

РУКОВОДСТВО К

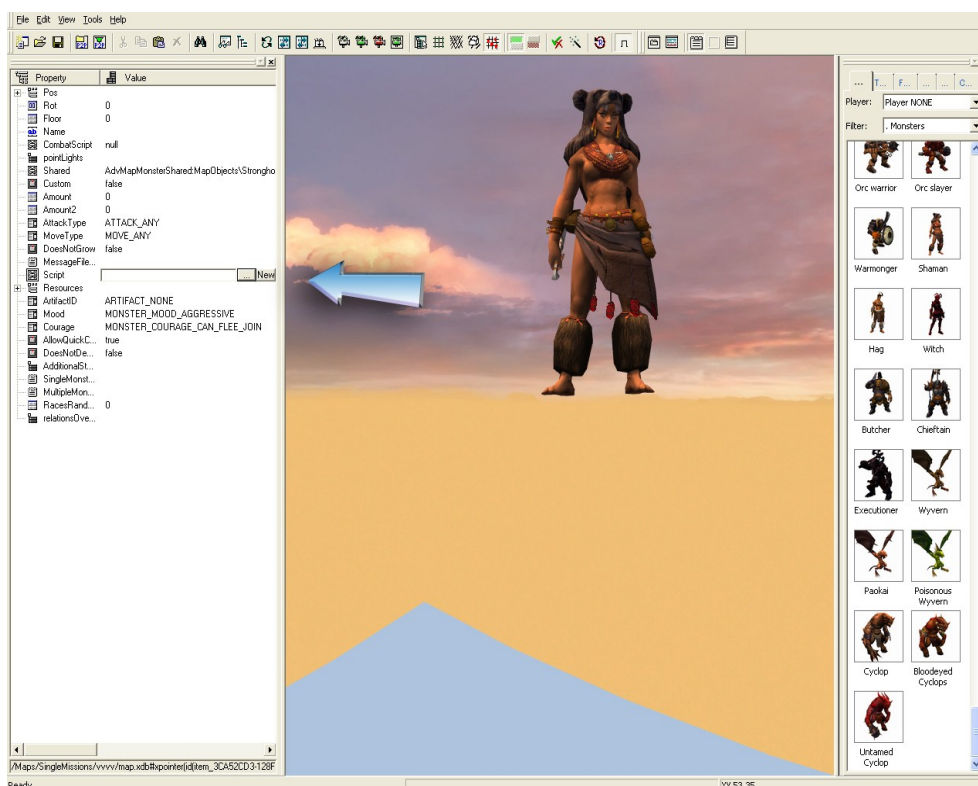
HoMM3 events library

автор OGO

Это руководство является чем-то вроде практического пособия по использованию библиотеки libhomm3.lua. То есть, если зная теорию, которая изложена в теоретическом руководстве, вы не можете применить её на практике — оно для вас. Так что будьте готовы постоянно заглядывать в теорию, ибо здесь описано лишь её применение на практике. Если вы совершенно не знакомы со скриптами и не знаете с чем их едят — обязательно прочтите [это](#) руководство (называется “[От чайника к чанику](#)”).

Что такое библиотека? Как я понимаю — набор команд. С помощью этой хитрой штуки у нас появился целый комплект возможностей, схожих с теми, что были в третьих героях — то есть, без лишних хлопот поставить на карте какие-нибудь скриптовые события. Правда, хлопоты всё-таки будут, однако эта система создания событий гораздо проще обычных скриптовых команд. Итак, начнём...

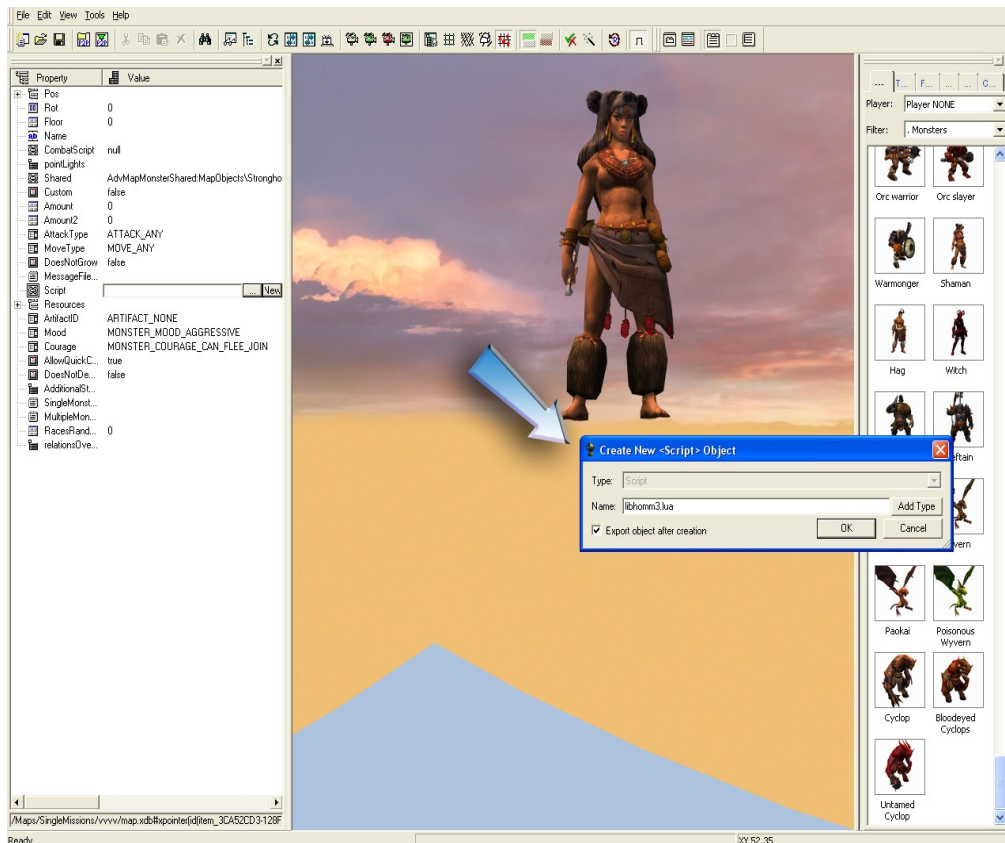
Самым тяжёлым для меня в библиотеке оказалось её подключение. Это действие порой даётся не просто. Поэтому я тщательно за скриншотил это событие:



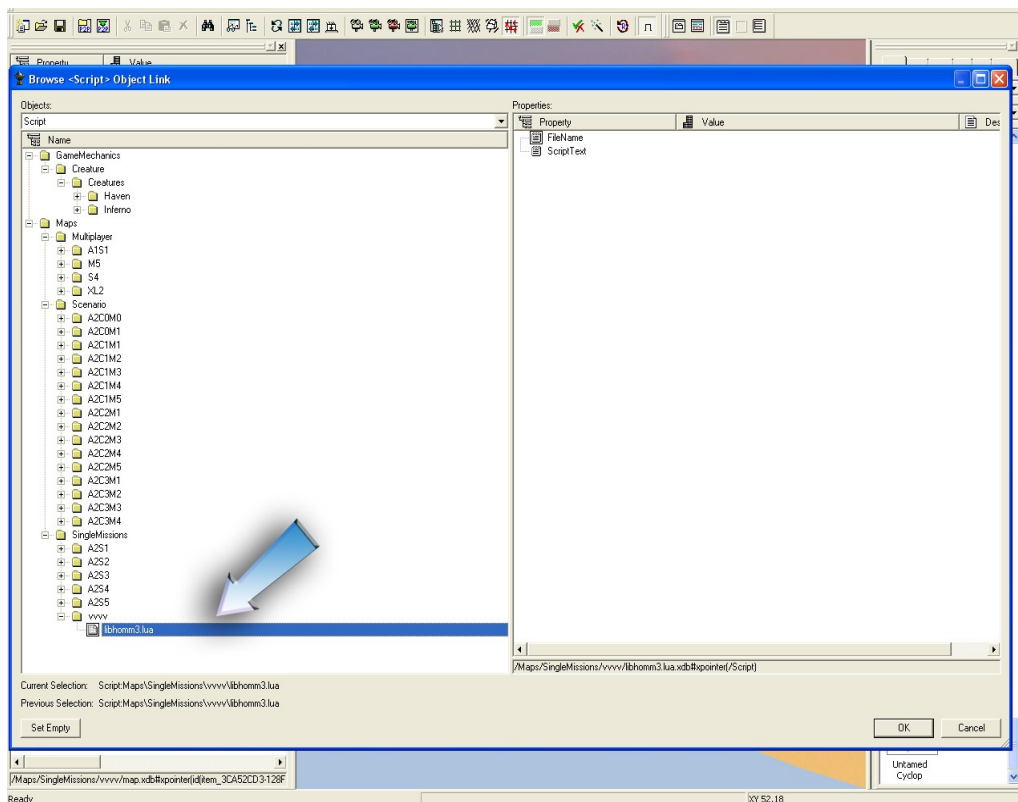
Для подключения библиотеки нужно для начала поместить её в архив карты. Это делается просто и скриншотов для объяснения не требует. Открываете карту любым архиватором, заходите в конечную папку и перетаскиваете туда libhomm3.lua...

А вот теперь можно смотреть картинки — для начала выставляем на карту любого монстра (если его там ещё не было, конечно). В нашем случае — это шаманка. Находим на панели

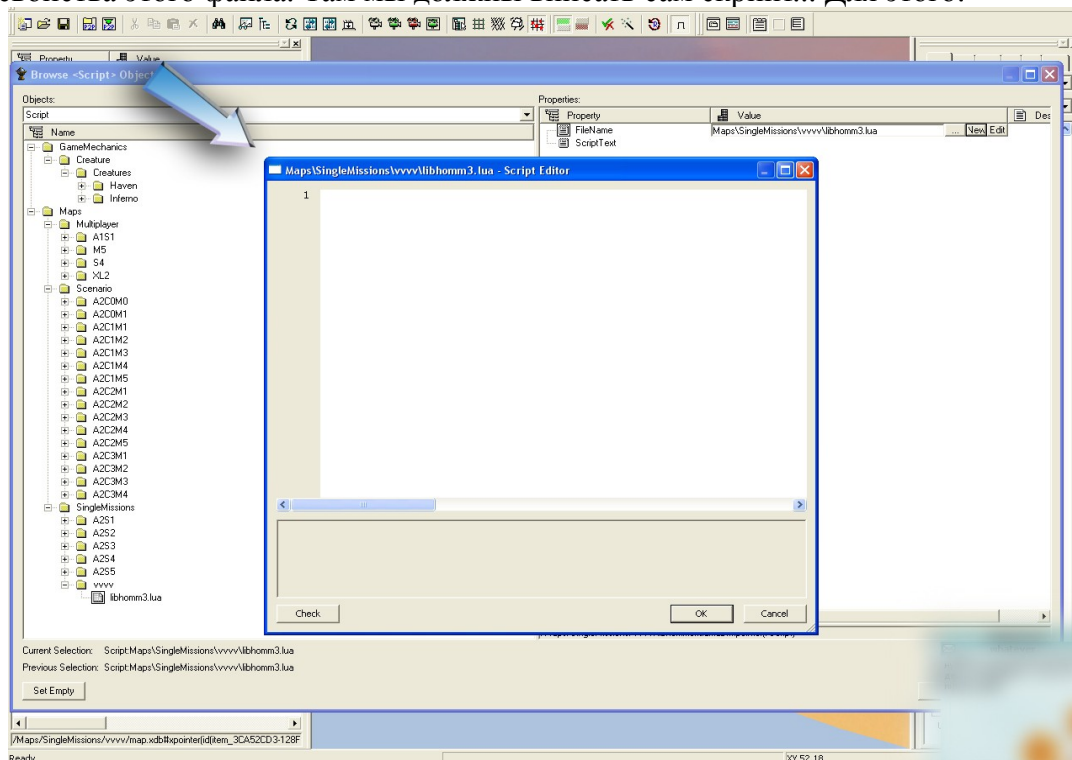
свойств строчку Scripts... Нажимаем на New и появляется окошечко...



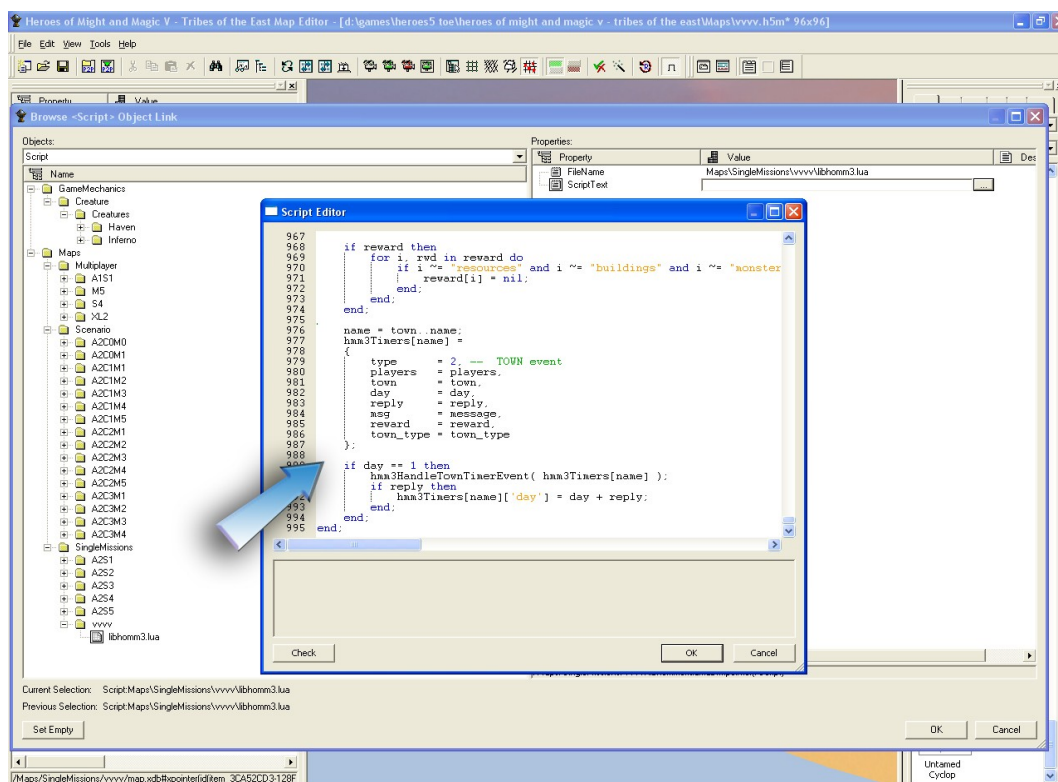
В этом окошечке мы вписываем имя библиотеки — libhomm3.lua... Тогда появляется большое окно:



В этом окне, слева, мы находим libhomm3.lua в папке с названием вашей карты. Справа мы видим свойства этого файла. Там мы должны вписать сам скрипт... Для этого:



Выбираем свойство ScriptFile, нажимаем кнопку New, снова вводим libhomm3.lua и нажимаем ОК... Теперь само открывается новое окно — редактор скриптов. Туда надо поместить весь текст библиотеки. Для этого открываем libhomm3.lua любым текстовым редактором, выделяем весь текст (Ctrl+A), копируем его (Ctrl+C), и вставляем в этом окне (Ctrl+V).



Нажимаете ОК. Библиотека ваша.

Вся эта непростая простая процедура стоит того, чтобы потом не мучаться с выдумыванием простых скриптовых функций.

ИСПОЛЬЗОВАНИЕ

Все возможности и значения библиотеки вы с лёгкостью найдёте в теоретическом руководстве. Мы же рассмотрим всё на примере. Для примера же я создал карту под название “Битва за существование” (прилагается в комплекте), которая сможет продемонстрировать возможности скриптовой библиотеки `libhomm3.lua`...

Для того чтобы библиотека заработала, нам нужно вписать следующую строчку на начале `doFile( GetMapDataPath().. "libhomm3.lua" );` и `startThread( initMap );` в конце.

```
3 function initMap()
4
5     homm3CreateTownEvent( {PLAYER_1}, "building", "gnomtown", 2, nil, "gnomtown.txt",
6     {
7         buildings =
8         {
9             { type = TOWN_BUILDING_DWELLING_1, level = 1 },
10            { type = TOWN_BUILDING_DWELLING_2, level = 1 },
11            { type = TOWN_BUILDING_DWELLING_3, level = 1 }
12        },
13        monsters = { 28, 16, 9, 0, 0, 0, 0 }
14    },
15    TOWN_FORTRESS
16    );
17
18    homm3CreateMapEvent( {PLAYER_1}, "king", 7, 7, "gold.txt",
19    {
20        resources = { gold = random (3000)+1500, wood = random (3)+2, ore = random (3)+2, mercury = random (3)+1, crystal = :
21    },
22    1
23    );
24
25    homm3CreateRegionEvent({PLAYER_1}, "zasada", nil, "zasada.txt", nil,
26    {
27        {monster = CREATURE_STALKER, count = 21},
28        {monster = CREATURE_ASSASSIN, count = 21},
29        {monster = CREATURE_ASSASSIN, count = 21},
30        {monster = CREATURE_ASSASSIN, count = 21},
31        {monster = CREATURE_ASSASSIN, count = 21},
32        {monster = CREATURE_STALKER, count = 21}
33    },
34    1
35    );
36
37    homm3CreateTreasureEvent( "major", nil, "major.txt",
38    {
39        {monster = CREATURE_ANGER_TREANT, count = 6},
40        {monster = CREATURE_ANGER_TREANT, count = 7}
41    },
42    );
43
44    homm3CreateMonsterEvent( "dancers", "Diraya", "dancers.txt",
45    {
46        creatures = {{monster = CREATURE_BLADE_SINGER, count = 30}}
47    },
48    );
49
50    homm3CreateTownEvent( {PLAYER_2}, "morecreatures1", "varvar1", 2, 15, nil,
51    {
52        creatures = { 36, 24, 16, 8, 6, 3, 1 }
53    },
54    TOWN_STRONGHOLD
55    );
56
57    homm3CreateTownEvent( {PLAYER_3}, "morecreatures2", "varvar2", 2, 15, nil,
58    {
59        creatures = { 36, 24, 16, 8, 6, 3, 1 }
60    },
61    TOWN_STRONGHOLD
62    );
63
64    end;
```

Все команды следует записывать в функцию `initMap ()`, как это показано на скриншоте.

Итак, разберём каждую команду — как я её записал, что она делает и зачем.

Первой идёт *hmm3CreateTownEvent(players, name, town, day, reply, message, reward, town_type);*

Событие реагирует на наступление определённого игрового дня и служит для работы с городскими строениями, выдачей и забираем у игрока ресурсов, пр.

Первым аргументом (*players*) выступает идентификатор игрока. Моя карта, “Борьба за существование” создана для одиночной игры. Поэтому все основные события разворачиваются вокруг первого игрока. Вот и событие на город происходит исключительно для него. В скриптах записано `PLAYER_1`. Обратите внимание на скобки, в которых записано значение. Они будут повсюду.

Вторым аргументом (*name*) я передаю имя команды. Вы можете записать в скрипте хоть сотню разных *hmm3CreateTownEvent*, поэтому каждый из них должен иметь уникальное имя (“**building**” у меня).

Третьим аргументом (*town*) выступает скриптовое имя города. Его мы задаём в свойствах данного города в редакторе карт. В данном случае это имя “**gnomtown**”.

Четвёртый аргумент (*day*) передаёт день, на который должно произойти событие. В данном случае — это второй день.

Если вы хотите, чтобы это событие повторялось с определённым промежутком времени, впишите в пятом аргументе (*reply*) значение этого временного промежутка в днях. Если хотите повтора каждую неделю — впишите число семь. Если повтор вам, как и мне в данном случае, ни к чему, впишите значение `nil`.

Шестым аргументом (*message*) выступает название файла с текстом, который должен увидеть игрок при наступлении события. Естественно, что это текстовый файл должен быть в `Unicod'e` и находиться в архиве с картой (подробности можно посмотреть в руководстве “От чайника к чайникам”).

Седьмым аргументом (*reward*) идёт награда. Визуально это достаточно страшно выглядит, но на самом деле, ничего страшного тут нет. Открываем мегаскобки и вписываем любые из предложенных видов наград: **resources, guard, buildings**.

После того, как вы выбираете одну из наград, записывайте её следующим образом:

```
buildings = {  
    { type = TOWN_BUILDING_DWELLING_1, level = 1 },  
    { type = TOWN_BUILDING_DWELLING_2, level = 1 },  
    { type = TOWN_BUILDING_DWELLING_3, level = 1 }  
}.
```

— это своеобразная матрица скобок. В скобках награда в

целом, как таковая, и каждый её компонент тоже в своих собственных скобках.

Наград может быть несколько, записываются они через запятую. У меня второй наградой выступает добавление юнитов, доступных к рекрутированию в городе. Записывается похожим образом:

`monsters = { 28, 16, 9, 0, 0, 0, 0 }`, где первое значение отвечает первому уровню существ и седьмое, соответственно, седьмому.

Последним аргументом (*town_type*); выступает идентификатор города. В примере — это город гномов, `TOWN_FORTRESS`

Далее в моём скрипте идёт *hmm3CreateMapEvent(players, name, day, reply, message, reward);*

Событие реагирует на наступление определённого дня и по желанию выдаёт сообщение и награду.

Первым аргументом (*players*) уже привычно выступает идентификатор игрока. Далее (*name*) уникальное имя, которое вы сами даёт команде.

Третий аргумент (*day*) это день, на который срабатывает команда. У меня это седьмой день.

Четвёртый аргумент (*reply*) даёт нам возможность повторять это событие. Оно будет повторяться с выставленным вами интервалом. Я поставил число 7 — соответственно

событие повторяется каждые семь дней. Если вы хотите, чтобы событие было однократным, то просто поставьте его в значение `nil`.

Далее, пятым аргументом (*message*), идёт сообщение, которое будет появляться при наступлении события.

Заканчивает всё это дело награда (*reward*). Она идёт знакомой матрёшкой скобок (массивы в массивах). Правда на этот раз доступен только один вид награды - **resources**. У меня это выглядит так:

```
{
    resources = { gold = random (3000)+1500, wood = random (3)+2, ore = random
(3)+2, mercury = random (3)+1, crystal = random (2), sulfur = random (1)+1, gem = random (2)+1 }
},
```

Тут всё просто... Вы задаёте, сколько золота, древесины или прочих ресурсов, получит игрок. Я к тому же использую значение рэндома, т.е. награда = (золото = случайное количество с 3000 + 1500, ... - именно столько золота получит первый игрок.

Далее я использую *hmm3CreateRegionEvent(players, region, hero, pre_message, post_message, guard, reward, once)*;

С этой командой достаточно зайти в обозначенный регион, как тут же что-то случится. Что? Зависит от вас:)

Первым аргументом (*players*), как всегда, выступает идентификатор игрока. Вторым аргументом (*region*) я указываю имя созданного региона. Регион можно создавать в редакторе карт, там же нужно задать ему имя (подробнее в руководстве “От чайника к чайнику”). Таким образом, при посещении данного региона будет срабатывать это событие.

Третий аргумент (*hero*) несёт в себе имя героя. Только тот герой, чьё имя там указано, сможет активировать событие. Если вы хотите оставить эту возможность для всех героев, поставьте значение в `nil` (то есть так же, как это сделано у меня).

Далее (*pre_message, post_message*) идёт два текста. Один появится перед событием, вызванным посещением этого региона, второй сразу после этого события (к примеру сообщение о начале битвы, битва, сообщение о награде за неё). Если вам не нужны сообщения. то поставьте значение `nil` на месте того сообщения, которое вам не нужно (т.е. вместо *pre_message* или *post_message*).

Шестой аргумент (*guard*) это матрёшка охраны (одни большие скобки, которые вмещают в себя много маленьких). Выглядит она так:

```
{
    {monster = CREATURE_STALKER, count = 21},
    {monster = CREATURE_ASSASSIN, count = 21},
    {monster = CREATURE_ASSASSIN, count = 21},
    {monster = CREATURE_ASSASSIN, count = 21},
    {monster = CREATURE_ASSASSIN, count = 21},
    {monster = CREATURE_STALKER, = 21}
}, - всё просто. Сколько значений monster вы напишите, столько
```

стеков и будет. Указываете название монстра и количество (count).

Далее (*reward*) идёт награда. Записывается точно так же, как и в том случае, который мы рассматривали выше, с той лишь разницей, что видов наград здесь море (смотрите руководство Ungerade).

В конце (*once*) осталось выставить, сколько раз вы хотите это событие повторить... Один? Два? Сколько угодно — просто поставьте значение.

hmm3CreateTreasureEvent (treasure, hero, pre_message, guard);

Первый аргумент (*treasure*), это имя объекта. У меня это артефакт “Кольцо машиниста”. В редакторе я назвал его **”major”**. Объектом может быть также ресурс.

Для какого героя должно сработать событие, передаёт второй аргумент (*hero*).

На моей карте, если вы захотите взять “Кольцо машиниста” - последует вопрос. Текст этого вопроса я задал в третьем аргументе (*pre_message*).

И последним аргументом (*guard*) идёт, конечно же, сама охрана, битва с которой начнётся, как только вы дадите утвердительный ответ на вопрос. Записывается тем же способом, что и раньше.

hmm3CreateMonsterEvent(monster, hero, pre_message, reward);

Первым аргументом (*monster*) выступает монстр, на которого “навешивается” событие. Аналогично предыдущей команде, имя монстра задаётся в редакторе. Я ему дал имя **“dancers”**.

Второй аргумент (*hero*), имя героя, которому суждено активировать событие. Тут, в отличие от предыдущих случаев, я в своей карте этот аргумент обозначил... Только герой Дираэль сможет его активировать (скриптовые имена героев уже прописаны, где их посмотреть написано в руководстве “От чайника к чайнику”).

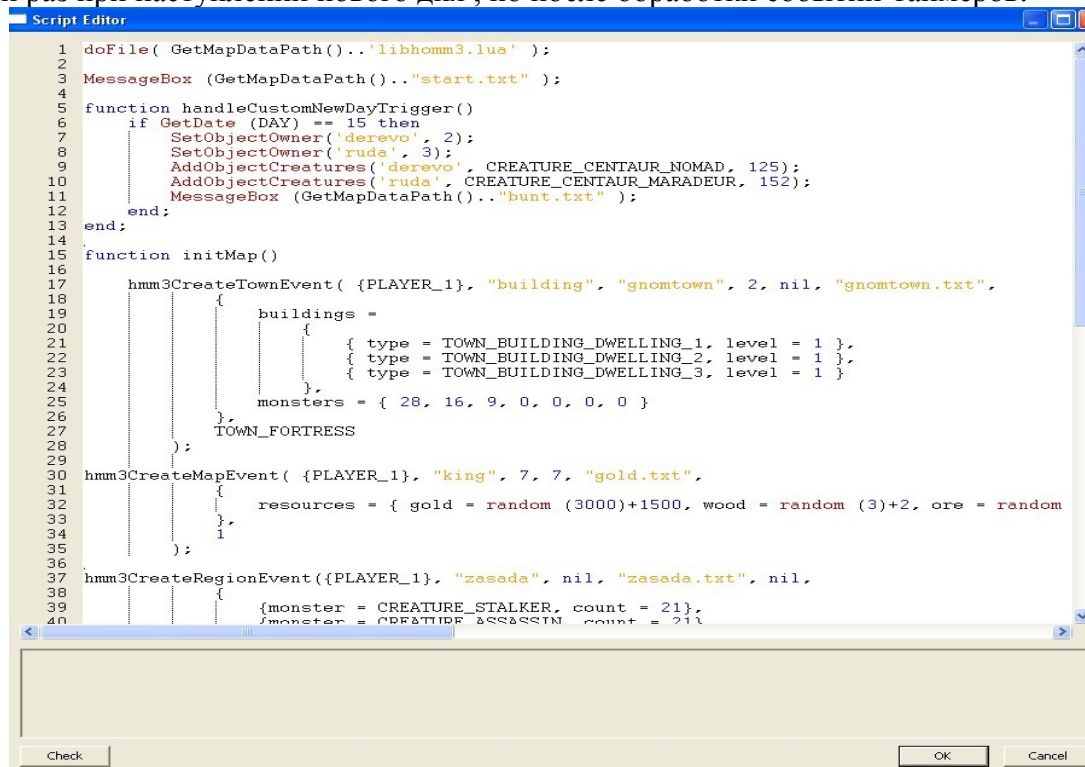
Третьим аргументом (*pre_message*) выступает текст, который появится перед событием.

И последнее (*reward*)... Награда. Записывается, как и раньше:)

Таким образом мы рассмотрели все команды. Как вы видите, это не заняло очень много время и страшным выглядит лишь на первый взгляд.

Кроме того, у каждого может возникнуть желание выйти за пределы этих событий и дописать парочку своих скриптовых команд. И тут вы ничем не ограничены. Просто помните, что **если вы используете функцию библиотеки для создания события для объекта или региона, то не следует использовать для работы с этим объектом / регионом какие-либо иные скрипты.**

Также, если вы захотите пользоваться триггером NEW_DAY_TRIGGER, просто создайте функцию с именем *handleCustomNewDayTrigger()* и поместите код обработчика в нее, как это сделал я в своей карте. Если такая функция есть в вашем скрипте, то она будет вызываться каждый раз при наступлении нового дня, но после обработки событий-таймеров.



Ещё раз хочу отметить, что это руководство призвано помочь освоиться в работе с библиотекой. Основное и главное руководство, которое содержит все команды, аргументы и тонкости написал Ungerade.

Желаю удачи!