ComBat software scripting manual

Contents

Introduction

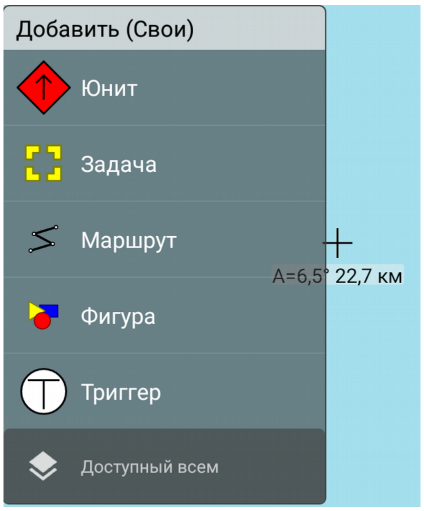
ComBat software has built-in scripting language and triggers concept for automation of military games or trainings.

Triggers are special map objects, which can be viewed only by «Game masters/Administrators» and are used to execute chain of commands (so called «**script**») after some condition appears. Condition can contain distance and azimuth between map items, their status, visibility, intersection with shape etc. Commands can change status, visibility, position of map items, send messages and orders to users etc.

At practice triggers can be used to automate processes like: imagine «mine» fields, warning about leaving game site or enemy approach, automatic respawn in some area with specific time delay, viewing next mission after completion of previous, capture the flag control and score count and much more. Special designed command set have huge potential and allows game masters to use ComBat system affectively.

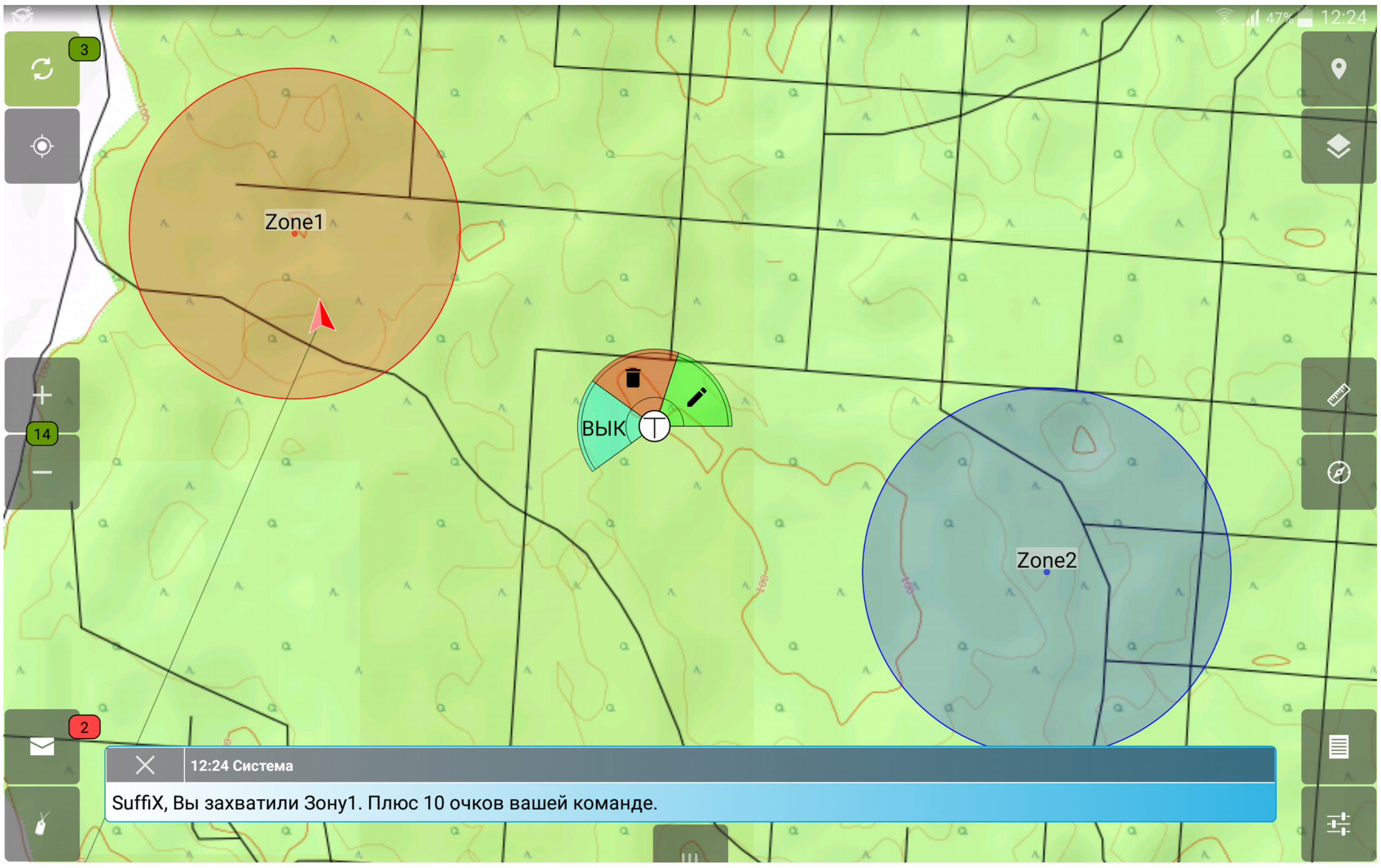
Adding trigger

Adding and viewing Triggers is allowed to «Game master»/«Administrator» users only.



To add Trigger it is necessary to tap on empty map space and choose «**Trigger**» in menu «**Add**».

As the result Trigger icon with sector menu appears on map.

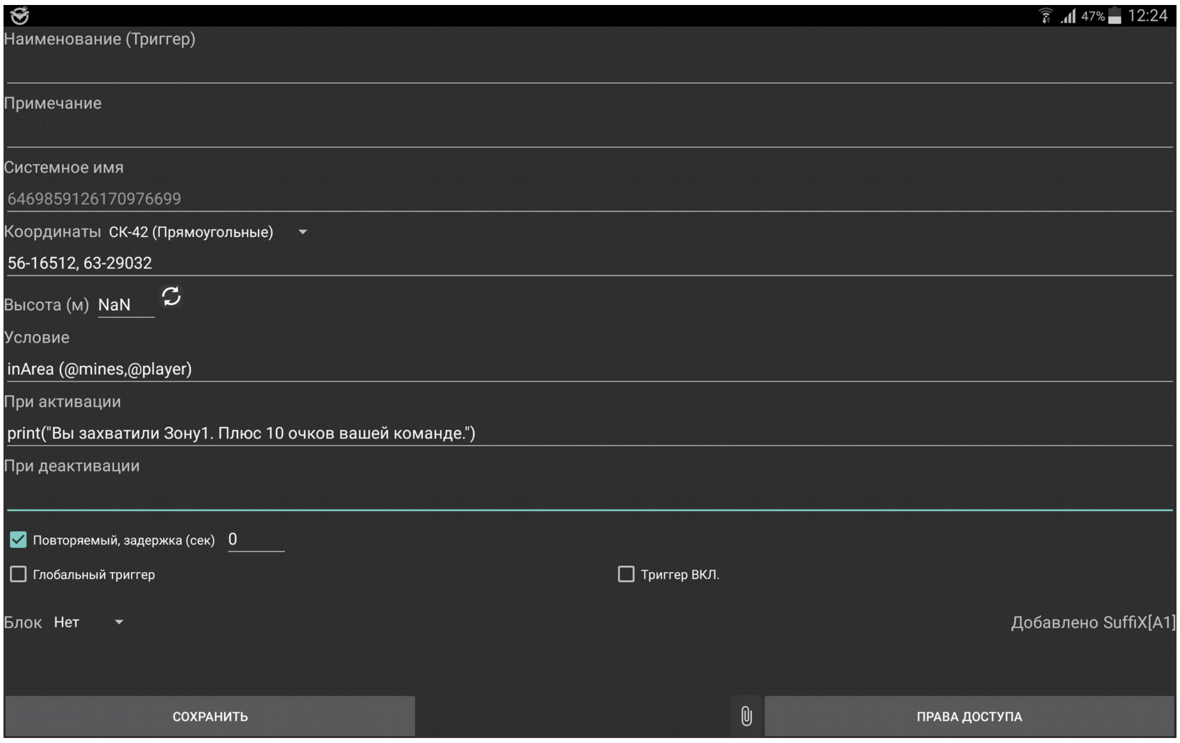


SuffiX, Вы захватили Зону1. Плюс 10 очков вашей команде.

Press green button «**Edit**» to start editing trigger attributes.

Button «**ON**» / «**OFF**» is designed to change Trigger state quickly during game event.

Press «**Delete**» button to delete Trigger.


Наименование (Триггер)

Примечание

Системное имя
6469859126170976699

Координаты СК-42 (Прямоугольные)
56-16512, 63-29032

Высота (м) NaN

Условие
inArea (@mines,@player)

При активации
print("Вы захватили Зону1. Плюс 10 очков вашей команде.")

При деактивации

☑ Повторяемый, задержка (сек) 0
☐ Глобальный триггер
☐ Триггер ВКЛ.

Блок Нет

Добавлено SuffiX[A1]

Each trigger has following attributes:

- **Caption** – visual name of Trigger on the map;
- **Info** – detailed description of trigger functionality;
- **System name** – reference name of Trigger to access it from the script;
- Activation **condition**, which consists of constructions described in this manual below;
- Script **on activation**, which executes if above condition become true;
- Script **on deactivation**, which executes when above condition become not true.

Trigger can be in two states – activated and deactivated. Activation occurs when condition become true. In this case state of trigger become activated and on activation script is executed. Script will not execute again until condition become false. When condition become false, then on deactivation script executes and trigger state become deactivated.

Triggers can be **repeatable** and **on time**. Repeatable trigger can switch state unlimited number of times, but one tie trigger can be activated only once. Repeatable trigger can have delay between activations. **Repeatable mode and delay** in seconds can be set in corresponding Trigger attributes.

Furthermore Triggers can be **local** and **global**. Local triggers are activating on each device independently, but global triggers send their activation state to other users and block activation on other devices. This function is switching by «**Global trigger**» option on the properties screen.

At last, Trigger can be **enabled** and **disabled**. Disabled triggers condition is ignoring by the system, as the result no on activation and on deactivation event occurs. Trigger status defined by «**Trigger ON**» flag or can be changed programmatically from the script (see below).

Script writing principles

Conditions can be constructed from the operations and functions described in the next article of this manual.

Script can contain commands from the list, described in the next article after operands and functions. Commands can be written at one line or with carriage return. Commands are split by ";". Last command in script or in block can have no separator at the end. Blocks of commands are separated by “{ }”. Separator may not be used after the end of block. Example:

If (getStatus(@miss1)==CMP) then { hide(@miss1); show(@miss2); say(@HQ,@Player,”Mission1 complete!”)}

System can have variables storage, which allow to store numerical, string variables, references to objects or their sets using variable name in storage. Storage variables can be used to measure past time or count amount of trigger activations, store intermediate position values, map item lists etc. All variables are technically of string type in storage: numbers are stored as strings; object references are stored as strings which started with «@» symbol; coordinates are stored as text; sets are stored as string delimited by “,” (see above). If *update(<@storage>)* command was not executed, then variables will be local on the current device. Example of variables assignment:

@storage~variable = <value>;

Example of variable usage:

if(@storage~variable == 2) then {};

Each Trigger has related Storage by default, where it stores state variables: **active** — Trigger state (TRUE — trigger is activated, FALSE — Trigger is deactivated), **activator** — numeric ID of user who activated Trigger, **executed** — non-repeatable Trigger execution flag (TRUE — already executed, FALSE — not executed yet) and **time** — time of last activation. This and other additional variables can be read or write passing trigger name or constant **@self** instead of storage name. Global trigger storage automatically marked to transfer variables to server after any variable change.

Program has ability to operate with sets defined in following way «@obj1,@obj2,...@objN». Sets are used to execute series of commands with several related objects. Except manual creation sets can be returned by **select**, **getUnits** and **getPoints** functions. Sets can be managed per item using **put** and **drop** functions. Command **count** returns the size of set.

Command **forEach**(<@set>) {<block>} allows to execute series of commands for each element on the set. Constant **@obj** will contain reference to current object during the foreach operation.

In case of syntax error, user will see it in the console activity of the application. The error log contains trigger name, operator name and other possible details. Console can be started from the mission menu button **«Console»**.

In case of functions cannot return value, it will return literal **NULL**.

Script constructions list

Logical operators

During condition construction following logical operators can be used:

- **&&** or **AND** - logical «AND»;
- **||** or **OR** - logical «OR»
- **«==»** - equals;
- **«!=»** - not equals;
- **«>=»** - greater or equal;
- **«<=»** - less or equal;
- **«>»** - greater than;
- **«<»** - less than;

Numbers are used to determine logical values as follows: all values more than 0 are TRUE, others are — FALSE. Literals **TRUE** and **FALSE** are interpreted by the system as 1 and 0.

Arithmetic operations

To calculate necessary formulas following arithmetic operations can be used:

- **«+»** - addition
- **«-»** - subtraction
- **«*»** - multiplication
- **«/»** - whole number division
- **«%»** - rest from the whole number division

Important! Script translator does not prioritize operations. To prioritize operators user must use "( )", then translator will start calculation from the most inner brackets. By default operations are processed left to right!

Example: 2+4*2 will equals 12 – addition will be done first, than multiplication. To calculate in classic way, user should write 2+(4*2). If you use calculated values in logical expression it is better to take operands in brackets, like (1+3) OR (2-4).

Conditional operators

To split script execution on branches there is following conditional operator exists:

If (<condition>) **then** {<block1>} **else** {<block2>};

If condition TRUE then <block1> executed, else <block2>.

Short version of operator can be used:

If (<condition>) **then** {<block1>};

Block of commands can be replaced with one command following by “;” smbol:

If (<condition>) **then** <command1>; **else** <command2>;

Constants

During scripting you will operate several entities, which is defined by constant values.

Global object references:

- **@self** – reference to trigger which process current script execution.
- **@player** – reference to user of current device.
- **@team** – reference to team of current user.
- **@leader** – reference to team leader of current user.
- **@side** – receiver type allowing to send message to all members of current user side (for messaging).
- **@all** – broadcast receiver type to send message to users of all sides
- **@obj** – reference to current iteration object in **forEach** loop**.**
- **@sel** – reference to object substitution comparable in **select** statement**.**

Global literals:

- **TRUE** – (1) expression is true;
- **FALSE** – (0) expression is false;
- **NULL** – value undefined.

Address type:

- **0** – specific user (by default);
- **1** – team leader;
- **2** – team members;
- **3** – team and sub-teams members down to hierarchy;

Item lock types:

- **NONE** – no lock;
- **OWNER** – locked by owner;
- **ADMIN** – locked by administrator.

Unit type:

- **EQUI** – equipment;
- **UNIT** – unit;
- **INST** – installation;
- **AVIA** – aviation;

Affiliation:

- **F** – friend;
- **H** – hostile;
- **U** – unknown;
- **N** – neutral;
- **S** – suspect;
- **A** – assuming friend;
- **P** – pending;

Unit size:

- **A** – team;
- **B** – squad;
- **C** – section;
- **D** – platoon;
- **E** – company;

- **F** – battalion;
- **G** – regiment;

Equipment mobility:

- **O** – whelled;
- **P** – cross-country;
- **Q** – tracked;
- **R** – wheel & track;
- **S** – towed;
- **T** – railway;
- **U** – snow;
- **V** – sled;
- **W** – aimals;
- **X** – barge;
- **Y** – amphibious;
- **Z** – unspecified;

User rank:

- **0** – Private;
- **1** - Corporal;
- **2** – Sergeant;
- **3** – Lieutenant;
- **4** – Captain;
- **5** – Major;
- **6** – Colonel.

Unit action type:

- **LOOK** – observing;
- **GO** – moving;
- **NONE** – do nothing**.**

Target action types:

- **BACK** - fall back into formation;
- **MOVE** - move;
- **ATTACK** - attack;
- **CAPTURE** - capture;
- **SPY** - research;
- **NONE** – at ease.

Task status:

- **ACT** – actual;
- **CMP** – completed;
- **FAIL** – failed.

User status:

- **ACTIVE** – on-line;
- **CLEAR** – sector clear;
- **DANGER** – in danger;
- **AMMO** – ammo low;
- **FUEL** – fuel low;
- **REPAIR** – need repair;
- **INJURED** – injured;
- **DEAD** – out.

Team status:

- **CLEAR** – sector clear;
- **DANGER** – in danger;
- **AMMO** – ammo low;
- **FUEL** – fuel low;
- **REPAIR** – need repair;
- **EQUIPMENT** – equipment lost;
- **CRITICAL** – critical losses.

Shape type:

- **RECT** – rectangle;
- **OVAL** – ellipse.

Multipoint object type:

- **PL** – polyline;
- **PG** – polygon.

Waypoint type:

- **0 –** Waypoint
- **1** – Coordination point;
- **3** – Named area.
- **4** – Start point.
- **5** – Casualities exchange point;
- **6** – Causalities collection point;
- **7** – Start of attack;
- **8** – Cover fire point;
- **9** – Ambush point.

Functions

User has access to following functions…

Logical and arithmetic:

not(<number>) - logical «NOT». If <number> <= 0 or FALSE, then return TURE, otherwise FALSE.

rnd(<number>) – generates random number from 0 to <number>. Parameter and result are decimal.

abs (<number>) – get absolute value of number.

round(<number>) – round <number> to whole.

Geographic:

inArea(<@shape>, <@obj>) – check if object is inside shape. Returns TRUE or FALSE.

getDist(<@obj1>,<@obj2>) – returns distance in meters between two objects.

getAngle(<@obj1>,<@obj2>) – returns azimuth (number from 0 to 359) from <obj1> to <obj2>.

getOffset(<position>,<distance>,<azimuth>) – returns new position shifted on specified distance and azimuth.

Object attriutes:

getClass(<@obj>) – returns object class (Example: Unit, Task, Shape etc.).

getId(<@obj>) - returns internal object ID.

getSide(<@obj>) – returns number identifier of game fraction. Identifiers can be viewed in application Settings.

getOwner (<@obj>) – returns ID of owner.

getModifier (<@obj>) – returns ID of user who changed object last time.

getTime (<@obj>) – return Unix time of last change in seconds.

getName(<@obj>) – returns system name of object (variable name).

getCapt(<@obj>) – returns object caption on the map (Icon caption).

getInfo(<@obj>) – returns detailed description of object.

getLock(<@obj>) – returns lock status (see constants).

getLayer(<@obj>) – returns layer ID where object is lied on.

getPos(<@obj>) – returns position string in internal format. To output position for user call posStr(getPos(<@obj>)).

getAlt(<@obj>) –returns altitude of object above the sea level in meters.

getDir(<@obj>) – returns unit azimuth of movement or observation, rotation angle of icon or shape from 0 to 359.

getAct (<@unit>) – returns unit activity type (see constants).

getType(<@obj>) – returns unit type, shape type, line point or waypoint type (see constants).

getAffiliation(<@unit>) – returns unit affiliation (see constants).

getIdentity (<@unit>) – returns unit identity (icon type, number value).

getUnitSize(<@unit>) – returns Unit size (see constants).

getMobility(<@unit>) – returns equipment mobility (see constants).

getRank(<@user>) – returns user rank (see constants).

getTeam(<@unit>) – returns ID of parent team of specified unit.

getLeader(<@unit>) – returns direct leader of unit.

getTeamLeader(<@team>) – return own leader inside team.

getUnits(<@team>) – returns set of units inside team.

getTarget(<@unit>) – returns unit target ID.

getTargetAction(<@unit>) – returns navigation order type (see constants).

getColor(<@shape>) – returns shape color (HEX string like #AARRGGBB).

getWidth(<@shape>) – returns shape width in meters.

getHeigth(<@shape>) – returns shape height in meters.

getPoints(<@line>) – returns points located in line, way or polygon.

getStatus(<@task>) – returns task status.

checkStatus(<@unit>) – checks if specified status is present in unit statuses. Returns TRUE or FALSE.

isExist (<@obj>) – checks if object exists on the map. Returns TRUE or FALSE.

isTeam(<@unit>) – checks if object is team. Returns TRUE or FALSE.

isUser(<@unit>) – checks if object is user. Returns TRUE or FALSE.

isPlayer(<@unit>) – checks if object is current user. Returns TRUE or FALSE.

isUpdated(<@obj>) – checks if object marked to transfer on server and still not transferred. Returns TRUE or FALSE.

isDeleted(<@obj>) – checks if object is marked for deletion. Returns TRUE or FALSE.

isShared(<@obj>) – checks if object is shared to all sides. Returns TRUE or FALSE.

isVisible(<@obj>) – checks if object or layer is visible on map. Returns TRUE or FALSE.

isEnabled(<@trigger>**) –** check trigger status: TRUE - enabled, FALSE - disabled.

isActivated(<@trigger>**) –** returns current trigger state: TRUE - activated, FALSE - deactivated.

Additional:

time(<hh:mm:ss>) – returns Unix time in seconds. Without parameter returns current time.

timeStr(<время в сек>**)** – format time for output like «hh:mm:ss».

dateStr(<время в сек>**) –** format date for output like «dd.mm.yy».

posStr(<координаты>**) –** format position for output according to user settings.

select("<condition>") – select object according to condition among all objects in the map, using @sel constant as reference to every next object during comparing. Example, select("getDist(@sel, @player) < 100") — selects all objects closer than 100 m to current user. **Warning!** Brackets around condition is obligatory.

count("<set>") – returns count of objects in set.

put("<set>",<@obj>) – add object to set and return resulted set.

drop("<set>",<@obj>) – remove object from set and return resulted set.

Commands

User can use following commands…

Messages:

log("<text>") **–** output text to debugging console available for game master/administrator. Expressions can be iserter in text closet in «[» ,«]». Example: Log("Your score: [val(@self.score)]")

eval(<expression>) **–** calculate and output single <expression> to console. Analog of Log(”[expression]”)

print (”<text>”) **–** Display local chat message on behalf of System. Expressions substitution also available hereusing «[» ,«]».

say (<@sender>, ”<set_of_receivers>”,”<text>”) – send message to specific user or set of users on behalf of sender. Special constants can be used instead of receiver, like @all, @side, @team, @leader, whicj allows to send message to all users in game, users of current side, team of sender or sender leader.

Object attributes:

update(<@obj>) – mark object as updated to transfer it on server. **Warning!** Do not forget tocall this command at the end of command series when you change object attributes, otherwise all changes remain local on you device.

delete(<@obj>) – mark objects for deletion from server and hide it from map.

share(<@obj>) – share object to users of all sides.

private(<@obj>) – make object private to own side.

setOwner (<@obj>,<@user>) – change object owner to specified user. Can be used with following object lock with OWNER type.

addAddress(<@obj>,<address>,<modified>) – adds object receiver.

remAddress (<@obj>,<address>,<modifier>) – remove receiver from object.

clrAddress (<@obj>) – remove all receivers from object (clear permissions filter).

show(<@obj>) – shows object or layer on the map.

hide(<@obj>) – hide object or layer from the map.

setName (<@obj>,<@name>) – sets object system name (variable name).

setCapt (<@obj>,<@caption>) – sets object caption (icon caption).

setInfo (<@obj>,<@text>) – sets detailed description of object.

setLock(<@obj>,<type>) – changes object lock status (see constants).

setLayer(<@obj>,<layer>) – changes layer of object to specified.

setPos(<@obj>,<@pos>) – moves object on map to specified position (internal position format).

setAlt(<@obj>,<alt>) – sets altitude of object above the sea level.

setDir(<@obj>,<azimuth>) – sets azimuth of icon rotation in degrees.

setAct (<@unit>,<@action>,<@azimuth>) – sets unit activity type (see constants).

setType(<@obj>,<type>) – sets type of unit, shape or point (see constants).

setAffiliation(<@obj>,<affiliation>) – sets unit affiliation (see constants).

setIdentity(<@unit>,<identity>) – sets unit identity (icon type, number value).

setUnitSize(<@unit>,<size>) – sets unit size (see constants).

setMobility(<@unit>,<mobility>) – sets equipment mobility (see constants).

setRank(<@user>,<rank) – sets user rank (see constants).

setOrder(<@target>,<action>,<@receiver>,<@sender>) – gives navigation order to receiver into target on behalf of sender with the specified order action type. Action types see in constants. User will receive message and navigation arrow.

addStatus(<@unit>,<status>) – adds specified status to user or team.

remStatus(<@unit>,<status>) – removes specified status from user or team.

drop(<@unit>) - drop unit from current team.

addUnit(<@team>,<@unit>) – adds unit into specified team.

remUnit(<@team>,<@unit>) – removes unit from specified team.

setLeader(<@team>,<@leader>) – set team leader in group.

setColor(<@shape>,<color>) – change shape color (HEX string like #AARRGGBB or name — red, blue etc.).

setWidth(<@shape>,<width>) – sets shape width in meters.

setHeight(<@shape>,<height>) – sets shape height in meters.

addPoint(<@line>,<@point>,<@previous>) – add point to lone after previous point.

remPoint(<@line>,<@point>) – removes point from line.

setStatus(<@task>,<status>) – changes task status.

enable(<@trigger>) – enables trigger.

disable(<@trigger>) – disables trigger.

activate(<@trigger>) – call on activated commands and change state to activated unrespecting condition.

deactivate(<@trigger>) - call on deactivated commands and change state to deactivated unrespecting condition.

reset(<@trigger>) - returns trigger to initial state (deactivated, not executed).

System:

clearVars(<@storage>) – clears all variables in storage.

clearMsg() – clears all chat messages history.

sound(<id>) – make sound from the library.

delay(<sec>) – perform specified delay before next command processing.

forceUpdate() – force synchronization with server to transfer all map changes (visibility, status, message etc.).

Examples

Simple example of logic. If score of side 1 is more than 100, then send message to every user of side 1 with text “Bingo! Victory, you have reached N score.”

Condition: *@self~side1_score >= 100*

On activation:

foreach(select(“isUser(@sel) and (getSide(@sel) == 1)))

say(@player, @obj, "Victory! You have reached [@self.side1_score] score.");

To check trigger, go to console and type:

@self~side1_score = 200;

Next more complex example is - **autorespawn**:

1) Create ellipse or rectangle of respawn with system name *resp*.

2) Create local trigger:

2.1) Condition: *inArea(@resp, @player)*

2.2) On activation: *print("You have reached respawn. Please wait 30 min for revival..."); delay(1800); remStatus(@player, DEAD); print("You are live again. Please, continue game.")*

One more variant – **mine field** with probability of execution 50%:

1) Create ellipse or rectangle of minefield with system name *mine*.

2) Create local trigger:

2.1) Condition: *inArea(@mine,@player)*

2.2) On activation: if(rnd(1) > 0.5) { print("You have exploded on mine! Game over."); addStatus(@player, DEAD) }

Automatic mission switch with player notification:

1) Rename game master user icon to "HQ".

2) Create task with name mission1. Put all necessary object on its layer.

3) Create task with name mission2. Put all necessary objects on it and hide it using flag "Temporary invisible from all".

4) Create local trigger:

4.1) Condition: getStatus(@mission1) == CMP

4.2) On activation: say(@player, select(“isUser(@sel)”),"Congratulations! You have successfully completed Mission1! New task receiver. See map..."); hide(@mission1); show(@mission2)

As the result if anybody set mission1 status to "Complete", then objects located on mission1 will be hidden and next mission will be displayed. All users will receive notification about this event.

Primitive **capture the flag**.

1) Rename game master user icon to "HQ" just for more beautiful messages.

2) Create several ellipses, rectangles or polygons of zones (naming them zone1,zone2...,zoneN). Make them green by default. Green zones means free, yellow – under fire, red – captured by red, blue – captured by blue.

3) Create global non-repeatable trigger with caption «Reset» and system name *var* to store score variables. When enabling this trigger the score and color of zones will reset to initial.

3.1) Condition: *TRUE* (always executed when enabled)

3.2) On activation:

```
@self~side_red = 157;

@self~side_blue = 156;

@self~zones = "@zone1,@zone2,@zone3";

@self~score_red = 0;

@self~score_blue = 0;

foreach(@self~zones) {

    setColor(@obj, #FF00FF00);

    setLock(@obj, ADMIN);

    update(@obj);

}

foreach(select("isUser(@sel)")) {

    setLock(@obj, ADMIN);

    update(@obj);

}

deactivate(@self);
```

4) Create local repeatable trigger with caption «Core» and time delay 60 sec (time of score calcualtion) which will analyze territory control.

4.1) Condition: *getName(@player) == Server* (works only on device with user name Server)

4.2) On activation:

```
forEach(@var~zones) {
    @self~count_red = count(select("isUser(@sel) AND inArea(@obj, @sel) AND
not(checkStatus(@sel, DEAD)) AND (getSide(@sel) == @var~side_red)"));

    @self~count_blue = count(select("isUser(@sel) AND inArea(@obj, @sel) AND
not(checkStatus(@sel, DEAD)) AND (getSide(@sel) == @var~side_blue)"));


    if ((@self~count_red > 0) AND (@self~count_blue > 0)) then {
        if(getColor(@obj) != #FFFFFF00) then {
            setColor(@obj, #FFFFFF00);
            update(@obj);
            say(@player, @all, "Heavy fight in [getCapt(@obj)]!")
        }
    } else if (@self~count_red > 0) then {
        if(getColor(@obj) != #FFFF0000) then {
            setColor(@obj, #FFFF0000);
            update(@obj);
            say(@player, @all, "[getCapt(@obj)] captured by RED!")
        }
        @var~score_red = @var~score_red + 1
    } else if (@self~count_blue > 0) then {
        if(getColor(@obj) != #FF0000FF) then {
            setColor(@obj, #FF0000FF);
            update(@obj);
            say(@player, @all, "[getCapt(@obj)] captured by BLUE!")
        }
        @var~score_blue = @var~score_blue + 1
    } else if(getColor(@obj) == #FFFFFF00) then {
        setColor(@obj, #FF00FF00);
        update(@obj);
        say(@player, @all, "[getCapt(@obj)] is free!")
    } else if(getColor(@obj) == #FFFF0000) then {
       @var~score_red = @var~score_red + 1
    } else if(getColor(@obj) == #FF0000FF) then {
        @var~score_blue = @var~score_blue + 1

    }

}

forceUpdate;

deactivate(@self);
```

The meaning - every 60 seconds all zones on the map are chacked on availability of users from red and blue teams Inside. If zone has players of both sides it become yellow and no one get score. If zone has users only from one side it will take color of this side and give one score to this side every 60 sec. Is zone was yellow and all players gone – it becomes green.

5) Create disabled local Trigger «Score» to show the results:

5.1) Condition: *TRUE*

5.2) On activation:

```
say(@player, @all, "Score - RED: [@var~score_red], BLUE: [@var~score_blue]!");

deactivate(@self);
```

On enabling it will send message with score to all players.