[12pt]article [russian]babel

document Game Maker Language (GML) is an interpreted programming language designed for use with the program for the development of computer games called Game Maker. Initially, the language support was implemented in Game Maker by mark Overmars to Supplement the system keypad events, however, later all the button events were included in GML, allowing the programmer to avoid the use of push-button functions. GML is very strongly associated with the environment of Game Maker. Game Maker is organized to avoid the need of manually programming such things as event management, level design and configuration objects. There is a misconception that GML supports insertion of snippets in other languages, such as Pascal, Assembler, or C++. The misconception arose from the partial similarity of the syntax of GML with Pascal and C++. (For example, the operator "&&" can be replaced with "and").

Library

In Game Maker a set of button events forms library. In the program interface library are displayed as tabs, which are different event icons. Each such event is a GML script or function that the user can use in the game. In the supply of Game Maker includes several standard libraries that contain basic events that are used in most games. It is also possible to create your own libraries, using the Library Maker. GMS2 has a built in conversion mechanism of actions you set the "buttons" in GML code and back that allows beginners to move on to GML, and improves understanding of how a standard action. The syntax and semantics of GML

GML is structurally similar to the language your code blocks, function calls, variable assignments, operator syntax, and so on. GML distinguishes between statements and expressions. For example,

g ¡ 1;

is not the right operator and will cause an error. Also, variable assignment is always a statement, and therefore cannot be used in expressions. For example, the following line would always generate an error because it would have calculated a nested expression as true or false, and then compare the Boolean result with the string "Yes" (incorrect comparison):

if ((answer = get$_s tring$("$YesorNo$?","")) == "Yes")

It is worth remembering that the equal sign "=" is an assignment operator and a Boolean operator in expressions, whereas in C++ in expressions write the double "==". However, a double equal sign "==" is interpreted correctly in the case of using it in expressions. The use of this sign as an assignment operator will cause a run-time error. GML also supports the increment:

g++; // supported as of Postfix and prefix entry

and

g += 1;

the same thing that

g = g + 1;

There are also operators: -=, *=, /=, —=, &= and $\hat{=}$. Since GMS2 introduced support for ternary operator ?:. Operators in GML can be separated by a semicolon, however, this is not a requirement (although it may fail in some specific cases).

Function

Game Maker contains an extensive library of built-in features to ensure basic functionality. The programmer can create their own scripts that are invoked in exactly the same way as functions. Drawing functions in Game Maker uses Direct3D API. If necessary, Game Maker also allows you to call native code of the platform by means of extensions (. DLL on Windows, Java on Android, JS, HTML5, etc).

Variables

Normally, GML does not need to pre-declare a variable, as is done in some other languages. Variable is created automatically after assigning them any values:

foo = "bar";

In Game Maker there are many built-in variables and constants. Each object instance contains many local variables such as "x" and "y". There are also several built-in global variables, like "score". These variables exist independent of object instances. These variables do not contain the prefix "global.", unlike the global variables specified by the programmer. One-dimensional and two-dimensional arrays are also supported.

In GML supports the following data types:

string (string) - a sequence of characters enclosed in single or double quotes (starting with GMS2, you should use double quotes).

real (number) - integer or floating point. Although all the values created in GML, are stored as a floating-point number double-precision, to work with extension you can use other types.

array (array) - variable using indexes to access elements. Can contain any data - numbers, strings, other arrays, descriptors, and other data structures, etc. They can also be passed as a parameter to the function, and they can be returned by the function as a result.

boolean - can be either true or false. Keep in mind that currently, GML does not support "real" Boolean value, and actually takes as false any number less than 0.5, and all that is equal to or greater than - true.

pointer (pointer) - pointer to the memory area. Used in some specific functions like buffer get address, etc.

enum (enumeration) is a user-defined collection of constants stored in a variable.

undefined (not set) is a special value that is returned in cases where the requested data is not found.

Scope of variables

Although GML and can be considered as an object-oriented language, the nature of objects and instances of objects in Game Maker creates some important differences in the way of differentiation variables. There are two types of locality: locality in object and locality in a script (or other piece of code contained in a separate container). That the variable is local to the instance of the object means that the variable is bound to a specific instance of the object and outside of this instance can only be used with a prefix that defines this instance. the fact that the variable is local to a script means that this variable can only be used in this script (and destroyed after the end of the script). Hereinafter, the term "local" will mean the locality object. By default, the variable is local to the object, but not local to the script in which it is used. In order to make the variable available to all instances of objects may be defined using the global namespace:

global.foo = "bar";

There is also the possibility to declare global variables using the globalvar keyword:

globalvar foo, bar;

But this method should be avoided, as this can easily lead to errors difficult to identify, due to the intersection of the scopes of variables (the same recommend directly the developers and GMS; moreover, it is possible that in the future, this keyword will be completely removed from language - at the moment it is left only for reasons of backward compatibility). In order to make the variable local to the script, it should be defined as:

var foo, bar;

Scope of a local variable is a script inside which it is declared. This implies that context switching (using with) it will still be available. For example:

var foo = "bar";

with other

show$_message(foo); //variable foo available$

Access to local variables of the object can be obtained using the instance ID of the object as the prefix instance.varname but, nevertheless, so it is impossible to get the local variables of one script from another, until they are passed as parameters to functions. Current namespace of the object can be changed through design "with". For example, the following script, when placed in a collision event, will destroy another instance of the object involved in this event (note that in the event of a collision Game Maker automatically sets the other variable of the second instance of the object with which the collision occurred): with other

instance$_destroy$();

Memory allocation GML automatically allocates memory for variables on the fly, and uses dynamic types, so assigning variable values of different types is also possible. For example, you can first create an integer variable and then change it to a string: intNumber = 1; intNumber = "This variable now contains the string"; In GML there is no special functionality that allows to release the memory occupied by a variable, but if necessary, you can assign a variable a new value, smaller size. For example, if you have a variable that stores large text, then setting the variable value of an empty string, it is possible to achieve memory. The same applies to arrays: data = [1, 2, 3, 4, 5]; // created the array (this syntax for creating arrays is available starting with GMS2) data = 0; // destroy the array (now it's just a variable) When the object is destroyed it also destroys all variables local to it, and any global variables exist independently of them.

Therefore, preference should be given to local variables, and global variables be used only in case of real necessity. To store large amounts of information more efficiently in Game Maker has support for several data structures such as stack, queue, list, map, priority queue and mesh. These structures are created, modifierade and destroyed through built-in functions. Also there are functions in almost all structures for sorting data in them. In some cases, more convenient and more efficient to use buffers that can store arbitrary data and which is, in fact, just the allocated pieces of memory. Objects and resources In work-based Game Maker resources are unique identifiers that serve to identify a particular resource or object instance. These identifiers can be used by scripts or functions to specify the required resource. Since the creation of resources directly in Game Maker involves specifying a name, this name serves as a constant containing the ID of the resource. The identifier of a particular instance is stored in the local variable "id". When dynamically creating resource always returns the ID of the newly created resource that can be used in the future.