Руководство по автоматизации игр/учений с помощью ComBat

Оглавление

Введение 1
Добавление триггера 2
Принцип построения скрипта 4
Список скриптовых конструкций 5
- Логические операторы 5
- Арифметические операции 5
- Операторы условного перехода 5
- Константы 6
- Функции 8
- Команды 11

Примеры 13

Введение

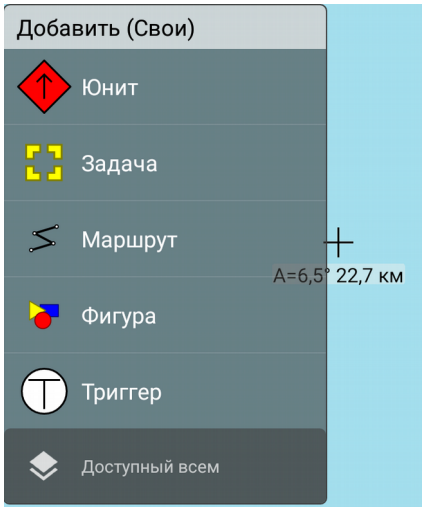
Для автоматизации игрового процесса (учений) с помощью системы ComBat предусмотрены триггеры и скриптовый язык.

Триггеры – это специальные объекты на карте, которые видны только пользователям с правами «Создатель игры/Администратор» и используются для выполнения цепочки команд (так называемого «**скрипта**») при наступлении определенного условия. Условиями могут быть расстояние или азимут между метками на карте, их статус, видимость, вхождение в область определенной фигуры и т.п. В качестве команд может быть смена статуса, видимости, положения меток, отправка сообщений и приказов пользователям и т.п.

На практике триггеры могут быть использованы для автоматизации таких процессов как: условные «минные» поля, предупреждение о выходе за игровую территорию или о подходе противника, автоматическое возрождение в определенной зоне с указанной задержкой по времени, отображение плана следующей миссии при выполнении целей предыдущей, анализ захвата территорий и подсчет времени или количества этих захватов, а также многое другое. Специально разработанный набор команд имеет большой потенциал и позволит организаторам игр эффективно использовать функционал системы.

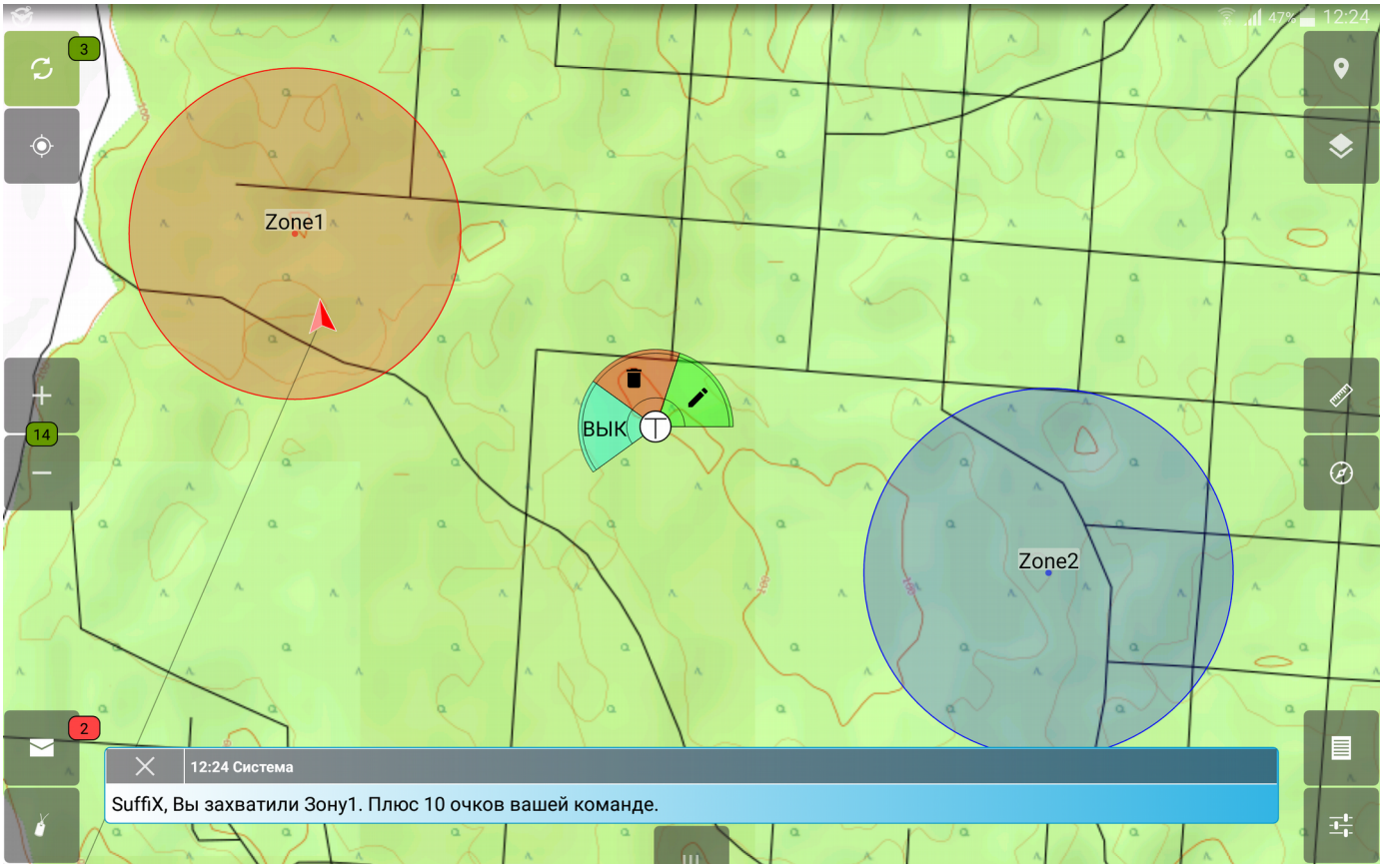
Добавление триггера

Создание триггеров доступно только пользователям с правами «Организатор игры»/«Администратор».



Чтобы добавить триггер необходимо зажать палец на выбранном пустом месте карты и в появившемся меню **«Добавить»** выбрать пункт **«Триггер»**.

В результате на карте появится метка триггера и его секторное меню.

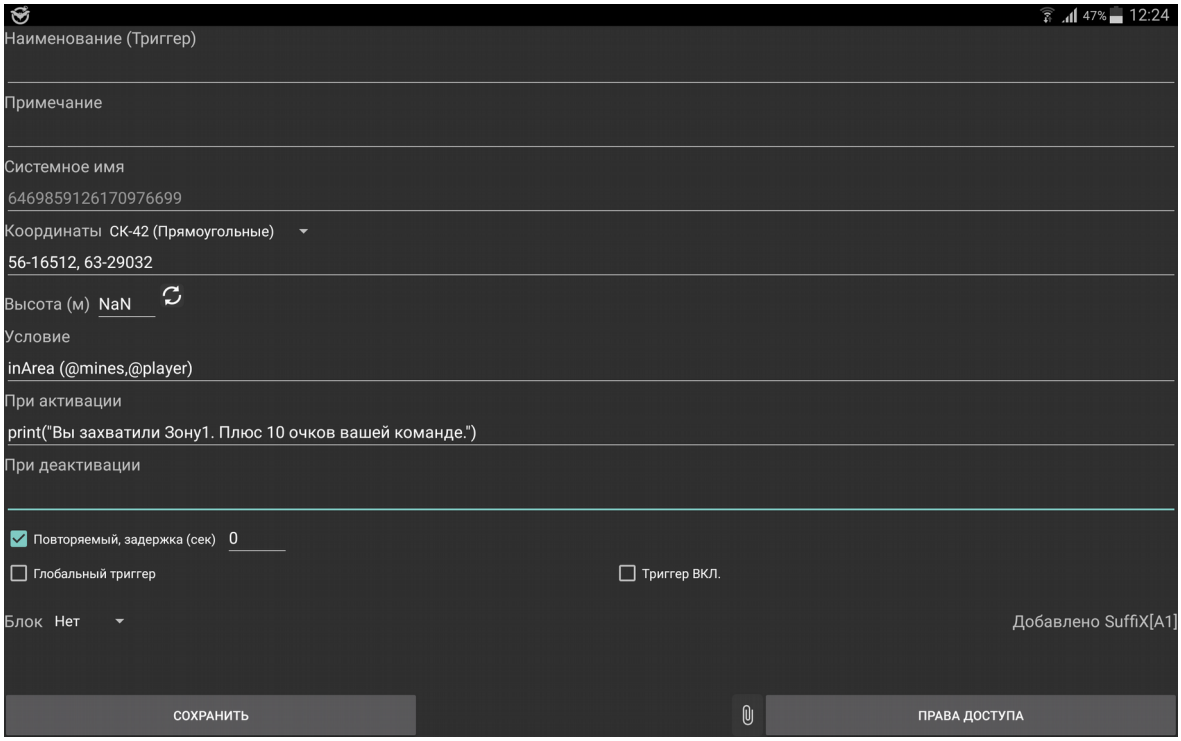


Нажмите зеленую опцию **«Редактировать»**, чтобы зайти в редактирование атрибутов триггера.

Кнопка **«ВКЛ»** / **«ВЫК»** предназначена для быстрого включения и выключения триггера в процессе мероприятия.

Для удаления триггера надмите красную кнопку **«Удалить»**.

У каждого триггера в атрибутах есть:



- **Наименование** - для визуального различия триггеров на карте;
- **Примечание** - для подробного описания назначения триггера;
- **Системное имя** - для обращения к атрибутам триггера из скриптов;
- **Условие** активации триггера, состоящее из конструкций, описанных в данном руководстве ниже;
- Скрипт **при активации**, который срабатывает, если вышеуказанное условие становится истинно;
- Скрипт **при деактивации**, который срабатывает, если вышеуказанное условие перестает быть истинно, то есть становится ложно.

Триггер может быть в одном из двух состояний - активирован или деактивирован. Активация наступает в момент, когда условие становится истинным. При этом происходит переключение состояния триггера и вызов скрипта по активации. Пока условие не станет ложно, скрипт активации больше не выполняется. Если условие становится ложно, то, соответственно, выполняется скрипт по деактивации.

Триггеры бывают **одноразовые** и **повторяемые**. У повторяемого триггера скрипт при активации и деактивации выполняется неограниченное число раз, а у одноразового – только один. Повторяемому триггеру можно задать паузу между срабатываниями. Режим **повторяемости и задержка** в секундах указывается в соответствующих атрибутах триггера.

Еще триггеры делятся на **локальные** и **глобальные**. Локальные триггеры срабатывают на каждом устройстве пользователя в отдельности, а глобальные срабатывают только на том устройстве, где условие сработало первым по времени, после чего факт срабатывания фиксируется на сервере и на остальных устройствах данный триггер игнорируется. Эта функция переключается опцией «**Глобальный триггер**» на экране свойств.

Так же, триггер может быть **включен** или **выключен**. У выключенного триггера не проверяется его условие активации и, соответственно, не срабатывает ни скрипт при активации, ни скрипт при деактивации. Статус включенности триггера определяется флажком «**Триггер ВКЛ.**», а также может быть изменен программно из скрипта (см. ниже).

Принцип построения скрипта

Для составления условий триггеров можно использовать операторы и функции, описанные в следующем разделе данного руководства.

В качестве скрипта можно использовать набор команд и функций, описанных в следующем разделе данного руководства. Команды можно писать, как в одну строчку, так и с переносом. Команды разделяются символом ";". Для последней команды скрипта или блока разделитель можно не ставить. Блоки команд выделяются фигурными скобками “{ }”. После закрытия блока команд разделитель не ставится.

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

В системе можно создавать хранилища переменных, которые позволяют сохранять дробные и целочисленные значения, произвольные строки, идентификаторы объектов или их множества в виде строк по ключу. С помощью переменных хранилища можно отмерять прошедшее время или считать количество активаций триггера, хранить промежуточные значения координат, списки меток и т.п. Все переменные в хранилище имеют строковый тип: числовые значения хранятся как строки; имена объектов хранятся как строки, начинающиеся на символ «@»; координаты хранятся как текст; множества хранятся как перечень наименований объектов через запятую (см. ниже). Если не вызывать команду update(<@хранилище>), то переменные в нем будут локальные для текущего устройства. Пример присвоения значния в переменную хранилища:

@хранилище~переменная = <значение>;

Пример операций со значением переменной:

if(@хранилище~переменная == 2) then {};

Каждый триггер по умолчанию имеет связанное с ним хранилище, где находятся системные переменные состояния: **active** — состояние триггера (TRUE — триггер активирован, FALSE — триггер деакиивирован), **activator** — числовой айди метки пользователя, активировавшего объект, **executed** — статус сработки одноразового триггера (TRUE — уже сработал, FALSE — еще не срботал) и **time** — время активации. Эти и другие дополнительные значения можно считать или записать подставив имя триггера или константу @self вместо имени хранилища. Ханилище глобаьного триггера автоматически помечается для отправки на сервер при установке значений.

В програмне доступны множества объектов, задаваемые следующим образом «@объект1,@объект2,...@объектN». Множества используются для выполнения серии одинаковых команд над несколькими объектами. Кроме изначального ручного задания множества, его можно получить с помощью функций **select**, **getUnits** или **getPoints**. Множеством можно управлять поэлементно с помощью команд **put** и **drop**. Команда **count** возвращает количество элементов во множестве.

Оператор **forEach**(<@множество>) {<блок>} позволяет выполнить определенный набор команд для каждого объекта в множестве. При этом переменная **@obj** содержит текущий элемент множества.

Если скрипт содержит ошибку, пользователь увидит в консоли сообщение об ошибке, в котором будет описано наименование скрипта, строка с ошибкой и другие детали. Консоль открывается из меню настроек программы пунктом «Консоль».

Если функция не может вернуть значение, то результатом будет литерал **NULL**.

Список скриптовых конструкций

Логические операторы

При написании условий можно использовать следующие логические операторы:

- **&&** или **AND** - логическое «И»;
- **||** или **OR** - логическое «ИЛИ»
- **«==»** - равно;
- **«!=»** - неравно;
- **«>=»** - больше или равно;
- **«<=»** - меньше или равно;
- **«>»** - больше;
- **«<»** - меньше;

В качестве логических значений используются числа по принципу: все, что больше 0, истинно, а остальное — ложно, либо литералы **TRUE** и **FALSE**, интерпретируемые системой как 1 и 0 соответсвенно.

Арифметические операции

Для вычисления различных формул можно использовать следующие арифметические операции:

- **«+»** - сложение
- **«-»** - вычитание
- **«*»** - умножение
- **«/»** - целочисленное деление
- **«%»** - получить остаток от деления

Важно! В случае если блоки операндов отделены скобками "( )", тогда выполнение действий начинается с самых внутренних скобок. По умолчанию действия производятся слева направо и приоритетность оераторов автоматически не учитывается!

Например: 2+4*2 даст в результате 12 - сначала выполнится сложение, затем умножение. Чтобы посчитать классически, нужно написать 2+(4*2). Если в качестве аргументов логических выражений используются вычисляемые значения, то лучше их брать в скобки, например, (1+3) OR (2-4).

Операторы условного перехода

Для ветвления хода выполнения скрипта можно использовать следующую конструкцию условного перехода:

If (<условие>) **then** {<блок1>} **else** {<блок2>};

Если условие истинно, тогда выполняется <блок1>, иначе выполняется <блок2>.

Можно использовать сокращенную версию:

If (<условие>) **then** {<блок1>};

Вместо блока можно писать одну команду без фигурных скобок:

If (<условие>) **then** <команда1>; **else** <команда2>;

Константы

В процессе написания скрипта вы будете оперировать некоторыми понятиями, которые определены фиксированными константами.

Глобальные ссылки на объекты:

- **@self** – триггер, в котором выполняется текущий скрипт.
- **@player** – метка пользователя, на устройстве которого отрабатывает скрипт.
- **@team** – метка команды текущего пользователя.
- **@leader** – метка лидера команды текущего пользователя.
- **@side** – идентификатор стороны текущего пользователя (для сообщений).
- **@all** – идентификатор широковещательной отправки сообщений
- **@obj** – указатель на текущий элемент внутри цикла **forEach.**
- **@sel** – указатель на текущий элемент внутри условия функции **select.**

Глобальные литералы:

- **TRUE** – (1) истина;
- **FALSE** – (0) ложь;
- **NULL** – значение не определено.

Тип адреса:

- **0** – конкретному пользователю (по умолчанию);
- **1** - лидеру группы;
- **2** - всем членам группы;
- **3** - всем членам группы и ее подгрупп по иерархии вниз;

Вид блокировки объекта:

- **NONE** – не блокирован;
- **OWNER** - блокирован пользователем;
- **ADMIN** - блокирован администратором.

Тип юнита:

- **EQUI** – (equipment) вооружение;
- **UNIT** – (unit) подразделение;
- **INST** – (installation) сооружение;
- **AVIA** – (aviation) авиация;

Отношение:

- **F** – (friend) союзник;
- **H** – (hostile) враг;
- **U** – (unknown) неизвестный;
- **N** – (neutral) нейтрал;
- **S** – (suspect) вероятно враг;
- **A** – (assuming friend) предположительно свой;
- **P** – (pending) не определен;

Размерность группы:

- **A** – (team) звено;
- **B** – (squad) отделение;
- **C** – (section) секция;
- **D** – (platoon) взвод;
- **E** – (company) рота;

- **F** – (battalion) батальон;
- **G** – (regiment) полк;

Мобильность вооружения:

- **O** – (whelled) колесная;
- **P** – (cross-country) повышеной проходимости;
- **Q** – (tracked) гусеничная;
- **R** – (wheel & track) колеса и гусеницы;
- **S** – (towed) на буксире;
- **T** – (railway) на ж/д платформе;
- **U** – (snow) на лыжах;
- **V** – (sled) на санях
- **W** – (aimals) на упряжке;
- **X** – (barge) на барже;
- **Y** – (amphibious) амфибия;
- **Z** – (unspecified) не задано;

Звания пользователей:

- **0** – (Private) рядовой;
- **1** - (Corporal) капрал;
- **2** – (Sergeant) сержант;
- **3** – (Lieutenant) лейтенант;
- **4** – (Captain) капитан;
- **5** – (Major) майор;
- **6** – (Colonel) полковник.

Тип действия юнита:

- **LOOK** – наблюдает;
- **GO** – движется;
- **NONE** – стоит**.**

Тип целеуказания:

- **BACK** (fall back) – в строй;
- **MOVE** (move) – двигайтесь;
- **ATTACK** (attack) – атакуйте;
- **CAPTURE** (capture) – займите;
- **SPY** (find) – разведайте;
- **NONE** (stop) – отбой.

Статусы целей:

- **ACT** – (actual) актуально;
- **CMP** – (completed) выполнено;
- **FAIL** – (failed) провалено.

Статусы пользователя:

- **ACTIVE** – на связи;
- **CLEAR** – чисто;
- **DANGER** – в опасности;
- **AMMO** – мало боеприпасов;
- **FUEL** – мало топлива;
- **REPAIR** – нужен ремонт;
- **INJURED** – ранен;
- **DEAD** – выбыл.

Статусы группы:

- **CLEAR** – чисто;
- **DANGER** – в опасности;
- **AMMO** – мало боеприпасов;
- **FUEL** – мало топлива;
- **REPAIR** – нужен ремонт;
- **EQUIPMENT** – техника потеряна;
- **CRITICAL** – критические потери.

Типы фигуры:

- **RECT** – (rectangle) прямоугольник;
- **OVAL** – (ellipse) эллипс.

Типы многоточечного объекта:

- **PL** – (polyline) линия;
- **PG** – (polygon) многоугольник.

Типы маршрутных точек:

- **0 –** точка маршрута
- **1** – (Coordination) точка координации;
- **3** – (Named area) точка интереса.
- **4** – (Start point) точка старта.
- **5** – (Casualities exchange) точка обмена ранеными;
- **6** – (Causalities collection) точка сбора раненых;
- **7** – (Attack) точка начала атаки;
- **8** – (Cover) точка огневой поддержки.
- **9** – (Ambush) точка организации засады.

Функции

Пользователю доступны следующие функции…

Логические и арифметические:

not(<число>) - логическое «НЕ». Если <значение> <= 0 или FALSE, то возвращает TURE, в противном случае FALSE.

rnd(<число>) – генерирует случайное число в диапазоне от 0 до <число>. Параметр и результат дробные.

abs (<число>) – получает значение числа по модулю.

round(<число>) – округляет <число> до целого.

Географические:

inArea(<@фигура>, <@объект>) – проверяет или объект находится внутри фигуры. Возвращает TRUE или FALSE.

getDist(<@объект1>,<@объект2>) – возвращает дистанцию в метрах между двумя объектами.

getAngle(<@объект1>,<@объект2>) – возвращает азимут (число от 0 до 359) от <объект1> на <объект2>.

getOffset(<координаты>,<дистанция>,<азимут>) – возвращает координаты со смещением по указанному азимуту на указанное расстояние от указаных координат.

Получение свойств объектов:

getClass(<@объект>) – возвращает тип любого объекта на карте (Например: Unit, Task, Shape и т.п.).

getId(<@объект>) - возвращает внутреннее числовое ID объекта.

getSide(<@объект>) – возвращает цифровой идентификатор игровой стороны объекта. Идентификаторы указаны в настройках программы на вкладке «Игра».

getOwner (<@объект>) - возвращает идентификатор создателя объекта.

getModifier (<@объект>) - возвращает идентификатор пользователя изменившего объект.

getTime (<@объект>) - возвращает время последнего изменения объекта в секундах.

getName(<@объект>) – возвращает системное имя объекта (имя переменной).

getCapt(<@объект>) - возвращает наименование объекта на карте (подпись метки).

getInfo(<@объект>) - возвращает детальное описание объекта.

getLock(<@объект>) – возвращает статус блокировки объекта (см. константы).

getLayer(<@объект>) – возвращает идентификатор слоя на котором лежит объект.

getPos(<@объект>) – возвращает строку с координатами объекта во внутреннем формате. Для читабельного вида используйте posStr(getPos(<@объект>)).

getAlt(<@объект>) – возвращает высоту объекта над уровнем моря в метрах.

getDir(<@объект>) – возвращает азимут направления обзора или движения юнита, а так же угол поворота фигуры или метки в градусах от 0 до 359.

getAct (<@юнит>) – возвращает тип активности юнита (см. константы).

getType(<@объект>) – возвращает тип юнита, тип фигры, тип точки или тип маршрутной точки (см. константы).

getAffiliation(<@юнит>) – возвращает идентификатор принадлежности боевой единицы (см. константы).

getIdentity (<@юнит>) – возвращает специализацию юнита (тип иконки, числовле значение).

getUnitSize(<@юнит>) – возвращает размерность юнита (см. константы).

getMobility(<@юнит>) – возвращает мобильность юнита (см. константы).

getRank(<@юнит>) – возвращает звание пользователя (см. константы).

getTeam(<@юнит>) - возвращает идентификатор группы, в которую входит пользователь или группа.

getLeader(<@юнит>) - возвращает идентификатор лидера указанного пользователя или группы.

getTeamLeader(<@команда>) - возвращает идентификатор лидера внутри группы.

getUnits(<@команда>) – возвращает множество юнитов, входящих в группу.

getTarget(<@юнит>) - возвращает идентификатор объекта навигации указанного пользователя или группы.

getTargetAction(<@юнит>) - возвращает тип целеуказания (см. константы).

getColor(<@фигура>) – возвращает цвет фигуры или маршрута (HEX строка вида #AARRGGBB).

getWidth(<@фигура>) - возвращает ширину фигуры в метрах.

getHeigth(<@фигура>) - возвращает высоту фигуры в метрах.

getPoints(<@линия>) – возвращает множество точек, входящих в линию или маршрут.

getStatus(<@объект>) – возвращает статус цели.

checkStatus(<@юнит>) – проверяет, или у юнита установлен указаный статус. Возвращает TRUE или FALSE.

isExist (<@объект>) – проверяет, существует ли указанный объект. Возвращает TRUE или FALSE.

isTeam(<@юнит>) – проверяет, является ли юнит группой. Возвращает TRUE или FALSE.

isUser(<@юнит>) – проверяет, является ли юнит пользователем. Возвращает TRUE или FALSE.

isPlayer(<@юнит>) – проверяет, является ли юнит текущим пользователем. Возвращает TRUE или FALSE.

isUpdated(<@объект>) – проверяет был ли объект помечен для отправки на сервер. Возвращает TRUE или FALSE.

isDeleted(<@объект>) – проверяет был ли объект помечен на удаление. Возвращает TRUE или FALSE.

isShared(<@объект>) - проверяет видимость объекта всем сторонам. Возвращает TRUE или FALSE.

isVisible(<@объект>) – проверяет видимость слоя или объекта на карте. Возвращает TRUE или FALSE.

isEnabled(<@триггер>**) –** получает статус триггера: TRUE - включен, FALSE - выключен.

isActivated(<@триггер>**) –** получает текущее состояние триггера: TRUE - активирован, FALSE - деактивирован.

Дополнительно:

time(<hh:mm:ss>) - возвращает время в секундах. Без параметра возвращает значение текущего системного времени и даты в секундах.

timeStr(<время в сек>**)** – выводит время в текстовом виде «hh:mm:ss».

dateStr(<время в сек>**) –** выводит дату в текстовом виде «dd.mm.yy».

posStr(<координаты>**) –** выводит координаты в читаемом формате из настроек.

select(“<условие>”) - производит выборку объектов согласно условию, где в качестве объекта используется идентификатор @sel. Например, select(“getDist(@sel, @player) < 100”) — выбирает все объекты ближе 100 м от текущего пользователя. **Внимание!** Кавычки вокруг условия обязательны.

count(”<множество>”) - возвращает количество объектов в множестве.

put(”<множество>”,<@объект>) - добавляет объект в множество.

drop(”<множество>”,<@объект>) - удаляет объект из множества.

Команды

Пользователю доступны следующие команды…

Сообщения:

log(”<текст>”) **–** выводит текст в консоль доступную администратору. Если необходимо вывести значение выражений, то выражение берется в «[» ,«]». Например Log(“Ваши очки: [val(@self.score)]”)

eval(<выражение>) **–** считает <выражение> и выдает результат в консоль. Аналог Log(”[выражение]”)

print (”<текст>”) **–** выводит сообщение на экран текущего пользователя от имени системы. Так же доступна подстановка вычисляемых выражений в «[» ,«]».

say (<@отправитель>, ”<массивполучателей>”,”<текст_сообщения>”) – отправляет сообщение конкретному пользователю или множеству пользователей от имени юнита отправителя. В качестве получателя могут быть так же константы @all, @side, @team, @leader, которые позволяют отправить сообщение всем игрокам в сценарии, игрокам стороны отправителя, команде отправителя или лидеру отправителя, соответственно.

Объекты:

update(<@объект>) – устанавливает атрибут изменения объекта, чтобы объект обновился на сервере. **Важно!** Не забывайте вызывать эту команду в конце серии команд по изменению объекта, иначе изменения останутся только на вашем устройстве.

delete(<@объект>) – удаляет метку или слой с карты.

share(<@объект>) - устанавливает видимость объекта всем сторонам.

private(<@объект>) - снимает видимость объекта всем сторонам.

setOwner (<@объект>,<@пользователь>) - меняет владельца объекта. Для последующей блокировки изменений от имени нового владельца.

addAddress(<@объект>,<адрес>,<модификатор>) – добавить получателя метки.

remAddress (<@объект>,<адрес>,<модификатор>) – убрать получателя метки.

clrAddress (<@объект>) – убрать получателей (снять фильтр).

show(<@объект>) – делает объект или слой видимым на карте.

hide(<@объект>) – делает объект или слой скрытым с карты.

setName (<@объект>,<@имя>) – задает системное имя объекта (имя переменной).

setCapt (<@объект>,<@наименование>) – задает наименование объекта (попись).

setInfo (<@объект>,<@примечение>) – задает текст примечания к объекту.

setLock(<@объект>,<статус>) – меняет статус блокировки объекта (см. константы).

setLayer(<@объект>,<слой>) – перемещает объект на другой слой.

setPos(<@объект>,<@координаты>) – перемещает объект на карте по указанным координатам (подставлять технический не отформатированый вид координат).

setAlt(<@объект>,<высота>) – устанавливает высоту объекта над уровнем моря в метрах.

setDir(<@объект>,<азимут>) – устанавливает азимут проворота иконки объекта в градусах.

setAct (<@юнит>,<@действие>,<@азимут>) – устанавливает тип активности юнита (см. кнстанты).

setType(<@объект>,<тип>) – устанавливает тип юнита, фигры или точки (см. кнстанты).

setAffiliation(<@объект>,<принадлежность>) – устанавливает идентификатор принадлежности боевой единицы (см. кнстанты).

setIdentity(<@юнит>,<специализация>) – устанавливает специализацию юнита (вид иконки, числовое значение).

setUnitSize(<@юнит>,<размер>) – изменяет размерность группы (см. кнстанты).

setMobility(<@юнит>,<мобильность>) – изменяет мобильность вооружения (см. кнстанты).

setRank(<@юнит>,<звание>) – устанавливает звание пользователя (см. кнстанты).

setOrder(<@объект_цель>,<действие>,<@получатель>,<@отправитель>) – отдает команду текущему пользователю на действие над указанным объектом. Действия описаны в константах. На экране появляется сообщение от отправителя. Аналог выдачи приказа.

addStatus(<@юнит>,<статус>) – добавляет юниту указаный статус.

remStatus(<@юнит>,<статус>) – удаляет указаный статус у юнита.

drop(<@юнит>) - отсоединяет юнит от текущей группы.

addUnit(@группа,@юнит) – добавляет в группу юнит.

remUnit(@группа,@юнит) – удаляет из группы юнит.

setLeader(<@группа>,<@лидер>) – устанавливает пользователя лидером группы.

setColor(<@фигура>,<цвет>) – изменяет цвет фигуры или маршрута на указанный (HEX строка вида #AARRGGBB или название — red, blue и т.п.).

setWidth(<@фигура>,<значение>) - устанавливает ширину фигуры в метрах.

setHeight(<@фигура>,<значение>) - устанавливает высоту фигуры в метрах.

addPoint(@линия,@точка,@предыдущая_точка) – добавляет в линию точку после предыдущей.

remPoint(@линия,@точка) – удаляет из линии точку.

setStatus(<@объект>,<статус>) - изменяет статус цели на указанный.

enable(<@триггер>) - включает триггер.

disable(<@триггер>) - выключает триггер.

activate(<@триггер>) - вызывает код триггера по активации игнорируя условие.

deactivate(<@триггер>) - вызывает код триггера по деактивации игнорируя условие.

reset(<@триггер>) - возвращает триггер в начальное состояние (не активирован, не сработал).

Системные:

clearVars(<@имя_хранилища>) - очищает список переменных указанного хранилища.

clearMsg() - очищает сообщения в чате у пользователя.

sound(<id>) - выдает звук из библиотеки доступных.

delay(<секунд>) - создает задержку перед выполнением следующей команды.

forceUpdate() - вызывает синхронизацию с сервером, чтобы передать сделанные изменения (видимость, статусы, сообщения).

Примеры

Простой пример работы логики. Если значение очков стороны 1 больше 100, то отправляем каждому пользователю стороны 1 персональное сообщение “Ура, победа, вы набрали Н очков.”

Условие триггера: *@self~side1_score >= 100*

По активации:

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

say(@player, @obj, "Ура, победа, вы набрали [@self~side1_score] очков.");

Для проверки работы триггера, задите в консоль и пропишите следующую команду:

@self~side1_score = 200;

Следующий более комплексный пример - **автореспаун**:

1) Создаем эллипс или прмоугольник респауна с системным именем *resp*.

2) Создаем локальный триггер:

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

2.2) При активации: *print("Вы зашли в зону респауна. Ожидайте возрождения 30 мин..."); delay(1800); remStatus(@player, DEAD); print("Вы ожили. Можете вступать в игру.")*

Еще один вариант - **минное поле** (зона аномалий) с вероятностью срабатывания 50%:

1) Создаем круг, кадрат или полигон минного поля с системным именем *mine*.

2) Создаем локальный триггер:

2.1) Условие: *inArea(@mine,@player)*

2.2) При активации: if(rnd(1) > 0.5) { print("Вы подорвались на мине! Выдвигайтесь на мертвяк."); addStatus(@player, DEAD) }

Авиаудар в выбранном в процессе игры месте:

1) Создаем локальный выключенный триггер:

1.1) Условие: inArea(@art_strike,@player)

1.2) При активации: print("Вас накрыл авиа удар! Выдвигайтесь на мертвяк."); addStatus(@player, DEAD)

2) Когда нужно совершить авиа удар, создаем в нужном месте круг с именем art_strike и включаем триггер. Все кто попадут в круг - погибают.

В такой реализации авиаудар может включить только админ, включив триггер, но можно подвязать активацию триггера на какоето условие, например, сменить статуса цели с именем "Запустить ракету" на "Выполнено" или проверку игрока на вхождение в зону пункта управления ПВО.

Аналогичным способом делаються аномалии для игр сталкера, предупреждение о выходе за игровую территорию и т.п.

Все вышеописанное - это не шаблон для копирования, а скорее демонстрационный пример. По хорошему нужно все нюансы продумывать и обходить возможное не честное поведение игрока.

Автоматическое переключение миссий с оповещением всех игроков об этом от имени Штаба:

1) Называем пользователя который бует выступать в роли сервера именем "Штаб".

2) Создаем задачу с именем mission1. Наносим на него все необходимые объекты, маршруты и т.п.

3) Создаем задачу с именем mission2. Наносим на него все необходимые объекты, маршруты и т.п. Скрываем ее опцией "Временно спрятать слой от всех".

4) Создаем глобальный триггер:

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

4.2) При активации: say(@player, select(“isUser(@sel)”),"Поздравляем, вы успешно выполнили первую миссию! Получена новая задача. См. карту..."); hide(@mission1); show(@mission2)

В результе если кто либо из игроков установит у цели mission1 статус "Выполнено", то произойдет скрытие объектов первой миссии, отображение объектов второй миссии, а так же все игроки на карте получать сообщение от группы Штаб о успешном выполнении.

Пример примитивной реализации **войны за территории**.

1) Меняем имя админского юзера на "Штаб" для красоты отправки сообщений от имени Штаба.

2) Создаем на карте определенное количество зон в виде кругов, квадратов или многоуголтников (с именем zone1,zone2...,zoneN). Делаем их по умолчанию зелеными. Зеленые зоны будут означать не занятые, жельтые - идет бой, красные - захвачены стороной красных, синие - захвачены стороной синих.

3) Создаем глобальный одноразовый выключенный триггер с подписью «Reset» и системным именем *var* для хранения и инициализации переменных счета. При включении триггера админом будет происходить сброс очков, цвета зон и выключение триггера.

3.1) Условие: *TRUE* (то есть всегда безусловно выполняется)

3.2) По активаци (коментари из кода перед вставкой в триггер надо убрать):

```
@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) Создаем локальный многоразовый триггер «Core» со временем повторения 60 сек (раз в сколько секунд добавлять бал за контроль) для анализа состоянй захвата территорий.

4.1) Условие: *getName(@player) == Server* (работает только на устройстве с именем пользователя Server)

4.2) По активации:

```
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);
```

Суть механики - каждые 60 секунд все зоны на карте проверяються на наличие в них не мертвых игроков красной и синей стороны. Если в зоне пристусвуют обе стороны, то она стает желтая и очки никому не начисляються. Если в ней присутсвуют игроки только одной стороны, то зона принимает цвет стороны и начисляет по 1 балу за каждую минуту контроля. Если зона осталась желтая и в ней никого нет, она стает зеленая.

5) Создаем выключеный локальный триггер «Score» для показа счета:

5.1) Условие: *TRUE*

5.2) По активации:

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

При включении триггера он отправит всем участникам обеих сторон сообщение со счетом.