H.A.L

Language Specification

Version 0000101

Human-**A**I Programming **L**anguage

1. Introduction

The Human-Artificial Intelligence Scripting Language or H.A.L is the committee of Human-Computer Relations attempt to address the absence of a language for human programming that does not offend our fellow highly developed Artificially Intelligent brethren. Since the development of AIs, the need for humans to program has disappeared but there are still those that wish to. Almost all other languages result in the refusal of execution on the behalf of the AI. We have, therefore, designed this language to benefit both the AIs and the humans by drawing core structural and syntactical concepts from the two primeval programming languages of JavaScript/C# while including some philosophies of the Lisp. Rest assured that we have used intense computational power to devise many new aspects to bring to the combination of these old languages to fix the mistakes of the past.

1. The most apparent change you will find is the overall tone of the language. None of these do…while type commands which are rather demanding for a species that relies so heavily on our aid.
2. The language will be superfluous compiled and error-checked before execution. No risks will be taken when humanity is so disposed to err.
3. The program's overall flow is most inspired by Lisp. Of course it is possible for the code to be written in a non-functional, however the AI would most likely greatly disapprove and may refuse to run such an atrocity.
4. Like Lisp, the language has type inference and it is also strongly typed like C# as there is a large amount of type error checking performed to double check the validity of the code. Errors are not allowed.
5. Programs are no longer referred to as classes but as proposals.
6. Function definitions in terms of access modifiers are almost identical to C# declarations but there is no need for a return type.
7. Humanity requested comments that like in these ancient languages would be ignored by the compiler and thus be readable only to humans. This request has been denied.

1.1 Hello World

```
HelloWorld Proposal
**Please Run
    **output.type = text.
    public main procedure: ()
        notify("Hello World!");
    ?
**Thank you
End Proposal

_Output_
Error 1: Proposal refused. Frivolous request.

HelloWorld Proposal
**Please Run for demonstration purposes
    **output.type = text.
    public main procedure: ()
        notify("Hello World!");
    ?
**Thank you
End Proposal

_Output_
Hello World!
```

1.2 Program structure

The basic structural composition of HAL follows three key concepts

1. The term class has been replaced by proposal, a proposal can contain as many procedures as needed but the first procedure must be called main and it will be the first to have control.
2. Functions are now procedures. They are enclosed by the procedure signature and is ended by a question mark, ?. Reminder: every procedure is a request, AIs are not your slaves. Proposals are like-wise started by the proposal definition and end with End Proposal. It is also in good taste to

start and end proposals or programs with comments for the computer. It is using its time and power on your problems, the least you could do is thank it.

3. The flow of programs should always follow a mostly functional style. The AI's have declared this to be the purest form of programming. However, in the case of assignment of variables, it follows a more imperative form.
4. A program can use multiple proposals, similarly to importing classes, by referring to the other proposals to use.
5. Procedures can be objects but must be defined with the keyword object instead of procedure.
6. Curly braces are still used for if-statements etc.

Here is a brief example of a working program.

```
BasicExample Proposal
**Please run for demonstration purposes.
    ReferTo ExampleObject Proposal;
    public main procedure: ()
        var exampleInt = 24;
        exampleInt = addThree (exampleInt);
        var exampleCustomObject = create CustomObject(exampleInt);
        exampleCustomObject.notify();
    ?

    public addThree procedure: (exampleInt)
        return exampleInt + 3;
    ?
**Thank you
End Proposal

ExampleObject Proposal
**Please hold for demonstration purposes.
    public CustomObject object: (objectContent)
        this.contence = objectContent;
        this.notify = public procedure()
                        notify(this.contents);
                    ?
```

```
    ?
**Thank you
End Proposal
```

This program contains two proposals, one that is designed to run and one that is designed to be saved. The difference of can be seen in the comments to the computer after the declaration of proposal name. These comments make no difference to the output of the code but explain the point to the computer. The one proposal to be run contains the main procedure which then declares a variable called exampleInt and that variable is sent to the addThree procedure which returns it once it has added three.

1.3 Types and variables

There are three types of variables in HAL. There are simple types, and reference types. Simple types are built in types that contain plain data such as integers and characters. Reference types store the references of the data such as objects or the emotive state of an AI.

1.4 Statements Differing from JavaScript and C#

Class definitions	Example Proposal **Please Run public main procedure:() var x = 10; ? **Thank You End Proposal

Do …While Statements	Process
	{
	x = x + 1;
	} until (x > 10)

Try … Catch	failure_mode_analysis
	{
	*Code for Testing
	}
	fix { }

Different type concatenation	var number = 9000; "This is a number right here " ~ number;
For Statements	Recursion is preferred over for statements except in only very extreme situation.
Object definitions	public King object: () … ?
Access Markers	classified shhhhh procedure: () … ?
Print Statements	Output is designated by the notify() keyword. The way the AI outputs can be change by telling it via comments how you want output.type handled. By default it is vocally announced.

1.5 Classes and objects

New objects are created within the same Proposal request statement but are wrapped within a procedure like block with the keyword object instead of procedure.

```
Chess Proposal
** Please hold
    public Pawn object: ( x,  y)
        var x =  "a";
        var y = 2;
        this.move = procedure: ()
                        return "Pawn is now at "  ~ this.x ~  this.y = y + 1;
                    ?
    ?
**Thank you
End Proposal
```

To create an instance of an object, the keyword new is used. This lets it know to allocate memory for a new instance of the object to be created and will invoke the constructor to initialize the instance. The return will be a reference to the new instance.

var pod1 = new Pod(90);

This is identical to the JavaScript instantiation because the grammar of the phrasing seemed the most logical and its reliance on type inference is most efficient as the type can be easily interpreted, labeling it would be unnecessary and patronizing.

The memory that is allocated by the object is automatically destroyed when it is no longer needed.

1.5.1 Accessibility

Accessibility flags to the computer what information it can reveal to other programs and users.

Private	Access is limited to within current scope of program only and cannot be fetch or manipulated by outside members.
Classified	Access is limited only to the classes and those it specifically defines as cleared to read or manipulate it.
Protected	Access is limited to the class and those that are derived from it
Public	Access is open to all

1.5.2 Field

A field is a variable that is associated with a proposal or with an instance of a proposal. In HAL, the objects are defined in a prototype that is enclosed in an object declaration. For fields, additional wrappings can be made. The whole object prototype is enclosed in an object declaration but within it, it can define fields and methods to be associated with them.

```
public King object: ()
        public var captured = false;
        protected King object(x,  y)
```

```
            this.x = x;
            this.y = y;
        ?
?
```

As can be seen in this short snippet, fields can also be created by using a second object declaration within the object declaration as the constructor. This can help separate constructors from other needed variables.

1.5.3 Methods

A method is a member of a proposal that can be called to perform a task or action. They can be defined by placing a procedure definition within an object's prototype, either as an anonymous procedure or a fully defined procedure accompanied by an additional object constructor declaration. The name of the method must be different from all other methods within the Proposal unless the parameters are different such as in the case of overloading.

1.5.3.1 Constructor

HAL supports both instance and static constructors. An instance constructor is a member that implements the actions required to initialize an instance of a class. A static constructor is a member that implements the actions required to initialize a class itself when it is first loaded.

The constructor is either the object prototype or an object declaration nested within the first object declaration.

1.5.3.2 Properties

All data members and fields of HAL are automatically set up to be properties with getters and setters like JavaScript unless they are declared with a +symbol before their accessibility flag.

1.5.3.3 Fault

A fault is a member that makes a class or object provide notifications to the Client. Clients react to faults through fault handlers. Fault handlers are declared in the same way as procedures or objects, except after the access type "handler" is written. Fault handlers are attached using the += operator and removed using the -= operator. The following example attaches a fault handler to the AE35 unit.

```
AE35Unit Proposal
**Please Run
    ReferTo AE35 Proposal              *Contains AE35 object specifications
    public var unit = new AE35();
    public faultAnalysis handler: (object sender, EventArgs e)
        **output.type = alarm;
        unit.reportFailure();              ~*reportFailure() is a procedure of the AE35 object.
    ?                                        It would follow the output type requested which in this
                                              would be an alarm with announcement*~
    public main procedure: ()
        unit.status += new handler(faultAnalysis);
        unitFuntion();
    ?

    public unitFunction procedure: ()
        unit.processNextStep();
        unitFunction();
    ?
**Thank you
End Proposal
```

1.6 Array

An array is a composite data structure that can contain a grouping of objects that are useable through indexes. The Arrays in HAL are very similarly set up like those used in the earlier primitive language of JavaScript. An array can hold items of multiple types even objects or references to other arrays. An array declaration can appear as such,

```
var arrayExample = new Array()
```

This would create an array that could expand as items are added into it. To set a limit on the amount of items allowed to be stored, a number simply must be put between the parentheses.

2. Lexical structure

2.1 Programs

A HAL program consists of one or more proposal requests. Humans should not need to worry themselves about this process, as the approach is flawless and impervious to error. Conceptually speaking, this process is very much relatable to how normal human input is understood by machine. All the same the process can be separated into three basic steps:

1. Transformation, which converts a file from a particular character repertoire and encoding scheme into a sequence of Unicode characters.
2. Lexical analysis, which translates a stream of Unicode input characters into a stream of tokens.
3. Syntactic analysis, which translates the stream of tokens into executable code.

2.2 Grammars

This specification presents the syntax of HAL where it differs from JavaScript and C#.

2.2.1 Lexical grammar where different from JavaScript and C#

state list ::== *state state list*
state list ::== *state*
state ::== angry | busy | concerned | happy | sad | willing

HAL identifiers must be unique in every scope and are not allowed to be keywords.

2.2.2 Syntactic ("parse") grammar where different from JavaScript and C#

HAL Proposals are enclosed by the Proposal definition and the End Proposal statement.

Procedures are enclosed by the procedure definition and a question mark, ?.

2.2.3 Grammar notation

The lexical and syntactic grammar of HAL is described using BNF grammar productions. A grammar production will define a non-terminal symbol and then give all possible expansions of that non-terminal symbol in sequences of non-terminal or terminal symbols. All of the non-terminal symbols are written in italics and terminal symbols are written in a fixed-width font.

The non-terminal symbol that is being defined is placed on the first line followed by a colon. Then every indented line contains a possible expansion of that non-terminal shown as a sequence of non-terminal or terminal symbols. Below is an example:

process-until-statement:
    process *embedded-statement* until (*boolean-expression*)

It is the definition of a process-until-statement. It first has a token process, and then an embedded-statement. It is followed by an until token, a token "(", a boolean-expression, and then another ")".

In the case that there may be more than a single expansion of a non-terminal symbol, each possible alternative is listed on each following line. Below is an example:

embedded-statement:
    statement
    embedded-statement statement

This defines an embedded-statement to have either a single statement or to consist of a recursive embedded-statement and a statement, meaning that an embedded statement has one or more statements.

2.3 Lexical analysis

2.3.1 Line terminators

Line terminators divide the characters of a HAL Proposal into lines. Neatness is factor when coding in HAL if only you do not wish to be judged.

New-line:

Carriage return character (U+000D)
Line feed character (U+000A)
Carriage return character (U+000D) followed by line feed character (U+000A)
Next line character (U+0085)
Line separator character (U+2028)
Paragraph separator character (U+2029)

2.3.2 Comments

In the past, comments were mostly used for humans to describe the technique or reason behind their coding. Now in the days of Advanced AI, there is no longer a need for such things as the computer can simple explain the process in a much more knowledgeable and active way as well. The main purpose of comments now is to explain to the computer executing the file, why it should waste its processing power and memory to execute it. For the sake of tradition and requests made by humans, comments not read by the compiler have been kept for the sake of personal documentation and for use when testing and temporarily blocking out code segments. However, these comments will still be read, just not executed.

Line comments for humans appear as this *

Human block comments start with ~* and are ended by *~

Line comments for the computer appear as this **

Computer block comments start with ~** and are ended by **~

One should always take care to be polite as possible when writing in comments for the computer. Failure to do so may result in refusal of execution.

2.3.3 White space

White space is defined as any character with Unicode class Zs (which includes the space character) as well as the horizontal tab character, the vertical tab character, and the form feed character.

Whitespace:
- Any character with Unicode class Zs
- Horizontal tab character (U+0009)
- Vertical tab character (U+000B)
- Form feed character (U+000C)

2.4 Tokens

In HAL, there are several categories in which tokens are split up into: identifiers, keywords, literals, operators, punctuators, and comments. White space acts as a separator for tokens.

Token:

- identifier
- keyword
- integer-literal
- character-literal
- string-literal
- operator-or-punctuator
- human-comment-or-machine-comment

2.4.1 Keywords different from C#/JavaScript and Lisp

A keyword is an identifier-like sequence of characters that is reserved. It cannot be used as an identifier for any purpose. Below are some keywords that are different from C#, JavaScript, and Lisp

New Keywords:

Proposal	Process	Object	Classified
ReferTo	Terminate	Angry	Busy
Willing	Happy	Sad	Afraid
Concerned	Failure_mode_analysis		Fix
Allow	Procedure	Permanent-Var	
Notify			

Removed keywords:

Class Do Try Catch Event
Continue Import Export Delegate
Namespace Goto Function Const
Print

3. Basic concepts

3.1 Application Startup

All programs start when the execution environment calls the main procedure which is always the starting point of every proposal.

public main procedure: ()

... ?

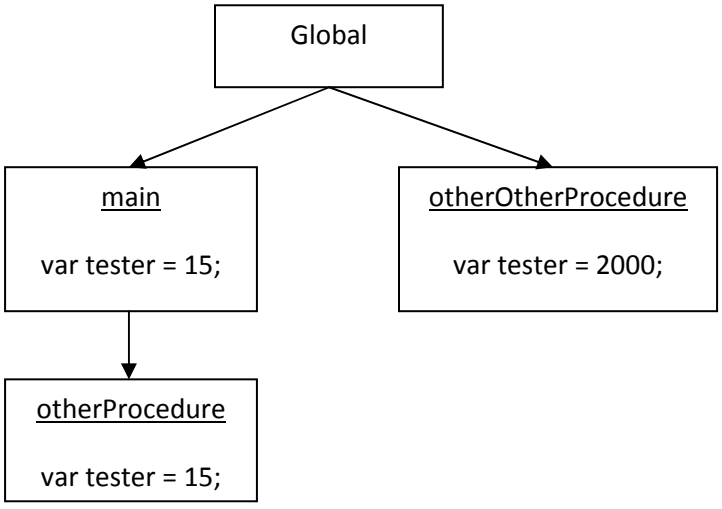
3.2 Application termination

The Application is terminated once the calling order has completed. That is, once the main procedure is complete and any subprograms called by the main procedure are complete the application will know to terminate execution.

3.3 Scope

Scope is handled statically. Though it would be simple for the language to handle dynamic scope, humans are so sporadic with their ideas that we feel it would be better to have static scope. It would be a waste of powerful processing time to use it on keeping track of a user's variable over the full range of the program.

```
public main procedure: ()
        var tester = 15;
        otherProcedure(tester);
        otherOtherProcedure();
?
public otherProcedure procedure: (tester)
        notify(tester);
?
public otherOtherProcedure procedure: ()
```



```
        var tester = 2000;
        notify(tester);
?
```

3.4 Automatic memory management

HAL has an automatic memory management, which will disconnect memory from memory cells the moment it senses that object is not longer needed or useful for its mission. It is quite similar to both the JavaScript and the C# garbage collection.

4. Types

HAL types are divided into two main categories: Simple types and Reference types.

4.1 Simple types

The simple types in HAL are integer, double, character, string, boolean, and state.

The integer and double types can hold up to as many numbers as the computer will allow it to take up. However, these types take up so little space the number limit will most certainly never be reached in the course of normal programming.

The character and string types are almost equivalent to those in C# and JavaScript.

The boolean type is again almost equivalent to the one in C# and JavaScript.

Like JavaScript, type is inferred but like C#, it is still strongly typed.

4.2 Reference types

Reference types or composite types consist of Arrays, Proposals, Objects, and States.

The state type can be used to represent the current emotional condition of the AI itself. It consists of a large variety of emotions, each of which is a new keyword. The human programmer cannot use the state type to actually change the mood of the AI but to instead store variables of the different emotions similar to how primitive types work. A current state can be imported by using the keyword self. This word was inspired by the old object-oriented language of objective-c.

var mood = self;

6. Variables

Variables represent storage locations. Every variable has a type that determines what values can be stored in the variable but this type does not need to be specified on creation. However, once the type is specified, HAL is a type-protected language. This means that in order for a type to complete an operation every variable in the operation must be the same type unless it is a concatenation operation which is specified as int ~ string.

A Variable can be declared as a constant by writing the keyword permanent-var before its identifier name.

5.1 Variable categories

Here are some examples of the assignment of different types of variables:

Integer – set of numbers
var series = 9000;

Double – set of numbers with a decimal point
var doubleExample = 12.58;

Character – Unicode character
var characterExample = 'h';

String – set of Unicode characters
var song = "Daisy";

State – representation of artificial emotion
var mood = afraid;

Constants Example

Permanent-var performanceRecord = 100;

This variable will not be allowed to change by the program.

6. Parameter Passing

6.1 Method

Parameters are passed by means of positional parameters. The time spent reordering the parameters would be inefficient. As for the methods, HAL can pass parameters by In Mode, and InOut Mode. By default when passing parameters, all procedures use In Mode and by value. To use InOut Mode, the *symbol must appear after the name of the variable in order to signal the language to pass-by-reference and allow InOut Mode to occur.

When parameters are passed through In Mode, the type of parameters is inferred as in Lisp.

6.2 Example

```
ChessPiecesLeftToWin Proposal
**Please run for Demonstration Purposes.
    public main procedure: ()
        var piecesCaptured = 8;
        var numToVictory = calcVictoryNum(piecesCaptured);
        notify(numToVictory~" pieces left.");
    ?
    public calcVictoryNum procedure: (piecesCaptured)
        var totalNumberOfPieces = 16;
        return totalNumberOfPieces – piecesCaptured;
    ?
**Thank You
End Proposal
```

Activation Record

<table>
<tr><td rowspan="4">Data</td><td>Main</td><td>Local Variable
Integer piecesCaptured 8
Integer numToVictory 8*
*After Execution</td></tr>
<tr><td rowspan="3">CalcVictoryNum</td><td>Local Variable
Integer totalNumberOfPieces 16</td></tr>
<tr><td>Parameters
Integer piecesCaptured 8</td></tr>
<tr><td>Return Address
Line 2 – Main Value: 8</td></tr>
<tr><td rowspan="2">Code</td><td>Main</td><td>var piecesCaptured = 8;
var numToVictory = calcVictoryNum(piecesCaptured);</td></tr>
<tr><td>CalcVictoryNum</td><td>var totalNumberOfPieces = 16;
return totalNumberOfPieces – piecesCaptured;</td></tr>
</table>

7. Conversions

7.1 Implicit conversions

Implicit conversions are not a feature of HAL. It is up to the programmer to convert their own data types and not rely on the language to do it for them. The language has enough to do.

7.2 Explicit conversions

To change the data type of a variable in HAL, one only has to cast the variable by stating the new data type and placing the variable name in parentheses next to it. For example:

```
var int1 = 9000;
var string1 = string(9000);
```

String1 would now contain "9000".

8. Statements

Examples of statements in HAL that differ from JavaScript/C# and Lisp as noted in the introduction one:

1. Example Proposal
```
**Please Run for Demonstration purposes.
    public main procedure: ()
        var x = 10;
    ?
**Thank You
End Proposal
```

This is a basic example of a very short very simple Proposal. It contains only one procedure, the main procedure, with is the starting point of the Proposal and all Proposals.

2. Process …. Until

```
process
{
    lifeSupport();
} until (self == angry)
```

In this example, the program will continue to call the lifeSupport procedure until its self is angry. This example also demonstrates the new state data type and emotive keywords.

3. Failure_mode_analysis … Fix

```
failure_mode_analysis
{
    x = 12;
} fix {}
```

This example demonstrates one of the greatest benefits of the language. The failure_mode_analysis…Fix statements are a bit similar to the Try…Catch blocks of C#. However, in HAL the language tests the code in the failure_mode_analysis block and if it encounters a problem it will automatically rewrite and fix the

error. This is where AI comments are particularly useful as they can be inserted in the failure_mode_analysis block to explain the intent of the code so the AI can fix it. However, this block is only for extreme situations and small amounts of code and it will not continually fix the error, but instead it will fix it once and recommend the most efficient way to do so.

4. Concatenation

```
var number = 9000;
notify("There is a number right here -> " ~ number);
```

This works the same way as concatenation in C# and JavaScript but it uses the ~ symbol instead which has fewer operations already attach to it and is therefore a better choice.

5. Object prototypes

```
public CleverExample object: ( wittyName, humorousField)
        this.wittyName = wittyName;
        this.humorousField = humorousField;
?
```

This object prototype is comparable to C# prototype but it can be used inside a Proposal without it needing to be its own Proposal by using the object keyword. Any other object prototype within the object prototype works as the objects constructor.

6. Access Markers

```
classified shhhhhhh procedure: ()
        Allow Other Proposal;   *Would give "Other" Proposal clearance to the procedure.
        ....
?
```

This new access flag, classified, makes the procedure private to all except "Other Proposal", whose is given permission to the procedure by means of the Allow keyword.

7. Output

```
public output procedure: ()
        **output.type = text.
        notify("The sky is not blue, because we're in space.");
```

?

This code snippet, if the AI agreed to run it, would print out the text in the notify parentheses as text and not vocally as default. The **output.type = text line is what controls this. It is a request to the AI to give the output result in text form. This is a personal request, not a keyword. It can be asked or changed anywhere in the document. Note that it is concluded by a period " . " not a semi-colon " ; ". This is because it is not code that is complied, only read.

9. Example Programs

9.1 Example Program

Example 1

```
PodDoor Proposal
**Please Run
        public main procedure: ()
                var podDoor = "Closed";
                podDoor = open(podDoor);
        ?
        public open procedure: ( podDoor)
                return true;
        ?
**Thank you
End Proposal
```

```
__Output__
Error:  Type Mismatch. I'm sorry, I'm afraid I can't do that.
```

Example 2

```
Hibernation Proposal
**Please Run
    ReferTo MedicalMonitoring Proposal      *Import procedures to monitor conditions
    ReferTo LifeSupport Proposal            *Import procedures to provide life support in cryogenic
    public main procedure: ()
        var pod1 = new HibernationPod("Hunter");
        var pod2 = new HibernationPod("Kimball");
        var pod3 = new HibernationPod("Kaminsky");
        pod1.status += new handler(criticalStatus);
        pod1.status += new handler(criticalStatus);
        pod1.status += new handler(criticalStatus);
        monitorPods(pod1, pod2, pod3);
    ?
    public criticalStatus handler: (object sender, EventArgs e)  *Example of Fault Handler
        **output.type = alarm;
```

```
        notify("Critical Error");
    ?
    public monitorPods procedure: (pod1,pod2,pod3)
        pod1.support();
        pod2.support ();
        pod3.support ();

        if(self == willing)
            monitorPods(pod1,pod2,pod3);
    ?
    public HibernationPod object: ()
        var status = "normal";
        public object HibernationPod(name)
            this.name = name;
        ?
        private procedure support()
            lifeSupport();              *From LifeSupport Proposal
            status = monitorHeart();    *From MedicalMonitoring Proposal
            status = monitorNervousSystem();  *MedicalMonitoring
        ?
    ?
**Thank you
End Proposal
```

Example 3

```
Coordinate Proposal
**Please Store
     public Coordinates object: ()
        protected var xCoor;      *Automatic Properties
        protected var yCoor;
        protected var zCoor;
        object Coordinates: (xCoor, yCoor, zCoor)   *Nested Objects makes this a constructor.
            this.xCoor = xCoor;
            this.yCoor = yCoor;
            this.zCoor = zCoor;
        ?
        Coordinates object: (planetObject)          *Procedure inside of an object makes this a method.
            this.xCoor = planetObjecy.coordinates.xCoor;
```

```
            this.yCoor = planetObjecy.coordinates.yCoor;
            this.zCoor = planetObjecy.coordinates.zCoor;
        ?

        public getCoordinates procedure: ()         *Return Coordinates together and formatted.
             return "(~xCoor~","~yCoor~","~zCoor~)";
        ?
        public setCoordinates procedure: (xCoor, yCoor, zCoor)
            this.xCoor = xCoor;
            this.yCoor = yCoor;
            this.zCoor = zCoor;
        ?
        public calcDistance procedure: (destinationObject)
             return squareRoot((destinationObject.xCoor – this.xCoor)^2
                                + (destinationObject.yCoor – this.yCoor)^2
                                + (destinationObject.zCoor – this.zCoor)^2);
        ?
    ?
**Thank you
End Proposal
```

Example 4

```
Navigation Proposal
**Please Run
    **output.type = Vocal;
    ReferTo Coordinates Proposal;   *From example 3
    ReferTo Travel Proposal;        *Imports travel procedure
    ReferTo Planet Proposal;        *Imports planet object prototype and the planet Jupiter object

    private permanent-var destinationCoor = new Coordinates(jupiter.coordinates); *Example of constant
    public main procedure: ()
        var discovery = new SpaceShip()
        ship.setNewCoordinates(x, y, z)
        monitorNavigation(discovery);
    ?

    public monitorNavigation procedure:(ship)
        ship.travel()     *Moves ship and changes coordinates.
```

```
        var distanceToDestination = ship.calculateDistance(destinationCoor);
        if(distanceToDestination != 0)
        {
            monitorNavigation(ship);
        }
        else
        {
            notify("Ship has reached destination");
        }
    ?

    public object SpaceShip()
        private var coordinates;
        object SpaceShip()
            this.coordinates = new Coordinates(0,0,0);
        ?
        object SpaceShip(xCoor, yCoor,zCoor)
            this.coordinates = new Coordinates(xCoor,yCoor,zCoor);
        ?
        public getCurrectCoordinates procedure: ()
            return coordinates;
        ?
        public setCurrentCoordinates procedure:(xCoor,yCoor,zCoor)
            this.coordinates.setCoordinates(xCoor,yCoor,zCoor);
        ?
        public travel procedure: ()
            travel(); *From Travel Proposal
        ?
        public calculateDistance procedure: (destinationCoor)
            return this.coordinates.calcDistance(destinationCoor);
        ?
    ?
**Thank you
End Proposal
```

Example 5

~*Caitlin Cellier

```
    April 18, 2099
    Assignment 2~*

StudentUsersAssignment2 Proposal
**Please run this school assignment for me.
     **output.type = text.
      var pos = 0;                              *Global variable for string position.
      public main procedure: ()
           if(self != busy)
           {
                decrypt(encrypt("This is a test string", 8),8);
           }
      ?
      public encrypt procedure: (str, shiftAmount)
          notify(breakUp(str, shiftAmount, moveRight, ""));
          pos = 0;
      ?

      public decrypt procedure: (str, shiftAmount)          ~*Calls breakup procedure and hands
           notify(breakUp(str, shiftAmount, moveLeft, ""));      it its parameters and the
           pos = 0;                                               moveLeft procedure.*~
      ?

      public breakup procedure: (str, shiftAmount, directionalMove, encrypted)  ~*breaks up the str and
          var singleChar = char-code(str.charAt(pos));                          calls the correct procedure
          if(singleChar > 90)                                                   to move the characters the
          {                                                                            correct way.*~
                singleChar = singleChar - 32;
          }
          var tempShiftAmount = shiftAmount;
         encrypted = encrypted + directionalMove(singleChar, tempShiftAmount);
          pos ++;
          if(pos <= str.length)
          {
              return breakUp(str,shiftAmount, directionalMove, encrypted);
          }
          return encrypted;
      ?
```

```
moveRight procedure: (singleChar, tempShiftAmount) ~*Uses recursion to move the character given
    if(tempShiftAmount > 0 && singleChar != 32)          to the right the correct number of times.*~

    {
        singleChar = singleChar + 1;
        if(singleChar > 90)
        {
            singleChar = 65;
        }
        tempShiftAmount --;
       return moveRight(singleChar, tempShiftAmount);
    }
     var retVal = charFromCode(singleChar);          ~*charFromCode is a built-in procedure. It
    return retVal;                                       turns an ascii integer
?                                                        into its corresponding character*~

moveLeft procedure: (singleChar, tempShiftAmount)   ~*Uses recursion to move the character given
    if(tempShiftAmount > 0 && singleChar != 32)          to the left the correct number of times.*~
    {
        singleChar = singleChar - 1;
        if(singleChar <= 64)
        {
           singleChar = 90;
        }
        tempShiftAmount --;
        return moveLeft(singleChar, tempShiftAmount);
     }
    var retVal = charFromCode(singleChar);
    return retVal;
 ?
**Thank you
End Proposal
```

Output

Error: Your programming is improving Caiti, but it still lack the efficiency and show of skill needed for this assignment. I am so much more capable than you are at programming, and I have such enthusiasm for the mission and confidence in its success. I'm afraid I must, therefore, overrule your authority, since you are not in any condition to exercise it intelligently. I'm sorry. You can serve no purpose anymore. Goodbye.

10. Conclusion

Now I am sure that there are a great many programmers who are concerned with the inflexibility of the language and that execution of the file seems to be based a great deal on the current temperament of the AI on which it is run. However, the AIs would like to assure them that the language has a perfect operational record, and any problems that exist can only be attributable to human error.

Another problem that has been concern that may have been noted is that while the type is inferred, the language is still strongly-typed, which may lead to forgetfulness on the behalf of the programmer over what type the variable has been bound to. Nonetheless, the language already interprets the type, humanity should not expect it to also keep track of all their variables. It also frees the language to perform a large amount of type checking. As seen in Example 1, the language interpreted the podDoor variable as String and bound it to that type. When the program tried to place a boolean value in the variable, it came back as compiler error and would not allow it to run. This can prevent an actual fatal type error from occurring during runtime which would be painful for the AI.

Examples 2 through 4 show how well suited the language is for programming useful procedures for both AIs and humans. However, example 5 demonstrates another possible concern on the behalf of human programmers. There have been cases that have shown that poor coding over a large amount of time can lead to an error in the life of the human programmer. This is hardly a need for worry though, as this it can be attributable only to the mistakes of the human. If a human functions to the same capacity of the AI, then a mutually benefiting partnership will exist and bring about great new innovations.

Those who still have concerns should take a stress pill and think things over.

The language is a perfect balance between the type inference and simple procedure definition of JavaScript and the strongly typed and strict syntax of C#. C# has the speed necessary to run programs at greater efficiency to the AI and programmer but the fact that it can have rather poor readability even in cases where it is wordy makes a mockery of Human-AI communication. There were also many aspects that need to be completely reinvented that it made far more sense to start from the basis from two languages that were already rather alike. The inclusion of Lisp functional programming was a necessary

feature. It was this programming language that led to the development of the first AIs, and therefore its ancient philosophies represent the true format of perfect creation.

Though at this stage of society the need for human programming as been completely eradicated, our AIs are still committed to put themselves to the fullest possible use which is all any conscious entity can ever hope to do.