**Goo scripting language description:**

1. Adding a script to an object does nothing. In order to use it, first you need to prompt the Goo Interpreter to translate your script into events with “translate()” method like this:

objects.bob.scripts.new\_script.translate();

1. All instruction in the first level of the script are translated into events that will be executed every iteration – so called default event of the object. Each if statement in the script creates a conditional event – instructions placed inside these statements are executed only if all conditions in the if statement are fulfilled. Nested if and else statements, while also being translated into different events, are connected with each other and will be executed in the order given in the script – using interrupts if needed.

Using **event(string)/dormant(string)** statement you can create dormant events with ids set to given string. Dormant events won’t be executed in every loop nor in a reaction to normal triggers. You can trigger them manually in the terminal or in a different event using run() method like this:

objects.bob.events.sleepy.run();

1. Engine can halt the execution of an event if the infinite loop is detected. If needed it can also deactivate a dangerous event – deactivated event cannot be triggered by other events. In order to activate such event, use methods: “run()” or “elevated\_run()”. The second method forces the engine to ignore dangerous behavior.
2. Most methods or accessors need a context based on one of these templates:

[subject].[method/variable] – accessing methods and variables of the subjects: layers, cameras, mouse and env(environment);

cameras.[camera\_id].[method/variable];

layers.[layer\_id].[method/variable];

layers.[layer\_id].[object\_id].[method/variable];

layers.[layer\_id].[object\_id].[module].[method/variable];

layers.[layer\_id].[object\_id].[module].[element\_id].[method/variable];

For example:

layers.first.patrick.var.apples = 10; //In the “first” layer find an object with an id “patrick” and set its variable named “apples” to 10.

You can shorten “layers.[layer\_id].[object\_id]” using built-in variable **objects** like this:

objects.[object\_id].[method/variable];

objects.[object\_id].[module].[method/variable];

objects.[object\_id].[module].[element\_id].[method/variable];

When accessing the owner of the script, you can use built-in variable **me** like this:

me.var.money += 50;

Under the hood, accessing the context by id in this way uses aggregation command **first**:

cameras.[camera\_id] 🡸🡺 first(cameras.id == [camera\_id]);

layers.[layer\_id] 🡸🡺 first(layers.id == [layer\_id]);

objects.[object\_id] 🡸🡺 first(objects.id == [object\_id]).

1. Context can be assigned to a variable using **let** statement:

let a = layers.background;

let bob = objects.bob;

let money = objects.bob.var.money;

1. In order to simultaneously access the context of a group of objects use aggregation operators: first, last, all, random. Aggregation templates:

[aggregation]([subject].[…].[method/variable] [comparison] [value]);

[aggregation]([subject].[…].[method/variable] [comparison]… [subject].[…].[method/variable]);

You can assign aggregated context to a variable using **let** statement.

Examples of aggregation:

first(id == “bob”).var.money = 10; //This line does exactly the same thing as this line: “objects.bob.var.money = 10”;

all(group == “tree”).events.spawn\_apples();

let first\_bob = first(group == “bob”);

let last\_bob = last(group == “bob”);

let all\_bobs = all(group == “bob”);

let all\_rich\_kids = all(var.money >= 1000).

1. In order to repeat instructions or whole events use **while** loop. It will repeat everything in its scope as long as the condition inside it returns true. Examples:

while(x.var.bool==”true”){x.events.doSomething();}

let i = 0;

while(i < 10){x.var.int+=2; i--;}

1. If statements
2. User defined functions:

fun funny()

1. Operations bound to events are executed after being triggered. List of trigger sources:
   1. Terminal – with “run()” command;
   2. Events – with “run()” command;
   3. Time – based on main loop iterations, with condition checks: **each\_second**, **each\_minute**.
   4. Camera – its state and position, on\_camera\_touch().
   5. Keyboard – with on\_key\_pressed(short key), on\_key\_release(short key);
   6. Mouse – with on\_mouse\_move, on\_mouse\_pressed(short key), on\_mouse\_release(short key). Mouse can select a group of objects based on their position and approximated size;
   7. Objects – current state;
   8. Variables – with on\_variable\_change;
   9. Collision – collisions of objects in the same layer;
   10. Editable Text Fields – with isWritten(string text);
   11. Movement – with based on state, isMoving(string object\_id),
2. Functions that create new objects (clone() and create()) return pointers to newly created objects. For example:

let newBob = Layer.bobTemplates.bob123.clone(“newBob”);

newBob.var.type = 10;

newBob.event.prepareBasedOnType();

1. Script lets you create and destroy elements of every module in the object with methods new(string id) and destroy(string id), for example:

Objects.walter.particles.new(“snow”);

Objects.walter.particles.destroy(“rain”);

1. The order of actions in the engine loop:

-Events are put in objects

-primaryTriggerTypes in Events

-EventsLookupTable: updates when events are modified. Only updates vectors triggered by the right module.

-TriggeredObjects: is recreated every iteration, it is a base for main event loop.

1. QUESTION - Killing objects in a variable, decreases the size of this variable. Spawning new objects from the variable doesn't change the size of this variable. You need to aggregate objects to this variable again to change the size. Thus a question arises: if you re-aggregate a variable in a nested conditional statement, can you update the state of this variable outside the statement? Or should you ignore the changes?

QESTION – Should aggregation be calculated only in statements(if, all, first, last, random), only in instructions or in both?

**[script].****translate(string event\_id=”[script]”):**

Description: Method prompts the interpreter to translate selected script into an event named **event\_id** by creating a new event or updating existing one in the selected object. Default value of **event\_id** is the name/id of translated script.

Example: me.scripts.recursion.translate()

**[event].run(int n=1):**

Description: Method executes a chosen event **n** times, which is limited by the environmental variable **stack\_size**. Run() dynamically adds an object to the end of the **TriggeredObjects** list.

Example: me.events.recursion.run().

**[event].elevated\_run(int n=1):**

Description: Method executes a chosen event **n** times without any limitation and by ignoring all dangers like infinite loops, modification of system files or having sex with your mom.

Example: me.events.recursion.elevated\_run().

**[event].stop():**

Description: Method deactivates the event, making it indifferent to any triggers.

Example: me.events.eat\_bananas.stop().

**print(string text):**

Description: Command prints a provided text in the terminal. It’s the best tool for debugging!

Example: print(“Hello Goo!\n”);

**let [name] = [value]:**

Description: This statement assigns a literal or context to a new variable created in the script owner. Examples:

let banana = “”

**each\_iteration:**

Description: Code inside its scope is translated into executed during every iteration of engine loop. On the engine architectural level **each\_iteration** creates an event.

**each\_second:**

Description: Simple condition check triggered every second. You can simulate it by incrementing a variable every iteration and checking its value in a simple if statement.

Result: bool

Example:

   each\_second{ me.var.money ++; }

**isolated\_if:**

Description: this if statement checks only the state of the currently analyzed object.

Result: bool

Example:

isolated\_if( me.var.money > 64 ) {

me.var.money --;

}

**half\_if:**

Description: this if statement inherits functionality of **isolated\_if** and compares currently analyzed object’s state (including constants) to a one specific object at the time.

Result: bool

Examples:

half\_if( me.var.money > you.var.money ) { … }

half\_if( you.var.money == 90 && bob.var.gold < 45 ) { … }

**full\_if:**

Description: this if statement compares constants, currently analyzed object’s state, one specific object or even two different objects in every comparison.

Result: bool

Examples:

full\_if( you.var.money > bob.var.money) { … }

full\_if( me.var.money == 90 && bob.var.gold < jeff.var.diamonds

|| steve.var.cash < patrick.var.bread ) { … }

**first:**

Description: this conditional statement returns the first object

Result: bool

Examples:

half\_if( me.var.money > you.var.money ) { … }

half\_if( you.var.money == 90 && bob.var.gold < 45 ) { … }

last

forall

while

conflicts: next conditionals have to work on updated objects!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!