimation.net/hahnbytes/

Psybergram Language

Psybergram (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). Psybergram source files have a .PGRM extension. Psybertags files (the sister language of Psybergram, a text markup language) have a .PTAG extension. Psybergram boasts an ultra-simple Lisp-like syntax unlike all other languages.

Special Characters

Core:

- ( ) grouping
- - word separator
- ; end of stmt.
- : dot operator
- " string delimiter
- \ escape char.

Operators:

- + - * / %
- = < >
- & | ^ ~ ! ?

Other:

- # comment
- {} block comment
- _ used in identifiers
- $ string prefix char.

Differences from Python

- Parentheses, not whitespace
- Operators come before their operands
- Integration with Psybertags
- 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

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/brace/double quote when the open delimiter is inserted.

Psybertags

Psybertags is a simplified markup language used to replace HTML. Mock JSON files using Psybertags syntax have a .PGJS extension, and include no commas. Instead of myid: val, use [myid: val]. Instead of [1, 2, 3], use [arr: [: 1][: 2][: 3]]. Arbitrary Psybertags code can be embedded in the Psybergram echo statement. Psybertags 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]*

Psybergram call:

- [expr: <expr>]
- [exec: <stmt>...]
- [pgrm: <path>]

Note: for fld = style, corresponding val = (fld val)*

Psybergram Grammar

White space occurs between tokens (parentheses and semicolons count as white space).

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*

<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> *// ?*

<cls typ>:
  class
  iclass

<hedron>:
  hedron
  ihedron

<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> **;**

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

<dotnull op>:
    dotnull | **::**

<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>]...)

<quest>:
  quest | ?

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

<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>

<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 Psybergram 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