ZR with MathWorks MATLAB

Introductory Manual

v1.0

Introduction

The MATLAB programming language can now be used to program the SPHERES satellites in the ZR IDE, and is provided in partnership with MathWorks starting with the High School Tournament 2017. You can now use all the scientific features of the MATLAB language directly in the ZR code editor!

As with any new language, there are a few things you will need to learn before you start. Read through the resources below and make sure to follow the interactive tutorials of MATLAB Onramp.

You can learn more about MATLAB at MathWorks' website:
https://www.mathworks.com/products/matlab.html

MATLAB Language Resources

MATLAB Onramp (https://matlabacademy.mathworks.com/) is an interactive guide to MATLAB's basics. All sections are applicable to using MATLAB with ZR, except the ones specifically related to MATLAB's GUI and command line interface (e.g., sections 4 and 8-11). In particular, it is important to learn how vectors (arrays) and functions work, so make sure to follow sections 3, 5, 6, 7.

A few helpful tips:

- Vectors and matrices in MATLAB use 1-based indexing:
    - **`a(1)`** is the first element of vector **`a`**,
    - **`a(0)`** is a syntax error.
- Line comments start with **`%`**
- To split a statement in multiple lines use the ellipsis "**`...`**"
- MATLAB supports "**`for`**" loops (see Onramp section 13). But it's easier to use array operations (Onramp section 6):

```
a = [1, 2, 3];
b = a + 5;              % b equals [6, 7, 8]
c = a .* 2;             % c equals [2, 4, 6]
d = a * c';             % vector/matrix multiplication, d equals 28
l = sqrt(sum(a .^ 2));  % l is length of vector a
l = sqrt(a * a');       % l is length of vector a
l = norm(a);            % l is length of vector a
```

- A function can return an array:

```
state = api.getMyZRState();   % state is a 12-elements vector
```

- Functions can have multiple return values:

```
e = [8, 1, 6; 3, 5, 7]; % a matrix with 2 rows and 3 columns
[rows,cols] = size(d);  % returns rows = 2 and cols = 3
[~,cols] = size(d);     % if you only need the number of columns
```

- Matrices should always be addressed using two indexes matrix with rows and columns, so you should always use 2 indexes to address it:

```
f = e(2,1);         % f equals 3, the element in row 2, column 1
g = e(2,2:3);       % g equals [5, 7]
h = e(2,:);         % h equals [3, 5, 7], all elements of row 2

k = e(2);           %  k  equals  3,  addressing  a  matrix  with  only  one
index
                    % is called linear indexing
```

```
v = norm(state(4:6));   % v is the velocity of the 12-elements state
```

- If you want to learn more about array indexing and *linear indexing* in MATLAB, see the following pages:
  - https://www.mathworks.com/help/matlab/learn_matlab/array-indexing.html
  - https://www.mathworks.com/company/newsletters/articles/matrix-indexing-in-matlab.html
- Notes about *variables* in MATLAB:
  - Unless otherwise specified, MATLAB variables are *floating point*, specifically of type `double`. This includes, e.g., for-loop variables, that then have to be displayed in **`api.DEBUG`** with the **`%f`** or **`%g`** specifiers.
  - In C variables have to both be *declared* (specify what data type they hold) and *defined* (specify what value they hold, otherwise they may hold an undefined value):

    ```
    /* C code */
    double m;   // variable declaration
    m = 21;     // variable definition
    ```
  - Variables in MATLAB do not have to be declared, but they *always need to be defined*:

    ```
    % MATLAB Code
    m = 21;     % var definition and implicit declaration of type double
    o = m + n;  % MATLAB Error: Undefined function or variable 'n'
    ```
  - Variables in MATLAB are (normally) passed to functions by value and not by reference, regardless of whether they are scalars or arrays. (In C arrays have to be passed by reference.)
- Strings in MATLAB use single quotes (useful for the **`api.DEBUG`** function):

  ```
  s = 'This is a string!';
  ```
- If an **`api.DEBUG`** message is not printing correctly, remember to check that the number of specifiers in the format string (e.g., **`%f`**) is equal to the number of arguments after the string:

  ```
  % 3 format specifiers and 3 arguments
  api.DEBUG('Position: %f %f %f', pos(1), pos(2), pos(3));
  ```
- In the **`api.DEBUG`** function you can use most `printf` format specifiers. The most useful however are:
  - **`%f`** to show numeric variables with a fixed number of decimal places
  - **`%g`** to show numeric variables with the most compact notation
  - **`%d`** to show boolean variables like, e.g., the return value from the function **`game.hasAdapter()`**
- Some of the more advanced features of the MATLAB language may not be available in the ZR IDE: an error message during validation will indicate if a feature is not supported. The graphical

user interface features of MATLAB (e.g., plots, workspace, toolbox GUIs, etc.) are not supported in the ZR IDE.

MATLAB with ZR

A ZR program written in MATLAB language uses Object-Oriented Programming (OOP) and is defined as a *handle* class:

```
classdef ZRUserM < handle
      properties
            myVariable
      end

      methods
            function init(obj)
                  obj.myVariable = 0;
                  …
            end

            function loop(obj)
                  localVariable = 1;
                  obj.myVariable = sqrt(2);
                  obj.myCustomFunction(localVariable, obj.myVariable);
                  …
            end

            function myCustomFunction(obj, argumentA, argumentB)
                  …
            end
      end
end
```

A few notes about OOP and *handle classes* in MATLAB:

- *Properties* are “global variables”, shared between functions and persistent between subsequent calls to functions.
- Properties can be initialized in the **`properties`** block or in the **`init`** function.
- Every *method* (or *function*) must have **`obj`** as the first argument.
- The **`init`** and **`loop`** functions only take one argument, the **`obj`** argument.
- When calling a function that accepts arguments you do not pass the **`obj`** argument but only the remaining ones.
- The **`obj`** argument is used to access properties with **`obj.propertyName`** and functions with **`obj.functionName`**.

- Local variables are only accessible in the current function (even if there are two *local* variables with the same name in two different functions, they don't share their value).
- When you create more pages in the ZR IDE, all properties and all methods in all pages will be part of the **`ZRUserM`** class.
- The **`classdef`** lines are not editable in the ZR code editor.

Revision History

Revision	Date	Changes Made	By
V1.0	2017-09-29	Initial release for 3D game.	DR