Instructions

power_off

Description

```
Exit the program.
```

reboot

Description

```
Restart the process. All boot triggered events will be executed in the next engine
iteration.
```

return

Description

```
Interrupt the execution of all events from the object.
```

break

Description

```
Interrupt the execution of the current scope.
```

continue

Description

```
Interrupt the execution of the current scope.
```

all / first / last / random

Syntax

aggregation_type source [context_list] [camera_id] [layer_id] [object_id] [module_type] [module_id] [attribute] [expression] [output]

Description

```
Return a context that fulfils the boolean expression and optional parameters.
```

Parameters:

- aggregation_type (string): “all” - returns whole context, “first” - returns the first context, “last” - returns the last context, “random” - returns one random context;
- source (string): “camera”, “layer”, “context”, “_” - if context ids are not provided, choose one of the highest types of abstractions. Each camera contains its attributes and a bitmap (screen/window), while each layer contains its attributes and objects;
- [context_list] (vector*string*) - context ids will be used when source is equal to “context” or “_”. Each context can have one of these types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”. Type of the context affects other parameters;
- [expression] (vector*ConditionClass*) - the list of locations of values and the list of operators that create a relationship between those values, which in turn results in a single boolean value. This whole expression is calculated for each entity separately and if it returns truth, entity is aggregated into the new context. But if expression is empty, this boolean is equal to true by default;
- [camera_id] (string) - if source is equal to “camera” or provided context is of a camera type, only the camera with this id can be aggregated;
- [layer_id] (string) - if source is equal to “layer” or provided context is of a layer type, only the layer with this id can be aggregated;
- [object_id] (string) - if source is equal to “layer” or provided context is of a layer or object type, only the object with this id can be aggregated;
- [module_type] (string) - if source is equal to “layer”, or provided context contains layers, objects or modules; only the module of this type can be aggregated;
- [module_id] (string) - if source is equal to “layer”, or provided context contains layers, objects or modules; only the module with this id can be aggregated;
- attribute (string) - if provided, aggregate this attribute from selected entities;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

index

Syntax

index *source* [indexes] [context_list] [attribute] [output]

Description

```
Take a list of indexes and returns a context with entities found by indexes.
```

Parameters:

- source (string): “camera”, “layer”, “context”, “_” - if source is equal to “camera” or “layer”, contexts from context_list are treated as indexes.
- [indexes] (vector*int/unsigned*) - the list of indexes used to find entities in vectors;
- [context_list] (vector*string*): “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value” - if source is not provided, the first id serves as the container for index search. The rest of ids, no matter the source, are treated as indexes - they have to be of “pointer” or “value” type.
- [attribute] (string) - if provided, aggregate this attribute from selected entities;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

sum

Syntax

sum *left right* [output]

Description

```
Take a pair of contexts and return the sum of these sets.
```

Parameters

- left (string) - id of the selected context. This instruction accepts all types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;
- right (string) - id of the selected context. This instruction accepts all types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

intersection

Syntax

intersection *left right* [output]

Description

```
Take a pair of contexts and returns the intersection of these sets.
```

Parameters

- left (string) - id of the selected context. This instruction accepts all types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;

- right (string) - id of the selected context. This instruction accepts all types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

difference

Syntax

difference *left right* [output]

Description

```
Take a pair of contexts and return the difference of these sets.
```

Parameters

- left (string) - id of the selected context. This instruction accepts all types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;
- right (string) - id of the selected context. This instruction accepts all types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

value

Syntax

value *expression* [output]

Description

```
Return a context made out of given literals or values found in different sources.
```

Parameters

- expression (vector*ConditionClass*) - the list of locations of values and literals. Each value has one of these sources: “context”, “camera”, “layer”, “object”, “variable”, “literal”, “on_boot”, “second_passed”, “key_pressed”, “key_pressing”, “key_released”, “any_key_pressed”, “any_key_pressing”, “any_key_released”, “mouse_moved”, “mouse_pressed”, “mouse_pressing”, “mouse_released”, “window_w”, “window_h”. If you choose “literal”, you can provide literals from these types: bool, int, double, string.
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

bool / int / double / string

Syntax

literal_type literals [output]

Description

```
Return a context filled with provided literals.
```

Parameters

- literal_type *string*: “bool”, “int”, “double”, “string”;
- literals (vector*VariableModule*): bool, int, double, string;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

random_int

Syntax

random_int literal *min max* [output]

Description

```
Return a random integer from the range provided in the pair of literals.
```

Parameters

- min (VariableModule): bool, int, double, string - literal, minimal value of the output;
- max (VariableModule): bool, int, double, string - literal, maximal value of the output;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

random_int

Syntax

random_int context *min max* [output]

Description

```
Return a list of random integers from the ranges provided in the pair of contexts.
```

Parameters

- min (string) - id of the choosen context. Context types allowed by this instruction are: “value” and “pointer”;
- max (string) - id of the choosen context. Context types allowed by this instruction are: “value” and “pointer”;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

find_by_id

Syntax

find_by_id *source* [context_list] [camera_id] [layer_id] [object_id] [module_type] [module_id] [attribute] [output]

Description

```
Return a context filled with entities which ids are equal to ones provided in
parameters. If no ids are provided, aggregate all possible entities.
```

Parameters

- source (string): “camera”, “layer”, “context”, “_” - if a context is not provided, choose one of the highest types of abstractions. Each camera contains its attributes and a bitmap (screen/window), while each layer contains its attributes and objects;
- [context_list] (vector*string*) - context ids will be used when source is equal to “context” or “_”. Each context can have one of these types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”. Type of the context affects other parameters;
- [camera_id] (string) - if source is equal to “camera” or provided context is of a camera type, only the camera with this id can be aggregated;
- [layer_id] (string) - if source is equal to “layer” or provided context is of a layer type, only the layer with this id can be aggregated;
- [object_id] (string) - if source is equal to “layer” or provided context is of a layer or object type, only the object with this id can be aggregated;
- [module_type] (string) - if source is equal to “layer”, or provided context contains layers, objects or modules; only the module of this type can be aggregated;
- [module_id] (string) - if source is equal to “layer”, or provided context contains layers, objects or modules; only the module with this id can be aggregated;
- [attribute] (string) - if provided, aggregate this this attribute from selected entities;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

let

Syntax

let *context output*

Description

```
Create a variable with provided id from the selected or last existing context or
overwrite the context of the existing variable with the same id. New variables
will exist only in the current scope.
```

Parameters

- context (string) - id of the context. Context can have one of these types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;
- output (string) - id of the new variable or the id of existing variable intended for a context overwrite.

clone

Syntax

clone *left_context right_context* [changeOldID]

Description

```
Clone the values and structure of the right context to the left context.
```

Parameters

- left_context (string) - id of the context. Context can have one of these types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;
- right_context (string) - id of the context. Context can have one of these types: “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar", “pointer”, “value”;
- [changeOldID] (bool) - if true, the left context will inherit the id of the right context (with the last number incrementation). True by default.

+, -, *, /

Syntax

operator left_context right_context [output]

Description

```
Return a context with a result of an arithmetic operation on the pair of contexts
(with numeric values).
```

Parameters

- operator (string): "+", "-", "*", "/";
- left_context (string) - id of the context. This instruction accepts these two context types: “pointer” and “value”;
- right_context (string) - id of the context. This instruction accepts these two context types: “pointer” and “value”;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

++, --

Syntax

operator context

Description

```
Increment or decrements the numeric value found in the context by 1.
```

Parameters

- operator (string): "++" - increment by 1, "--" - decrement by 1;
- context (string) - id of the context. This instruction accepts these two context types: “pointer” and “value”.

=, +=, -=, *=, /=

Syntax

operator left_context right_context

Description

```
Move a numeric value from the right context to the left context based on the
choosen operator.
```

Parameters

- operator (string): "=", "+=", "-=", "*=", "/=" - these work exactly like in c++;
- left_context (string) - id of the context. This instruction accepts these two context types: “pointer” and “value”;
- right_context (string) - id of the context. This instruction accepts these two context types: “pointer” and “value”.

in

Syntax

in *left_context right_context* [output]

Description

```
Return true if any entity from the left context occurs in the right context. In
case of numeric values, compare their values. For other entities like layers,
compare their ids, but only if both entities are of the same type.
```

Parameters

- left_context (string) - id of the context. This instruction accepts these two context types: “pointer” and “value”;
- right_context (string) - id of the context. This instruction accepts these two context types: “pointer” and “value”;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

new

Syntax

new *type* location/l [layer_id] [object_id] [size_1] [size_2] [new_ids] [output]

Description

```
Create new entities and returns a context with their pointers.
```

Parameters

- type/source (string): “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar" - the type of the new entities;
- [layer_id] [semi-optional] (string) - the id of the layer with the role of a container for new objects. Required only if the new entities are objects;
- [object_id] [semi-optional] (string) - the id of the object with the role of a container for new modules. Required only if the new entities are modules;
- [size_1] (int) - this is the number of new entities that will be created. If not provided, operation will try to get this number from size2 *parameter*;
- [size_2] (string) - id of the context of the type “pointer” or “value”. This is the number of new entities that will be created;
- [new_ids] (string) - the id of the context with the type “pointer” or “value”. This context provides the list of strings that will become the new identificators for new objects. You can provide any number of ids and if the number is too low, the last id will be repeated. Although, to ensure the uniqueness in the given container, indexing numbers will be automatically added to ends of repeated ids and incremented by one for every next repetition. The same automatic action will be taken if no new ids are provided;

- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

new

Syntax

new *type* context/c [destination] [size_1] [size_2] [new_ids] [output]

Description

```
Create new entities and return a context with their pointers.
```

Parameters

- type/source (string): “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar" - the type of the new entities;
- [destination] [semi-optional] (string) - id of the context with a container for new entities;
- [size_1] (int) - this is the number of new entities that will be created. If not provided, operation will try to get this number from size2 *parameter*;
- [size_2] (string) - id of the context of the type “pointer” or “value”. This is the number of new entities that will be created;
- [new_ids] (string) - the id of the context with the type “pointer” or “value”. This context provides the list of strings that will become the new identificators for new objects. You can provide any number of ids and if the number is too low, the last id will be repeated. Although, to ensure the uniqueness in the given container, indexing numbers will be automatically added to ends of repeated ids and incremented by one for every next repetition. The same automatic action will be taken if no new ids are provided;
- [output] (string) - giving an id to a new context creates a variable in the current scope or overwrites the context of an existing variable with the same id.

delete

Syntax

delete *context*

Description

```
Delete all entities provided in the context.
```

Parameters

- context (string): “camera”, “layer”, “object”, “text”, "editable_text", "image", "movement", "collision", "particles", "event", "variable", "scrollbar" - id of the context with entities selected for deletion.

bind

Syntax

bind *context source scripts*

Description

```
Add or remove paths to scripts from objects.
```

Parameters

- context (string) - id of the context with objects;
- source (string):
  - “context” or “c” - next parameter will require an id of the context with paths to scripts. Instruction will bind those paths to provided objects;
  - “literal” or “l” - next parameter will require a list of string values - script paths. Instruction will bind those paths to provided objects;
  - “remove_context”, “rcontext” or “rc” - next parameter will require an id of the context with paths to scripts. Instruction will remove those paths from provided objects;
  - “remove_literal”, “rliteral” or “rl” - next parameter will require an id of the context with paths to scripts. Instruction will remove those paths from provided objects;
  - “reset” - instruction will remove all binded paths from provided objects.
- scripts (string / vector*string*) - paths to the scripts, their purpose depends from “source” parameter.

build

Syntax

build *context* [reset]

Description

```
Translate all scripts binded to objects provided in the context and create events
based on those scripts.
```

Parameters

- context (string) - id of the context with objects intended for event building;
- [reset] (bool) - if true, before creating new events, instruction removes all events from provided objects.

env

Syntax

env *name variable* [value]

Description

```
Change the value of an environmental variable.
```

Parameters

- name (string): window_title, window_size, fullscreen, pixel_art, draw_text_borders, draw_hitboxes, ignore_distant, draw_only_visible, bitmap_layers_number, print_logical_evaluations, print_instructions, reservation_multiplier - the name of the variable selected for modification;
- variable (string);
- [value] (VariableModule): bool, int, double, string - the new value of the selected variable.

proc

Syntax

proc *name variable* [value]

Description

```
Change the value of a process's variable.
```

Parameters

- name (string) - the name of the variable selected for modification;
- variable (string): is_active, can_interact_with_user, is_rendering, draw_camera_borders, draw_text_borders, draw_hitboxes, ignore_distant, draw_only_visible, bitmap_layers_number, print_logical_evaluations, print_instructions, auto_print_stack, reservation_multiplier, window_pos, window_size, min_window_size, window_tint
- [value] (VariableModule): bool, int, double, string - the new value of the selected variable.

fun

Syntax

fun *context attribute* [[type] [value] ...]

Description

```
Execute a function for all objects in the context. You can add as many type-value
pairs as needed.
```

Parameters

- context (string) - id of the context with objects;
- attribute (string) - name of the function intended for execution;
- type (string): context, c, bool, int, double, string - type of the next parameter;

- value (string / vector*VariableModule*) - id of the context with values or a list of values.

load_bitmap

Syntax

load_bitmap *path name*

Description

```
Load a bitmap from a file path to the engine's RAM.
```

Parameters

- path (string) - path to an image file;
- name (string) - name for the new bitmap.

mkdir

Syntax

mkdir *path*

Description

```
Create a new directory in the filesystem of the host system. A new directory can
be created only under the directory of the engine's directory.
```

Parameters

- path (string) - path to an image file.

rm

Syntax

rm *path*

Description

```
Remove a file or an empty directory in the selected path.
```

Parameters

- path (string) - path to the file or directory.

rmll

Syntax

rmll *path*

Description

```
Remove a file or directory in the selected path using recurrency.
```

Parameters

- path (string) - path to the file or directory.

print

Syntax

print [delimeter] [[type] [value] ...]

Description

```
Remove a file or directory in the selected path using recurrency.
```

Parameters

- delimeter (string) - path to the file or directory.
- type (string): context, c, bool, int, double, string - type of the next parameter;
- value (string / vector*VariableModule*) - id of the context with values or a list of values.

load_text

Syntax

load_text *path text*

Description

```
Load a text from a file and move it into a string variable.
```

Parameters

- path (string) - path to a text file;
- text (string) - id of the context of the string type.

save_text

Syntax

save_text *path text*

Description

```
Saves a string value to a text file.
```

Parameters

- path (string) - path to a text file;
- text (string) - id of the context of the string type.