HoMM3 events library

version: 0.90-beta

Написано: Максим Ефремов (Ungerade)
umgerade@yandex.ru

Зачем это нужно?

Не так давно, я закончил прохождение аддона «Повелители Орды», а до этого с удовольствием прошел оригинальную HoMM5. После этого решил ознакомится с редактором карт. При ближайшем рассмотрении оказалось, что удобство создания игрового мира на высоте, создание ландшафтов – очень интересное и не особо сложное занятие. Основная сложность, как оказалось, заключалась в новой скриптовой системе HoMM5, основанной на языке lua. Понятное дело, для человека, чье знакомство с программированием ограничилось школьным изучением BASICa, программирование карт для (пусть и любимой) игры занятие сложное и непонятное (начинать всегда сложно). В HoMM3 была простая и понятная каждому система событий (events), которая позволяла сделать карту более интересной и не ограничивать интерес к ней только лишь ее дизайном. Пятые герои дают картостроителю несоизмеримо больше возможностей, но, как водится, чем больше дают, тем сложнее взять.

Поэтому, у меня возникла идея создания библиотеки готовых скриптов на разные случаи жизни. Затем эта идея, с подачи **Ogo,** переросла в идею создания библиотеки скриптов, эмулирующих события из HoMM3. На данный момент готова, так называемая **бета-версия** библиотеки. Это не окончательная версия, и она может содержать ошибки. Я, разумеется, проверял ее в рамках своих возможностей, но, к сожалению, автор скрипта по ряду причин не может выявить и устранить всех ошибок. Библиотека проверена еще двумя людьми: **Ogo** и **ArchWarlock.** На данный момент основные ошибки устранены, работа скриптов библиотеки проверена как на однопользовательских картах так и на мультиплейерных. Однако отдельные баги могут присутствовать.

Поэтому, **если вы используете мою библиотеку и нашли в ней ошибку, напишите мне о ней, пожалуйста, на e-mail (или личным сообщением на форуме forum.ag.ru) и я постараюсь ее исправить. Также приветствуются советы по улучшению/расширению библиотеки.**

Кому будет полезна моя библиотека? В первую очередь я писал ее для людей, которым, в основном, хватало функционала тройки в плане реализации сценария. Надеюсь, что и другим она тоже будет полезна. Я по мере своих скромных сил постараюсь расширять библиотеку (в разумных пределах). И надеюсь на вашу помощь в выявлении ошибок и поиске новых идей.

Кто может использовать мою библиотеку? Любой человек, который заинтересован в возможностях, которые она предоставляет. Отмечу лишь, что используете вы ее на свой страх и риск, и я не могу гарантировать, что она будет корректно работать у вас на компьютере.

Благодарности

Особую благодарность хочу выразить:

- **Илье Огоновскому (Ogo) –** за идею создания библиотеки; за создание демонстрационной карты и руководства к ней; за советы и найденные баги; и за всю остальную бесценную помощь, которую я забыл упомянуть;
- **Ивану Замиралову (ArchWarlock) –** за тестирование работы библиотеки в мультиплейере, ценные советы и найденные баги;

- Евгению Германову (EyeGem) - за помощь в постижении тонкостей скриптописания и проявленное терпение в оказании оной.

Ограничения и совместимость с другими скриптами

Во-первых, данная библиотека использует некоторые функции, которые появились в **версии игры 3.0** (аддон «Повелители орды»). Это означает, **что с более ранними версиями игры библиотека работать не будет!** Совместимость с более поздними версиями будет обеспечиваться по мере выхода патчей к игре.

Функции библиотеки используют малую толику скриптовых возможностей HoMM5. В частности, используются триггеры OBJECT_TOUCH_TRIGGER, REGION_ENTER_AND_STOP_TRIGGER, NEW_DAY_TRIGGER и ряд функций. Общее условие использования функций библиотеки таково. **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. Если вы используете любое событие-таймер из библиотеки, то вам не следует назначать свой обработчик триггеру NEW_DAY_TRIGGER. Если вам крайне необходим такой обработчик, то ниже в документации описано, как его создать.** Других ограничений нет. Вы можете создавать свои скрипты любой степени сложности и использовать их вместе с библиотекой (но не для объектов/регионов, для которых вы создали события средствами библиотеки).

Модель событий (events) в HoMM3

Всего в тройке было пять различных событий: событие для региона, событие для монстра, событие для сокровища, событие-таймер для города, событие-таймер для карты. Ниже кратко описано каждое из них.

1. **Событие для региона.**
   Правильнее было бы называть его событием для тайла (клетки игрового поля), поскольку регионов в тройке не было и событие размещалось ровно на один тайл. Событие возникало в тот момент, когда герой приходил на указанный тайл. Состояло оно из трех этапов, каждый из которых мог быть опущен. На первом выводилось текстовое сообщение о том, что произошло. На втором этапе, в том случае если задавалась охрана (guard) тайла, стартовало тактическое сражение армии героя с этой охраной. И на третьем, в том случае, если герой выиграл это сражение (либо его не было), герой получал бонусы (как положительные, так и отрицательные). Среди бонусов: опыт, удача, мораль, первичные и вторичные навыки, мана, существа, артефакты, ресурсы и заклинания. Существовала также возможность выбрать игроков, для героев которых могло наступить событие. Также можно было установить, будет ли событие повторяться каждый раз, когда герой пересекает тайл, или же закончится после первого возникновения. Применялось это событие для информационного сопровождения игрока при перемещении им своих героев по стратегической карте, а также для организации засад монстров, внезапно нападающих на героев.

2. **Событие для монстра.**
   Событие задавалось в свойствах отряда существ, который был размещен на стратегической карте. Событие состояло из трех этапов. Первый – вывод текстового сообщения игроку, второе – взаимодействие с отрядом существ (бой или присоединение, в зависимости от настроек поведения отряда существ), и третье – выдача награды герою. Событие возникало для героев всех игроков, без исключения. Применялось для того, чтобы разнообразить взаимодействие игрока с определенными отрядами существ на стратегической карте.

3. **Событие для сокровища.**
   Событие задавалось в свойствах сокровища (артефакта или ресурсов), размещенного на стратегической карте. Позволяло задать охрану для сокровища. Состояло из трех этапов, но возникало только в том случае, если было задано сообщение-вопрос и охрана сокровища. В противном случае происходило обычное взаимодействие (герой подбирал сокровище). На первом этапе выводилось текстовое сообщение (в диалоговом окне с кнопками «Да», «Нет»), в котором игроку предлагалось сразиться за сокровище. В случае согласия игрока, стартовала битва с охраной. Если герой выигрывал бой, то он подбирал сокровище, в противном случае погибал, а сокровище оставалось на стратегической карте. Применялось для задания дополнительной охраны на артефакты или ресурсы.

4. **Событие-таймер для города.**
   В отличие от событий, описанных выше, это событие возникало не в результате взаимодействия героя с каким-либо объектом, а в результате наступления определенного игрового дня. Задавалось оно в свойствах конкретного города. Возникало в начале хода определенного игрока (или игроков) и состояло из двух этапов. На первом этапе выводилось текстовое сообщение, информирующее игрока о событии. На втором, игроку выдавались бонусы (для ресурсов как положительные, так и отрицательные) трех типов: ресурсы, строения и существа для найма. Это событие позволяло управлять скриптовой отстройкой города, корректировать баланс при помощи изменения количества существ для найма, а также выдавать игроку ресурсы. Можно было указать игроков, для которых наступало данное событие (для остальных не наступало). Также для события мог быть указан временной интервал повтора (в днях). В случае его установки событие повторялось до конца игры через заданный промежуток времени.

5. **Событие-таймер для карты.**
   Аналогично предыдущему, это событие возникало при наступлении определенного игрового дня. Задавалось оно в свойствах карты и позволяло выводить информационные сообщения о ходе сюжета, а также изменять количество ресурсов у игрока (в большую или меньшую стороны). Для этого события можно было выбрать игроков, для которых оно наступает, и указать интервал повторения.

Инсталляция библиотеки

Для инсталляции библиотеки необходимо поместить ее в архив карты и подключить к основному скрипту. Делать это необходимо так.

1. Скопировать файлы libhomm3.lua, libhomm3.xdb из дистрибутива в архив карты, в каталог, в котором лежит основной скрипт карты (карта – это обычный zip архив).

2. Чтобы подключить библиотеку к основному скрипту используется следующая конструкция. В основном скрипте пишем.

```
doFile( GetMapDataPath()..”libhomm3.lua” );

function initMap()

-- здесь будут все вызовы функций библиотеки
```

end;

startThread(initMap);

После этого библиотека подключена, и можно приступать к созданию событий. Для проверки того, правильно ли вы подключили библиотеку, в архиве дистрибутива есть тестовый скрипт test.lua. После подключения библиотеки просто скопируйте его содержимое в файл основного скрипта и запустите карту. В начале игры первый игрок должен получить по 777 единиц каждого ресурса.

Данный способ подключения прост, но если по какой-либо причине он не заработал, вы можете попробовать альтернативный способ, описанный в руководстве к демонстрационной карте **Ogo,** которое находится в дистрибутиве библиотеки.

Функции библиотеки

Каждое из событий, описанных выше, задается при помощи вызова ровно одной функции библиотеки. При написании функций, я старался сделать их использование как можно более простым для картостроителя, дабы облегчить насколько возможно его непростой труд. Более того, в дистрибутиве имеется демонстрационная карта и руководство к ней, созданные **Ogo,** которые содержат различные примеры использования функций библиотеки для создания событий.

Для тех, кто плохо знаком с программированием. Данное руководство скорее теоретическое, нежели практическое, и написано человеком, который зарабатывает себе на жизнь программированием. Вероятно, отдельные моменты недостаточно подробно описаны, а много описано слишком сложным языком. Если вам написанное ниже покажется излишне сложным, то воспринимайте это как справочный материал, к которому можно обращаться, а можно и не обращаться. Для работы с библиотекой это не обязательно. В составе дистрибутива имеется демонстрационная карта и руководство к ней, в котором **Ogo** максимально доступным и понятным (и мне увы недостижимым) языком описал как пользоваться библиотекой. В случае возникновения затруднений я всегда рад буду помочь. Задавайте вопросы по электронной почте и личными сообщениями на форуме **AG.RU.**

О массивах. Опять же, для тех, кто не очень хорошо знаком с программированием, хочу пару слов сказать о массивах. Ряд аргументов функций библиотеки задается в виде массивов, поэтому для правильного написания вызовов функций необходимо знать основы. Постараюсь описать максимально простым языком. Простой массив задается последовательностью значений в фигурных скобках, разделенных запятой. Например, так:

local colors = { “красный”, “зеленый”, “синий” };

Переменная *colors* представляет собой простой массив, можно понимать это как ящик, в котором хранятся значения. Для того, чтобы обратиться к определенной ячейке массива (добраться до информации, которая в нем содержится), необходимо знать индекс (их еще называют ключами) этой ячейки. В случае простого массива индексами являются положительные целые числа. Так элемент «красный» имеет индекс 1, элемент «зеленый» - индекс 2, элемент «синий» - индекс 3. Чтобы в скрипте получить значение «синий», мы должны написать вот так

local blue = colors[3];

Индекс записывается в квадратных скобках после имени массива. Переменная *blue* после выполнения скрипта будет иметь значение «синий». Аналогично, *colors[1]* имеет значение «красный», а *colors[2]* – «зеленый». Довольно просто, не правда ли?

Немного более сложны, так называемые, ассоциативные массивы. В отличие от обычных массивов индексами для них служат не целые числа, а строки символов. Например, вышеупомянутый массив *colors* можно записать в виде

local colors = { red = “красный”, green = “зеленый”, blue = “ синий” };

Индексы такого массива пишутся слева от знака равно, а соответствующие им значения – справа. Как и прежде, элементы массива разделяются запятыми и обрамлены фигурными скобками. Чтобы получить значение из массива применяется вот такая конструкция:

local blue = colors[‘blue’];

В квадратных скобках записано значение индекса, обрамленное апострофами (отмечу, что в записи массива индексы апострофами не обрамляются). Переменная *blue* после выполнения данного скрипта будет иметь значение «синий». Аналогично *colors[‘green’]* равно «зеленый», а *colors[‘red’]* – «красный». Тоже не очень сложно.

Пустой массив (массив, в котором нет элементов) записывается вот так *{}*.

Массивы могут быть вложенными, т.е. элементом одного массива может быть другой массив. Например,

```
local colors = {
        en = {“red”, “green”, “blue”},
        ru = {“красный”, “зеленый”, “синий” }
};
```

Обращение к элементам такого массива также осуществляется через индекс. Например, переменная

local color_en = colors[‘en’];

после выполнения скрипта принимает в качестве значения массив *{“red”, “green”, “blue”}*, а переменные

```
local blue_en = colors[‘en’][3];
local blue_en_alt = color_en[3];
```

значение «синий».

Вот и все, что я хотел сказать о массивах. Больше узнать вы сможете из соответствующей литературы. Скажу лишь, что обращаться по индексам к элементам массивов при написании вызовов функций библиотеки вам необязательно. Достаточно уметь записать массив в соответствии с приведенным шаблоном. Всегда следите за тем, чтобы число открывающих фигурных скобок в записи массива равнялось числу закрывающих. И помните, что индекс и ключ в данном руководстве, если речь идет о массивах, - это одно и тоже.

Ну а теперь подробное описание всех функций и их аргументов.

Событие для региона

Описание функции.
Задается событие при помощи вызова функции

hmm3CreateRegionEvent(players, region, hero, pre_message, post_message, guard, reward, once);

Аргументы функции таковы:

players – массив идентификаторов игроков, для героев которых может возникнуть данное событие. Например, массив ***{ PLAYER_1, PLAYER_6 }*** ограничивает наступление события для первого и шестого игроков. Значение ***nil*** для данного аргумента недопустимо. Если вы хотите задать наступление события для единственного игрока, его идентификатор все равно нужно передавать внутри массива, вот так: ***{ PLAYER_1 }.***

region – строка, уникальное имя региона, которое задается в редакторе при его создании.

hero – строка, уникальное имя героя, которое задается в его свойствах, в редакторе. Не путайте с его игровым именем (Зехир, Раилаг и т.п.). Если задано, то событие наступает только для этого героя. Если вы хотите, чтобы событие наступало для всех героев игрока, установите значение этого аргумента в ***nil***.

pre_message – строка, название текстового файла, содержащего информационное сообщение, которое будет выведено на первом этапе события. Например «hello_world.txt». Внимание, не задавайте полный путь к файлу, только его название! Аргумент может быть ***nil***.

post_message – строка, название текстового файла, содержащего информационное сообщение, которое будет выведено после сражения героя с засадой (и в случае его победы). Например «hello_world.txt». Внимание, не задавайте полный путь к файлу, только его название! Аргумент может быть ***nil***. Это мое нововведение, в тройке его не было, но оно может быть полезно в отдельных случаях.

guard - массив, задает охрану. Задается в виде массива массивов

```
{
    {monster = <идентификатор монстра>, count = <количество>},
     ...
    {monster = <идентификатор монстра>, count = <количество>}
}
```

Каждая внутренний массив задает ровно один стек. **Максимально допустимое количество стеков – 7**, но их может быть и меньше. Если вы не хотите задавать охрану, установите этот аргумент в ***nil***. Не передавайте в качестве аргумента пустой массив – это приведет к ошибке скрипта. Пример, массив

```
{
    {monster = CREATURE_CYCLOP, count = 10},
    {monster = CREATURE_GOBLIN, count = 200}
}
```

задает охрану в один стек гоблинов (200 штук) и один стек циклопов (10 штук).

reward – массив, задает награду, которая выдается герою на заключительном этапе события. Этот массив ассоциативный, грубо говоря, это означает, что индексируется он не целыми числами, а строками символов (индексами или ключами). Для данного массива множество ключей ограничено и каждый ключ означает задание определенного бонуса. Формат таков

{
<ключ1> = <значение1>,
<ключ2> = <значение2>,
...
<ключn> = <значениеn>
}

Допустимые ключи (индексы) и их значения перечислены ниже.

expo – задает опыт, который будет выдан герою. Формат таков

expo = <количество опыта>

Например, запись expo = 1000 означает, что герою будет выдано 1000 очков опыта. Количество опыта не может быть отрицательным.

mana – задает очки маны, которые будут выданы герою (или отобраны у него). Формат таков

mana = <количество маны>

Например, запись mana = 10 означает, что герою будет выдано 10 очков маны. Количество маны может быть отрицательным, в этом случае очки маны будут отняты у героя.

morale– задает очки морали, которые будут выданы герою (или отобраны у него). Формат таков

morale = <количество морали>

Например, запись morale = 5 означает, что мораль героя увеличится на 5 единиц. Количество маны может быть отрицательным, в этом случае мораль героя понижается. **Из-за текущей реализации скриптов в HoMM5 мораль выдается герою перманентно (на все оставшееся время), и не будет возвращена в первоначальное состояние после первой битвы. Ситуация будет исправлена после выхода патча игры версии 3.1.**

luck – задает очки удачи, которые будут выданы герою (или отобраны у него). Формат таков

luck = <количество удачи>

Например, запись luck = 2 означает, что удача героя увеличится на 2 единицы. Количество удачи может быть отрицательным, в этом случае удача героя понижается **Из-за текущей реализации скриптов в HoMM5 удача выдается герою перманентно (на все оставшееся время), и не будет возвращена в первоначальное состояние после первой битвы. Ситуация будет исправлена после выхода патча игры версии 3.1.**

resources – ассоциативный массив, задает ресурсы, которые будут выданы владельцу героя и их количество. В общем случае имеет вид

resources = { wood = <количество>, ore = <количество>, gem = <количество>, crystal = <количество>, sulfur = <количество>, mercury = <количество>, gold =

<количество> }

Ключи данного массива совпадают с идентификаторами ресурсов, но в отличие от них записываются в нижнем регистре (маленькими буквами). Количество может быть, как положительным (ресурсы выдаются), так и отрицательным (ресурсы изымаются). В массиве ресурсов допускается пропуск отдельных ключей, однако задавать в качестве аргумента пустой массив запрещено. Например, передавая в качестве параметра массив

resources = { wood = 50, ore = -50 }

мы говорим, что игроку будет выдано 50 единиц дерева и изъято у него 50 единиц руды (или вся руда, если ее меньше 50) . Не задавайте пустой массив *{}*! Если вы не хотите, чтобы игроку выдавались ресурсы, просто пропустите ключ **resources** массива ***reward.***

primary_skills – ассоциативный массив, задает изменение первичных навыков героя (атаки, защиты, колдовства и знания). В общем случае имеет вид

primary_skill = { attack = <количество>, defence =<количество>, spellpower = <количество>, knowledge = <количество> }

Ключи данного массива отвечают за изменение: attack – атаки, defence – защиты, spellpower – колдовства, knowledge – знания. Количества могут быть, как положительными (навык увеличивается), так и отрицательными (навык теряется). Если вы не хотите менять какой-либо параметр, просто пропустите соответствующий ключ. Но, не задавайте в качестве значения **primary_skills** пустой массив. Если вы не хотите изменять первичные навыки героя, просто пропустите ключ **primary_skills** массива ***reward.***

secondary_skills – массив, содержит идентификаторы вторичных навыков (навыков и умений). Общий вид

secondary_skills = {<список идентификаторов> }

Например, такой массив

secondary_skills = {SKILL_LUCK}

говорит о том, что герой выучит навык Призрачная Удача. Не задавайте в качестве значения **secondary_skills** пустой массив. Если вы не хотите выдавать герою вторичные навыки, просто пропустите ключ **secondary_skills** массива ***reward.***

artefacts – массив, содержит идентификаторы артефактов, которые получит герой. Общий вид

artefacts = {<список идентификаторов> }

Например, такой массив

artefacts = { ARTIFACT_GRAAL, ARTIFACT_ANGEL_WINGS }

говорит о том, что герой получит артефакты Грааль и Крылья Ангела. Не задавайте в качестве значения **artefacts** пустой массив. Если вы не хотите выдавать герою артефакты, просто пропустите ключ **artefacts** массива ***reward.***

spells – массив, содержит идентификаторы заклинаний, которые получит герой. Общий вид

spells = {<список идентификаторов> }

Например, такой массив

spells = {SPELL_HASTE}

говорит о том, что герой получит заклинание Ускорение. Не задавайте в качестве значения **spells** пустой массив. Если вы не хотите выдавать герою артефакты, просто пропустите ключ **spells** массива ***reward.***

creatures – массив массивов, задает стеки существ, которые будут получены героем. Формат массива полностью соответствует формату массива **guard**, аргумента функции *hmm3CreateRegionEvent* за одним исключением: вы можете задавать более семи стеков существ.

На этом описание сложного (на первый взгляд) массива **reward** закончено. На самом деле сложности никакой нет, просто необходимо немного опыта.

once – флаг однократности события. Если вы хотите, чтобы событие возникало ровно один раз, установите значение этого аргумента в 1. Иначе в **nil** (событие будет повторяться каждый раз, когда герой попадет в регион). **Вы не можете задать количество наступлений события с помощью этого флага, оно либо наступает ровно один раз (первый), либо наступает каждый раз при возникновении необходимых для его активации условий (герой заехал в регион).**

Как обрабатывается событие?
Обрабатывается событие следующим образом. Как только герой одного из игроков указанных в массиве ***players*** входит в регион возникает событие. Если указано имя конкретного игрока ***hero***, то событие возникнет только для него. Далее, если указано имя файла ***pre_message***, содержимое этого файла будет выведено в диалоге с кнопкой ОК. После этого, если задана охрана региона, стартует битва. В случае, если герой побеждает в битве, либо битвы не происходит, наступает заключительный этап события. если указано имя файла ***post_message***, содержимое этого файла будет выведено в диалоге с кнопкой ОК. После этого герою будет выдана награда в соответствии с массивом ***reward***. Событие наступит ровно один раз для первого героя попавшего в регион, если аргумент ***once*** не равен **nil**. В противном случае событие будет возникать каждый раз, когда герой одного из указанных игроков попадает в регион.

Как создавать событие?
Событие создается в три шага.

1. Создайте регион средствами редактора карт и дайте ему имя.
   Например, назовем его «reg_ambush».
2. Создайте с помощью редактора карт текстовые файлы сообщений и введите тексты сообщений.

Например, мы хотим, чтобы сообщение выдавалось до боя. В таком случае создаем текстовый файл «reg_ambush_msg.txt» и записываем в него текст сообщения.

3. Пропишите в основном скрипте карты вызов функции *hmm3CreateRegionEvent* и задайте ее аргументы.
   Например, такой вызов

```
hmm3CreateRegionEvent({PLAYER_1}, "reg_ambush", nil, "reg_ambush_msg.txt", nil,
                      {{monster = CREATURE_WITCH, count = 1}},
                      {
                          morale    = -5,
                          luck      = -5,
                          resources = {wood = 100},
                          artefacts = {ARTIFACT_GRAAL },
                      },
                     1);
```

Когда любой герой первого игрока попадет в регион, произойдет следующее. Будет показано текстовое сообщение в диалоге с кнопкой ОК. После того, как игрок закроет диалог, стартует бой героя с одним стеком бестий в количестве 1 единицы. Если герой выиграет этот бой, то получит следующую награду: мораль и удача понизятся на 5 очков; 100 единиц дерева; артефакт Грааль. Событие возникнет только один раз, для первого героя первого игрока, который попадет в регион.

Событие для монстра

Описание функции.
Задается событие при помощи вызова функции

hmm3CreateMonsterEvent(monster, hero, pre_message, reward)

Аргументы функции таковы:

monster - строка, имя монстра, для которого будет создано событие. Задается в его свойствах, в редакторе. Не путайте с идентификатором монстра или его названием (пример, “goblin” – имя, CREATURE_GOBLIN – идентификатор, “Гоблин” - название)!

hero – строка, уникальное имя героя, которое задается в его свойствах, в редакторе. Не путайте с его игровым именем (Зехир, Раилаг и т.п.). Если задано, то событие наступает только для этого героя. Если вы хотите, чтобы событие наступало для всех героев, установите значение этого аргумента в ***nil***.

pre_message – строка, название текстового файла, содержащего информационное сообщение, которое будет выведено перед взаимодействием с монстром. Например «hello_world.txt». Внимание, не задавайте полный путь к файлу, только его название! Аргумент может быть ***nil***.

reward – массив, задает награду, которая выдается герою на заключительном этапе события. Если монстр настроен агрессивно и герой проиграет бой с ним, то награда выдана не будет. Формат полностью идентичен формату массива ***reward*** для функции *hmm3CreateRegionEvent.*

Как обрабатывается событие?
Событие обрабатывается следующим образом. Возникает при взаимодействии любого героя любого игрока с монстром ***monster***, для которого задано событие. Если задано имя героя ***hero***,

то событие возникнет только для него (если задано **nil**, то для любого героя). Если указан файл с сообщением ***pre_message***, то выводится диалог с кнопкой ОК, который содержит данное сообщение. После того, как игрок закроет диалог, происходит взаимодействие с монстром. Оно зависит от настроек поведения и может привести к битве, к бегству монстра или его присоединению. В случае, если герой выиграл битву или битвы не было (монстр присоединился или убежал), он получает награду согласно массиву ***reward***.

Как создавать событие?
Событие создается в три шага.

1. Поместите монстра на стратегическую карту с помощью редактора и дайте ему имя. Например, назовем его «goblin» и сделаем его always join (всегда присоединяется).
2. Создайте с помощью редактора карт текстовый файл сообщения и введите текст этого сообщения.
   Например, мы создаем текстовый файл «goblin_msg.txt» и записываем в него текст сообщения.
3. Пропишите в основном скрипте карты вызов функции *hmm3CreateMonsterEvent* и задайте ее аргументы.
   Например, такой вызов

```
hmm3CreateMonsterEvent( "goblin", nil, "goblin_msg.txt",
                        {
                                spells = {SPELL_HASTE}
                        }
);
```

В игре это выглядит следующим образом. Первый герой, который нападет на данный стек гоблинов, активизирует событие. Сперва будет показан диалог с текстом, затем стек гоблинов присоединится к армии героя и герой получит заклинание «Ускорение».

Событие для сокровища

Описание функции.
Задается событие при помощи вызова функции

hmm3CreateTreasureEvent (treasure, hero, pre_message, guard)

Аргументы функции таковы:

treasure **-** строка, уникальное имя сокровища (ресурса или артефакта), для которого будет создано событие. Задается в его свойствах, в редакторе.

hero – строка, уникальное имя героя, которое задается в его свойствах, в редакторе. Не путайте с его игровым именем (Зехир, Раилаг и т.п.). Если задано, то событие наступает только для этого героя. Если вы хотите, чтобы событие наступало для всех героев, установите значение этого аргумента в ***nil***.

pre_message – строка, название текстового файла, содержащего вопросительное сообщение, которое будет выведено перед взаимодействием с монстром. **Сообщение должно быть в виде вопроса, допускающего ответы «да» или «нет»**. Например «hello_world.txt». Внимание, не задавайте полный путь к файлу, только его название! Аргумент может быть ***nil***.

guard - массив, задает охрану сокровища. Формат массива полностью идентичен формату массива ***guard*** для функции *hmm3CreateRegionEvent.* Аргумент может быть ***nil***.

Как обрабатывается событие?
Событие возникает, если герой пытается подобрать сокровище ***treasure***, для которого создано событие. Если задано имя героя ***hero***, то событие возникнет только для него. Если указан файл с сообщением ***pre_message*** и задана охрана ***guard***, то выводится диалоговое окно с кнопками ОК и Отмена, который содержит текст вопроса. Например, игроку задается вопрос, желает ли он сразиться с монстрами за данное сокровище. Если игрок согласен – стартует битва, выиграв которую, герой подбирает сокровище. Если герой не согласен сражаться, либо проигрывает битву – сокровище остается на карте. Если хотя бы один аргумент ***pre_message*** или ***guard*** имеет значение ***nil,*** то сообщение не выводится и герой просто подбирает сокровище.

Как создавать событие?
Событие создается в три шага.

1. Поместите сокровище (ресурсы или артефакт) на стратегическую карту с помощью редактора, и дайте ему имя.
   Например, поместим на карту 100 единиц дерева и назовем его «res_wood».
2. Создайте с помощью редактора карт текстовый файл сообщения и введите текст этого сообщения. Сообщение должно быть в форме вопроса, допускающего ответы «да» или «нет».
   Например, мы создаем текстовый файл «res_wood_msg.txt» и записываем в него текст сообщения.
3. Пропишите в основном скрипте карты вызов функции *hmm3CreateTreasureEvent* и задайте ее аргументы.
   Например, такой вызов

```
hmm3CreateTreasureEvent( "res_wood", nil, "res_wood_msg.txt",
                         {
                             {monster = CREATURE_TREANT, count = 50},
                             {monster = CREATURE_TREANT, count = 50}
                         }
);
```

В игре это выглядит следующим образом. Первый герой, который попытается подобрать дерево, активизирует событие. Игроку будет показан диалог с вопросом, желает ли он сразится за дерево. В случае согласия за дерево будут сражаться два стека энтов по 50 единиц в каждом. Если герой выиграет бой, то он подберет 100 единиц дерева.

Событие-таймер для карты

Описание функции.
Задается событие при помощи вызова функции

hmm3CreateMapEvent(players, name, day, reply, message, reward)

Аргументы функции таковы:

players – массив идентификаторов игроков, для которых может возникнуть данное событие. Например, массив ***{ PLAYER_1, PLAYER_6 }*** ограничивает наступление события для первого

и шестого игроков. Значение ***nil*** для данного аргумента недопустимо. Если вы хотите задать наступление события для единственного игрока, его идентификатор все равно нужно передавать внутри массива, вот так: ***{ PLAYER_1 }.***

name – строка, уникальное имя события. Задается вами непосредственно в вызове функции. Например: «map_event_day5».

day – целое число, день первого возникновения события.

reply – целое число, интервал повторения события. Должно быть установлено в **nil** для однократных событий.

message – строка, название текстового файла, содержащего информационное сообщение о событии. Например «more_gold.txt».

reward - массив, задает награду, которая выдается игроку (игрокам) при наступлении события. Формат полностью идентичен формату массива ***reward*** для функции *hmm3CreateRegionEvent* за одним исключением. Единственным доступным ключом массива является ключ **resources**. Все остальные ключи будут проигнорированы.

Как обрабатывается событие?
Событие возникает в начале игрового дня, который имеет номер ***day***. Нумерация ведется с первого дня, первый день игры имеет номер 1. В месяце 28 дней. Например, 2й день 1й недели 2го месяца будет иметь номер 30. Вначале выводится информационное сообщение в диалоге с кнопкой ОК, а затем игроку (игрокам) выдается награда в виде ресурсов. Если задан интервал повторения ***reply***, то событие будет повторяться каждые ***reply*** дней после первого возникновения.

Как создавать событие?
Событие создается в один шаг.

1. Пропишите в основном скрипте карты вызов функции *hmm3CreateMapEvent* и задайте ее аргументы.
   Например, такой вызов

```
hmm3CreateMapEvent( {PLAYER_1}, "map_event_day5", 5, 7, " more_gold.txt",
                        { resources = { gold = 75000 } }
);
```

В игре первому игроку будет выдаваться по 75000 золота каждую пятницу, начиная с первой недели. Каждый раз перед наступлением события будет выводится текстовое сообщение, информирующее игрока о нем.

Событие-таймер для города

Описание функции.
Задается событие при помощи вызова функции

hmm3CreateTownEvent(players, name, town, day, reply, message, reward, town_type)

Аргументы функции таковы:

players – массив идентификаторов игроков, для которых может возникнуть данное событие (владельцы городов могут меняться). Например, массив ***{ PLAYER_1, PLAYER_6 }*** ограничивает наступление события для первого и шестого игроков. Значение ***nil*** для данного аргумента недопустимо. Если вы хотите задать наступление события для единственного игрока, его идентификатор все равно нужно передавать внутри массива, вот так: ***{ PLAYER_1 }.***

name – строка, уникальное имя события. Задается вами непосредственно в вызове функции. Например: «talon_event_day7».

town – строка, уникальное имя города. Задается в редакторе карт, в свойствах объекта. Не путайте с названием города. Например: «talon» - имя, «Коготь» - название.

day – целое число, день первого возникновения события.

reply – целое число, интервал повторения события. Должно быть установлено в **nil** для однократных событий.

message – строка, название текстового файла, содержащего информационное сообщение о событии. Например «bonuses.txt».

reward - массив, задает награду, которая выдается игроку (игрокам) при наступлении события. Формат массива следующий:

resources – ассоциативный массив, задает ресурсы, которые будут выданы владельцу города и их количество. В общем случае имеет вид

resources = { wood = <количество>, ore = <количество>, gem = <количество>, crystal = <количество>, sulfur = <количество>, mercury = <количество>, gold = <количество> }

Ключи данного массива совпадают с идентификаторами ресурсов, но в отличие от них записываются в нижнем регистре (маленькими буквами). Количество может быть, как положительным (ресурсы выдаются), так и отрицательным (ресурсы изымаются). В массиве ресурсов допускается пропуск отдельных ключей, однако задавать в качестве аргумента пустой массив запрещено. Например, передавая в качестве параметра массив

resources = { wood = 50, ore = -50 }

мы говорим, что игроку будет выдано 50 единиц дерева и изъято у него 50 единиц руды (или вся руда, если ее меньше 50) . Не задавайте пустой массив *{}*! Если вы не хотите, чтобы игроку выдавались ресурсы, просто пропустите ключ **resources** массива ***reward.***

guard - массив, задает существа и их количества, которые будут доступны для найма в городе после наступления события. Задается в виде массива из семи целых неотрицательных чисел.

guard = *{level1_count, level2_count, level3_count, level4_count, level5_count, level6_count, level7_count }*

На i-й позиции (нумерация начинается с единицы) стоит количество существ i-го уровня, которое прибавится к количеству существ данного уровня доступных для найма в данном городе. Если вы не хотите увеличивать количество существ, скажем первого уровня, то просто напишите 0 на первой позиции массива.

buildings - массив, задает строения, которые должны быть построены в городе. Задается в виде массива массивов

```
buildings = {
    {type = <идентификатор строения>, level = <уровень>},
     …
    {type = <идентификатор строения>, level = <уровень >}
 }
```

Здесь, идентификатор строения определяет строение, которое будет возведено до указанного уровня. Можно задавать несколько строений. Не задавайте пустой массив! Если вы не хотите ничего строить, просто пропустите ключ **buildings** массива ***reward***.

town_type – идентификатор типа города.

Как обрабатывается событие?
Событие возникает в начале игрового дня, который имеет номер ***day***. Нумерация ведется с первого дня, первый день игры имеет номер 1. В месяце 28 дней. Например, 2й день 1й недели 2го месяца будет иметь номер 30. Вначале выводится информационное сообщение в диалоге с кнопкой ОК, а затем игроку (игрокам), владельцу данного города типа ***town_type***, выдается награда согласно массиву ***reward***. Если задан интервал повторения ***reply***, то событие будет повторяться каждые ***reply*** дней после первого возникновения.

Как создавать событие?
Событие создается в два шага.

1. В редакторе поместите город на карту и задайте ему уникальное имя в свойствах. Например, дадим ему имя «talon».
2. Пропишите в основном скрипте карты вызов функции *hmm3CreateTownEvent* и задайте ее аргументы.
   Например, такой вызов

```
hmm3CreateTownEvent( {PLAYER_1}, "talon_event_day7", "talon", 7, nil, "bonuses.txt",
                        {
                            buildings =
                            {
                                { type = TOWN_BUILDING_TOWN_HALL, level = 4 },
                                { type = TOWN_BUILDING_DWELLING_7, level = 2 }
                            },
                            monsters = { 0, 0, 0, 0, 0, 0, 7 }
                        },
                        TOWN_HEAVEN
);
```

В игре, если городом «talon» владеет первый игрок, то на 7й день для него будет выведено информационное сообщение, а в самом городе будут построены Капитолий и жилище ангелов

второго уровня, кроме того, для найма станут доступны еще 7 ангелов. Событие наступит ровно один раз.

Важные замечания

1. Если на один и тот же день выпадают события для карты и события для городов, то вначале будут обработаны события для карты, а лишь затем для городов.

2. Важно понимать, что задавая событие-таймер вы тем самым назначаете обработчик триггеру NEW_DAY_TRIGGER и лишаете себя возможности назначить этому триггеру свой обработчик. Однако, у вас все еще есть возможность обрабатывать наступление нового дня. Для этого, просто создайте функцию с именем *handleCustomNewDayTrigger()* и поместите код обработчика в нее. Если такая функция есть в вашем скрипте, то она будет вызываться каждый раз при наступлении нового дня , но после обработки событий-таймеров.

3. Назначая событие монстру, региону или сокровищу, вы полностью определяете их поведение и не можете использовать их с другими скриптами.

4. В библиотеке есть и другие функции, помимо описанных в документации. Однако они предназначены для внутрибиблиотечного использования, и вызывать их не следует.

Вот и все! Жду ваших замечаний и предложений!