UO-The Expanse Quest Package 5.8.6752

written by Raist

What is Quest?

Quest lets you make interactive story games. Text adventure games like Zork and The Hitchhiker's Guide to the Galaxy. Gamebooks like the Choose Your Own Adventure and Fighting Fantasy books. You don't need to know how to program. All you need is a story to tell. Your game can be played anywhere. In a web browser, downloaded to a PC, or turned into an app.

Website*: http://textadventures.co.uk/quest*

Tutorial*: http://docs.textadventures.co.uk/quest/tutorial/*

UO-The Expanse Quest Package*: https://github.com/tass23/QuestPackage5.8.6752*

UO-The Expanse OQS*: https://www.uoexpanse.com/quests*

I was a big fan of Choose Your Own Adventure books when I was younger, and I got involved with Quest Text Adventures as a means to transcribe my books into CYA-style stories. I worked out a basic game to get a feel for the software, and had to decide if I wanted people to have to download the Quest app to play my games or if I wanted to convert them into an HTML file. I thought this would be a fun way to experience UO adventures, and after getting help from KVonGit, we ironed out some functions to work with the conversion tool QuestJS, and then I built the Offline Quest System for the UO-The Expanse website. After I released the edu Repack, I decided to also include the OQS as an optional tool for everyone to use. This required me to build new libraries that streamlined the creation process by including a few spells, items, and Mobs. I highly recommend walking through the tutorial on the Quest website before stepping into the OQS.

The Quest tutorial is here: Tutorial:
http://docs.textadventures.co.uk/quest/tutorial/

UO-The Expanse Quest Package 5.8.6752 on Github:
https://github.com/tass23/QuestPackage5.8.6752

(does not include any completed games)

To play UO-The Expanse OQS: https://www.uoexpanse.com/quests

The Interface: Quest has a nice UI to handle all your creation needs, with QuestJS serving as the conversion tool you can use later for the HTML file. On the left side of the window are all the Objects, Functions, Commands, and Timers that are created for that game. On the right side is the detail pane for whatever is selected on the left. Notice the hierarchy of the overall game itself.

The *Game* is an Object that can have attributes, but everything is created *inside* the Game. Adding Libraries for Spells and Items are added to the Game, but exist outside of the Game file (libraries have to be stored in the same folder as the Game they are added to). An "expression" is a coded text, like *This box of crayons has 24* ***+{object:crayons:crayon1}+*** *inside.* Otherwise this would just be a "messages" like, *This box of crayons has crayons inside.* Using expressions can make things players can interact with inside text blocks.

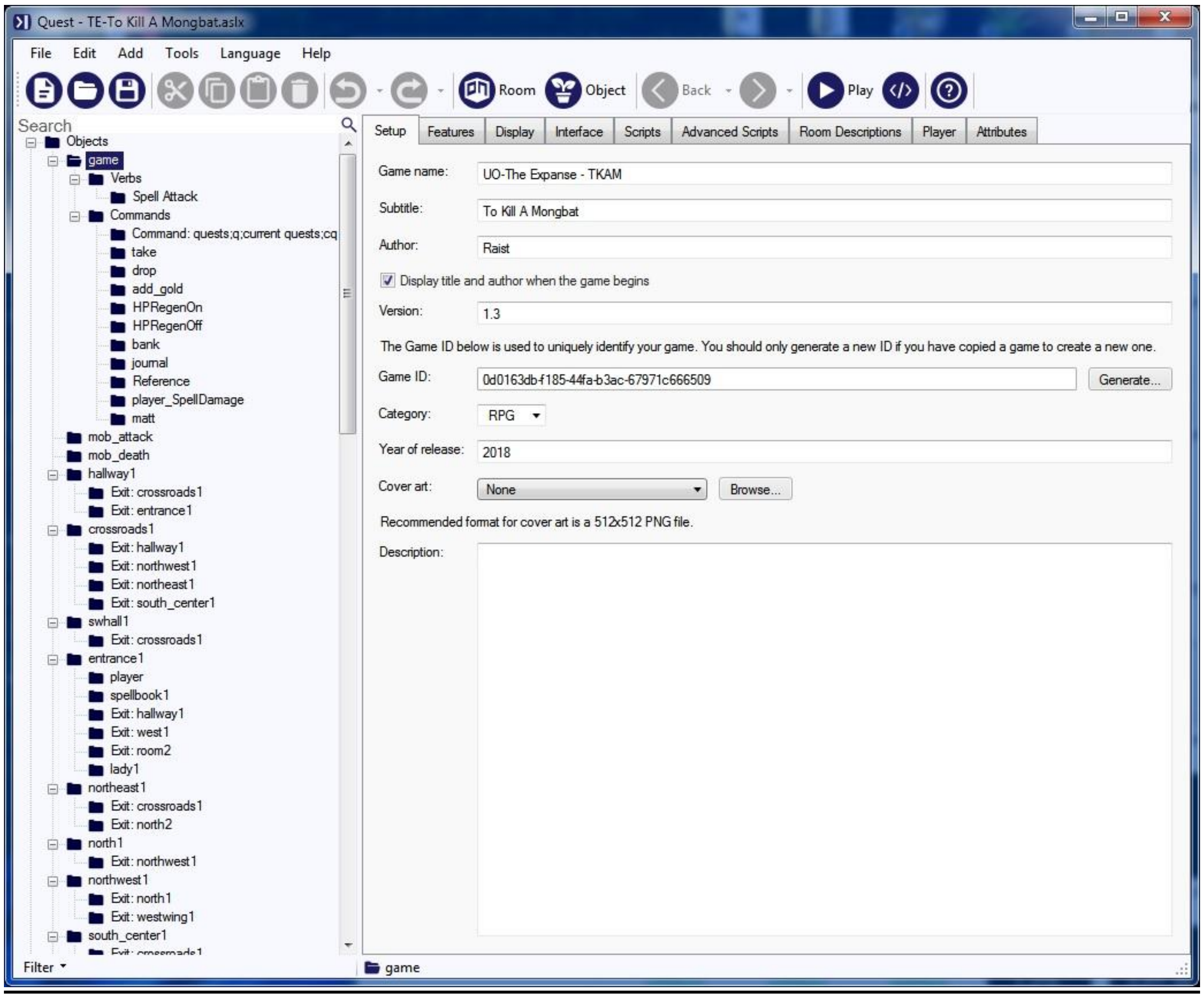
Objects vs Everything else An Object is something created when the game is launched that the player can interact with or that operates like scenery (you can *Look at* the Object, but you cannot pick it up). A Room is also an Object, but is a special type of Object that contains other Objects, like a box of crayons has individual crayon Objects inside the box.

Verbs are used to shortcut actions for players, like *Attack mongbat*, and are usually limited to an expression.

Commands are things the player types into the command bar, or when they use a hyperlink to interact with an Object in the game, and can be built with a script. Buy Object/Sell Object, that sort of thing.

Functions cause something to happen that the player does not control, but can cause. Like walking into a room, the player can use the light switch, and a function is turning on the light.

Timers are what you might expect; they use seconds to count time, and can be started when the Game starts or by a Command, a Verb, Script, or Function.


Quest - TE-To Kill A Mongbat.aslx
File Edit Add Tools Language Help
Room Object Back Play
Search
Setup Features Display Interface Scripts Advanced Scripts Room Descriptions Player Attributes
Filter
game

- Objects
  - game
    - Verbs
      - Spell Attack
    - Commands
      - Command: quests;q;current quests;cq
      - take
      - drop
      - add_gold
      - HPRegenOn
      - HPRegenOff
      - bank
      - journal
      - Reference
      - player_SpellDamage
      - matt
  - mob_attack
  - mob_death
  - hallway1
    - Exit: crossroads1
    - Exit: entrance1
  - crossroads1
    - Exit: hallway1
    - Exit: northwest1
    - Exit: northeast1
    - Exit: south_center1
  - swhall1
    - Exit: crossroads1
  - entrance1
    - player
    - spellbook1
    - Exit: hallway1
    - Exit: west1
    - Exit: room2
    - lady1
  - northeast1
    - Exit: crossroads1
    - Exit: north2
  - north1
    - Exit: northwest1
  - northwest1
    - Exit: north1
    - Exit: westwing1
  - south_center1

Game name: UO-The Expanse - TKAM

Subtitle: To Kill A Mongbat

Author: Raist

[✓] Display title and author when the game begins

Version: 1.3

The Game ID below is used to uniquely identify your game. You should only generate a new ID if you have copied a game to create a new one.

Game ID: 0d0163db-f185-44fa-b3ac-67971c666509 Generate...

Category: RPG

Year of release: 2018

Cover art: None Browse...

Recommended format for cover art is a 512x512 PNG file.

Description:

Rooms do not have to have exits, but if there is no way for the player to leave the room, they can't leave the room. Exits can be visible/invisble, locked/unlocked, have descriptions, or be "Look" only. Exits can be *one way*, meaning once you leave that room, you cannot go back.

Clones are copies of Objects. You cannot clone a Room, or command, or function, or verb, only Objects. When an Object is cloned, the original usually remains in the room it started in, and the clone would then go with the player. A "template" long sword the player buys, has specific attributes that are given to the clone that can be different from the template long sword. This is most evident with resistances on Objects for UO.

Turn Scripts are used every turn, which means after every time the player uses a Command. This is helpful for things like mob attacks that will happen automatically after every turn.

Libraries are used to store specific data that can be used in multiple games, like Spells, Items, Help files. Libraries can contain complex aspects like Game properties, Functions, Verbs, and Commands.

Object Property Tabs contain general information, and specific attributes for each Object in your game. Each of these tabs servers a different function for the Object.

Setup begins with basic information, name and alias of the Object, if the Object should be Scenery, Visible/Invisible, and the Look information.

Object handles Object aliases/abbreviations, specific Link Color, Generate/Disable automatic Verbs.

Inventory is where Take and Drop for the Object are controlled.

Verbs are where custom verbs are added with individual Assignments.

Features are specific actions the Object has, like Use/Give, is it a Container, is it Switchable, is it Edible, is it Wearable, does th Object light up the room, can the player become this Object, does this Object need initialization.

Use/Give only shows up if the Use/Give box is checked under Features, and sets up specific Uses and Gives.

Ask/Tell all Objects have Ask/Tell by default, so even a toaster can have dialogue.

Attributes houses various default variables, and custom variables of different types: String, Bool, Integer, Double, Script, String List, another Object, a Command pattern, String Dictionary, Script Dictionary, or Null.

Objects displays Objects *inside* this Object, like a container, but not all containers must be defined as a container. It depends entirely on the usage. A container like a Quiver would have arrows with a quantity and does not *have to be* a container, but something like a Backpack that can hold multiple types of Objects should be defined as a Container under the Features tab.

Room Property Tabs contain general information, and specific attributes for each Room in your game. While similar to the Object tabs, there are a few differences, such as:

Light/Dark which allows you to set a room as *initially* dark, and set up a description script of what happens when the room is dark.

Exits are as you might guess; ways to leave the room. A compass with 8 points offers 8 exits, plus In/Out and 2 None exits. Once an exit is defined for a Room, an *Object* is created for the Exit itself, and those have specific tabs.

Scripts can be defined as events that happen *Before entering*, *After entering*, *After leaving*, *Before entering for the first time*, *After leaving the room for the first time*, and finally *Turn scripts that happen while inside that room* (as opposed to *game* Turn Scripts).

Exit Property Tabs are limited to 3.

Exit being the general information about the Exit, Locked/Unlocked, Visible/Invisible.

Options defines the Exit as a Light Source or not.

Attributes houses the default and custom variables for that Exit.
Scripting in Quest can be done with JavaScript, or using the Quest interface.

CUSTOM LIBRARIES

Spells_Lib - Words of Power, based on UO, in the form of Commands. In general, players must type *spell name object*, where *spell name* is the *Command* and *Object* is the *Target* of the *Spell*. Includes a popup window for displaying the "Spellbook" with the Uses Remaining and the Words of Power. This popup is only available IF the player is carrying the Spellbook, and the same can be said for using the Words of Power (they are only available if the player is carrying the Spellbook). Spells/Words of Power: Each spell accounts performs a check against the spell target to determine if that spell can be cast on it (mob=True/False). The next check is for resistance. UO has the following resists/damage types: Physical Fire Cold Poison Energy Each spell fits into a primary damage category, i.e. Vas Flam is Fireball, so it does Fire damage. If a mob has Fire resist lower than X, the spell is cast, does damage, and the mob's health drops accordingly. Otherwise, the spell simply fails. This system could be adjusted to subtract a percentage based on the resist value, which would allow for more variations of damage. However, since each spell has a variable damage range, a single resist check functions more easily.

TEHelp_Lib - The Expanse Offline Quest system complex HELP system.

TEItems_Lib - Weapons, based on UO, complete with magical properties. Incorporated into the Save/Load system which creates hash codes for Items to be Copy/Pasted to a text file to save it. Weapon Room contains ALL available weapons. Default images loaded for each weapon and weapon types are defined. NOTE: Additional properties have been added for Game Rooms, Mobs and Player, including Status attributes.

TEMainSysLib - The Expanse Offline Quest System Main Library contains commands, functions and other universal data needed for the OQS to function. The following is a mandatory list of Attributes to be added to objects and which type of objects receive which Attributes. *Rooms*

1. cycle INT - Used for handling respawns and increasing difficulty of respawns.
2. map & map_zoom STR - Used for handling the Map library and displaying images for each room.
3. mlist STRLIST - Used for handling the list of available targets to attack in a room.
4. Scripts - OnExit: ClearScreen, Enter: targets
5. Add Supply Shop exits to Main Room

NPC Questers

1. avquests INT - Displays Total Available quests.
2. tquests INT - Displays Total Quests for that NPC.

Mobs

1. mob type INT - Used to determine respawns and target functions.
2. lootable INT - Used to generate Loot after death. Does not work with Dispelled mobs.
3. status STR - Displays current mob status in the mob window when LOOKed at.
4. desclive STR - Message to display when mob is Alive.
5. descdead STR - Message to display when mob is dead.
6. img_dead STR - Direct path online to dead mob image.
7. defense type INT - Used during combat for Resists.

TEMap_Lib - Map library to use generic code to call pre-defined attributes on Rooms (room.map and room.map_zoom). room.map image should be full size map of that room or the whole area. room.map_zoom should be a scaled section of the main map that matches that specific room or area. Change display text as needed to match each map. Construct one map library for each game.

NPC_Lib - Easy NPC. Allows for the player to select an additional helper to join them. Complete with STATS PANE visible on the Right side, under the player STATS. Choose from Mage or Archer classes.

<u>QUEST LIBRARIES</u>

Quest_Tracker_Lib -Used to create Quests and steps of Quests, as well as Rewards. Can be used to house multiple Quests.

Score_Lib - Used to define the Score of the game, broken down by each call of the IncScore function to increase or decrease the total Score. Offers two displays of Score Results.

This section is all about *How do I?* or *Here's what I did.* As Quest is coded in JavaScript, some people will be more or less comfortable scripting. Lean into the Quest interface until you feel more comfortable. Here are some examples taken from the Custom Libraries

<u>*SPELLS_LIB*</u> <u>*Spells*</u> All Spells are Commands, and as such must be defined as a Command with the proper Command-related layout. The structure for a Command item is: Name -Pattern -Scope -Script The first Spell in the Spells_Lib is Vas Flam. This Command is executed when the player types it into the command bar, or use the Spellbook to select a Spell and then select a Target. The Command is built like this: Name: vas flam Pattern: vas flam #object# Scope: blank Script: (see - VasFlam.xml)

<u>*Timers*</u> All Timers count time in seconds, and can begin counting when the game starts, or they can be started by outside scripts, commands, verbs, objects, and functions. The structure for a Timer is: Name -Start timer when game starts -Interval -Script The first timer in Spells_Lib is poison_tick1.

As you can imagine this Timer is executed when a mob, or the player are poisoned. Since the player has a poison spell, In Nox, that Spell can start the Timer too. The Timer is built like this: Name: poison_tick1 Start timer when game starts: not checked Interval: 5 (seconds) Script: (see - Poison.xml)

NPC_LIB *Function InitUserInterface* is used to build a display when the Game starts, before the player has a turn. The NPC_Lib was specifically built in order to add a character that would fight along with the player. The structure of the function is like this: JS.eval defines a span tag for an attribute on the NPC. s is used as a variable to build the actual Stats Pane. Each value of s is added to the next. JS.setCustomStatus (s) is called to process all the s entries. If a Stat/Attribute on the NPC is changed, a script is called to change the value in the Pane. To get all these values, we have to assign them somewhere, and we do that with each function that creates the type of NPC the player picks; Mage (function npcmage1) or Archer (function npcarch1). That structure goes like this: set an attribute on the NPC. JS.eval defines the SPAN tag for the attribute a Script is added for each attribute that is supposed to change during the Game. To set up the InitUserInterface, we begin with:

```
JS.eval ("$('#weapon-span').html('" + lady3.weapon.alias + "');")
```

Then the first s value that begins the HTML Table:

```
s = "<div id='status_div' style='padding:5px;padding-top:10px;border-radius:5px;border:grey solid thin;'>PARTNER"
```

Subsequent s values add the previous s value:

```
s = s+"<table width='100%'><tbody id='status_npc'>"
```

We call:

```
JS.setCustomStatus (s)
```

to process all the s values. Then the IF statement says if the script changedweapon is attached to the NPC, it must process the script changedweapon.

```
 if (HasScript(lady3, "changedweapon")) {
     do (lady3, "changedweapon")
   }
```

Lastly we secure the NPC Stats Pane location:

```
JS.eval ("$('#status_div').insertAfter($('#compassAccordion'))")
```

This finishes off the InitUserInterface function.

Next, we setup the Stats Pane and define the Span values for the HTML tags by using one of two functions: npcmage1, or npcarch1, for mage or archer respectively, which is chosen by the player when the game begins. We will use MAGE for the example. Since the Stats Pane must be generated before the Game starts, we have a conundrum, in that we can't get the correct Stats for the Pane until after the Game starts, so we must define "fake" values as placeholders inside InitUserInterface first. Now we can move into the correct values being updated after the player makes their choice. Firstly we set the attribute:

```
set (lady3, "weapon", spellbook2)
```

Then we use JS.eval to define the Span tag.

```
JS.eval ("$('#weapon-span').html('" + lady3.weapon.alias + "');")
```

Finally we define what changedweapon script does for this attribute:

```
 lady3.changedweapon => {
     JS.eval ("$('#weapon-span').html('" + lady3.weapon.alias + "');")
   }
```

Now when the Game begins, there is a Stats Pane on the Right side, under Places and Objects, which says PARTNER: Martika, and then all the values/attributes that we wanted to display are 0.

After the Game begins and the player has made their choices, all the "changed" scripts attached to each attribute are then called and each value is updated in the Stats Pane.

TEITEMS_LIB

Items are Objects, but these particular Objects can be "equipped" by the player as weapons and that increases the player's damage, gives bonuses to resistances, hit chance, and Elemental attacks. However, if the player is not carrying their Spellbook, they can't cast Spells. Creating an Item the player can equip as a weapon follows normal Object structure, with these additions as attributes: pic type -img -addy -src -f_type -wtype -m_props -AosStrReq -AosMinDam -AosMaxDam -AttackChance -BonusDex -BonusHits -BonusInt -BonusMana -BonusStam -BonusStr -Chaos -Cold -DefendChance -Direct -Energy -Fire -HitCold -HitDispel -HitEnergyArea -HitFireball -HiteHarm - HitLeechHits -HitLeechMana -HitLeechStam -HitLightning -HitLowerAttack -HitLowerDefend -HitMagicArrow -HitPhysicalArea -LowerManaCost -Luck -MageWeapon -NightSight -Physical -Poison -ReflectPhysical -RegenHits -RegenMana -RegenStam -ResistColdBonus -ResistColdBonus -ResistEnergyBonus -ResistFireBonus -ResistPhysicalBonus -ResistPoisonBonus -SpellChanneling -SpellDamage -WeaponDamage -AttackChance2 -BonusDex2 -BonusHIts2 -BonusInt2 -BonusMana2 -BonusStam2 -BonusStr2 -Chaos2 -Cold2 --DefendChance2 -Direct2 -Energy2 -Fire2 -HitCold2 -HitDispel2 -HitEnergyArea2 -HitFireball 2-HiteHarm2 -HitLeechHits2 -HitLeechMana2 -HitLeechStam2 -HitLightning2 -HitLowerAttack2 -HitLowerDefend2 -HitMagicArrow2 -HitPhysicalArea2 -LowerManaCost2 -Luck2 -MageWeapon2 -NightSight2 -Physical2 -Poison2 -ReflectPhysical2 -RegenHits2 -RegenMana2 -RegenStam2 -ResistColdBonus2 -ResistColdBonus2 -ResistEnergyBonus2 -ResistFireBonus2 -ResistPhysicalBonus2 -ResistPoisonBonus2 -SpellChanneling2 -SpellDamage2 -WeaponDamage2 -Wearable (true) -Wear_slots (lefthand) -Initialize (magical properties) All these properties are repeated on the player too. (see – AssassinSpike1.xml)