Inhaltsverzeichnis

Vorwort

Die Symfonia-QSB, oder auch QSB Plus, um ihren ursprünglichen Namen mal zu erwähnen, hat es sich zum Ziel gesetzt, das Leben des Mappers zu erleichtern. Aber wie es mit neuen Technologien stets der Fall ist, muss auch hier ein klein wenig beachtet werden, um in den Genuss der Vorzüge dieses Frameworks[1] zu kommen.

Nachfolgend werde ich die wichtigsten Dinge erklären, die Du benötigst um Symfonia optimal zu nutzen. Ich werde auf die speziellen Funktionalitäten eingehen, angefangen bei der Installation und den ersten Schritten zur Verwendung bis hin zu einigen speziellen Features.

Installation

QSB-Datei

Sobald Du eine neue Map erstellt hast, muss die QSB in die Map *importiert* werden. Siehe dazu das SEED Handbuch, falls Dir der Prozess nicht klar ist. Ein Release kommt mit *zwei* Versionen der QSB, die sich nur in einem Punkt unterscheiden: der Dateigröße.

Sobald Du in den Release-Ordner gegangen bist, wirst Du zwei verschiedene Dateien vorfinden: questsystembehavior.lua und questsystembehavior_min.lua.

> Die Datei questsystembehavior.lua ist die unveränderte QSB. Sie enthält noch alle Leerzeichen und Code-Kommentare und eignet sich zum stöbern, sollte Dir die Dokumentation nicht ausreichen um Deine Fragen zu klären. Sie ist allerdings sehr groß, etwa 1 Megabyte.
>
> Die Datei questsystembehavior_min.lua wurde auf ein Minimum komprimiert, sodass sie etwa um die Hälfte weniger Speicherplatz benötigt. Allerdings ist der Code durch das Entfernen von Leerzeichen und Kommentaren sehr unleserlich. Nutze diese Datei, wenn Du Speicherplatz sparen willst.

Letztlich ist es egal, welche der beiden Dateien Du importierst. Die minimierte QSB bietet dir nur den Vorteil weniger Speicher zu benötigen.

Die Mapskripte

Die QSB wird nicht mit den Standardskripten des Mapeditors funktionieren! Als erstes musst Du im Mapeditor den Expertenmodus aktivieren. Sobald das getan ist, kannst Du eigene Skripte importieren. Du findest weitere Informationen dazu im SEED Handbuch.

Im Release-Ordner befinden sich die Skripte mapscript.lua und localmapscript.lua. Diese Dateien müssen in deine Map importiert werden.

---

1 Framework: ist ein Programmiergerüst, welches vereinfacht und dem Anwender Vorteile gegenüber der nativen Möglichkeiten einer Umgebung bietet, in der er sich bewegt.

DEBUG aktivieren

Du möchtest vorher vielleicht den DEBUG aktivieren, damit Du Fehler schneller entdecken kannst. Der Debug lässt sich als Funktionsaufruf in der Mission_FirstMapAction im globalen Skript aktivieren. Die Funktion API.ActivateDebugMode ist dafür zuständig. Sie muss nach dem Aufruf von InitKnightTitleTables eingefügt werden.

Eine ausreichende Konfiguration ist folgende:

```
API.ActivateDebugMode(true, true, false, true);
```

Dies aktiviert alle Checks und die Konsole, aber nicht Quest Trace. Für mehr Informationen zum Debug findest du in der Dokumentation. Alternativ kann der Debug auch mit einem Behavior aktiviert werden, Reward_DEBUG.

Schnelles Testen

Nach *jeder* Änderung im Skript das Spiel zu verlassen, den Editor zu starten und das Skript neu importieren zu müssen, ist auf Dauer anstrengend und kostet viel Zeit. Daher bietet es sich für die Entwicklung an, das globale und lokale Skript auszulagern. Dazu wird ein Hilfsskript benötigt, das anstelle der eigentlichen Skripte in die Map importiert wird.

```
local PathToProject = "C:/MyProjects/MyMap";
if not GUI then
    Script.Load(PathToProject .. "/mapscript.lua");
else
    Script.Load(PathToProject .. "/localmapscript.lua");
end
```

Dieses Skript lädt automatisch die entsprechenden Skripte, da es zuerst prüft, ob es sich in der globalen oder lokalen Skriptumgebung befindet. Dadurch kannst Du während der Entwicklung viel Zeit sparen. Passe einfach den Pfad PathToProject an Deine Bedürfnisse an. Beachte: vor der Veröffentlichung musst Du aber die eigentlichen "richtigen" Skripte in die Map importieren.

Aufträge erzeugen

Aufträge im Questassistenten

Die QSB unterstützt natürlich wie gewohnt das Erzeugen von Aufträgen im Auftragsassistenten des Mapeditors. Allerdings gibt es keine Vorzüge, wie z.B. das weglassen von Parametern oder das einfache kopieren von Aufträgen.

Für eine Anleitung zum Auftragsassistenten des Editors konsultiere bitte das SEED Handbuch.

Aufträge im Skript

Aufträge, oder auch Quests[2], im Skript zu erzeugen, ist keine wirklich neue Idee. Eine Möglichkeit Aufträge zu generieren bot z.B. schon MachQuest an, dessen Handhabung jedoch nicht wirklich zufriedenstellend war. Neu ist allerdings, Parameter von Behavior weglassen zu können. Dies geht einher mit einer neuen Schreibweise.

Ein Beispiel für die neue Schreibweise:

```
local QuestName = API.AddQuest {
    Name        = "VisitCloister",
    Suggestion  = "Ich sollte die Mönchen des Klosters besuchen!",
    Success     = "Guten Tag, werter Herr!",

    Goal_DiscoverPlayer(4),
    Reward_Diplomacy(1, 4, "EstablishedContact"),
    Trigger_OnQuestSuccess("SomeOtherQuest", 8),
}
```

Jeder Auftrag benötigt wenigstens einen Namen. Wird kein Goal angegeben, wird automatisch Goal_InstantSuccess hinzugefügt. Fehlt der Trigger, wird Trigger_AlwaysActive den Behavior hinzugefügt.

Andererseits, falls Behavior und/oder Felder wie z.B. Success angegeben sind, kann der Name weggelassen werden. In diesem Fall wird ein automatischer Bezeichner gewählt. Auf jeden Fall wird der Name des Quests von API.AddQuest zurückgegeben und kann auch so auf andere Behavior übertragen werden.

Lokalisierung von Aufträgen

Die QSB sieht es vor, dass Texte in Deutsch und Englisch angegeben werden können. Es wird dann automatisch der deutsche Text angezeigt, handelt es sich um die deutsche Version des Spiels. Andernfalls wird der englische Text genutzt. Diese Funktionalität steht natürlich auch für Aufträge zur Verfügung.

Ein Beispiel für einen lokalisierten Auftragstext:

```
Suggestion  = {
    de = "Ich sollte die Mönchen des Klosters besuchen!",
    en = "I should visit the monks of the monastery!",
},
```

2 Quest: Ein Auftrag, der durch den Spieler erfüllt werden muss, meistens um die Handlung voranzutreiben oder Vorteile für den Spielverlauf zu erhalten.

Angaben für einen Auftrag

Aufträge haben viele mögliche Angaben. Die Eigenschaften eines Auftrags werden als Felder bezeichnet. Eigenschaften sind z.B. der Name des Auftrags. Du kannst die wichtigsten Eigenschaften eines Auftrags der nachfolgenden Tabelle entnehmen.

Eigenschaft	**Beschreibung**
Name	Über den Namen werden die Quests verwaltet
Suggestion	Der Text des Sprechers zu Beginn des Auftrages
Success	Der Text des Sprechers bei erfolgreichem Abschluss
Failure	Der Text des Sprechers im Falle eines Fehlschlages
Description	Die Beschreibung für benutzerdefinierte Auftragstypen
Visible	Erzwingt die Sichtbarkeit, wenn Suggestion nicht angegeben ist.
EndMessage	Erzwingt die Sichtbarkeit der Endnachricht.
Sender	ID des Auftraggebers
Receiver	ID des Auftragnehmers
Time	Zeit bis zum automatischen Fehlschlag/Erfolg

Die Behavior eines Quests werden als Funktionsaufrufe nach den Eigenschaften des Auftrages angegeben. Dabei ist zu beachten, dass jeder sichtbare Aufträge nur ein Goal haben kann. Auslöser, oder auch Trigger, können mehrere vorhanden sein. Sie müssen alle wenigstens einmal ausgelöst haben, damit der Auftrag startet.

Mission-Briefings und Cutscenes

DIE SIEDLER - Aufstieg eines Königreichs bietet nicht von Haus aus an, Missions-Briefings oder Kameraflüge zu erstellen. Das hat uns der Vorgänger unseres Spiel voraus! Allerdings bietet die Engine[3] des Spiels einen entsprechenden Kameramodus an, der für die Thronsaalszenen in der Kampagne genutzt wird. Das so genannte Briefing System unterteilt dabei in zwei verschiedene Arten von Briefings. Das Missions- oder Dialog-Briefing und der Kameraflug oder Cutscene.

Neu ist jetzt, das Briefings (bzw. Cutscenes) an einen Auftrag „angehangen“ werden können. Dies passiert mit den Behavior `Reward_Briefing` und `Reprisal_Briefing`. Über den Auslöser `Trigger_Briefing` kann anschließend ein *anderer* Auftrag gestartet werden, *sobald* das Briefing oder der Kameraflug *vollständig* durchgelaufen ist.

Auf diese Weise musst Du nicht mehr selbst überwachen, wenn ein Briefing beendet ist. Das System nimmt Dir diese Arbeit ab. Des weiteren können verschiedene Effekte für Briefings und Kameraflüge ein- und ausgeschaltet werden. Dazu zählen Anzeigen des Himmels oder verstecken der Grenzen von Territorien.

3 Engine: ist ein spezielles Framework für Computerspiele, das den Spielverlauf steuert und für die visuelle Darstellung des Spielablaufes verantwortlich ist. In der Regel werden derartige Plattformen auch als Entwicklungsumgebung genutzt und bringen dafür auch die nötigen Werkzeuge mit.

Das Briefing-Gerüst

Damit das BriefingSystem ordnungsgemäß arbeiten kann, muss ein Grundschema eingehalten werden. Dieses Schema werde ich nun grundlegend beleuchten.

```
local briefing = {
    barStyle = "big",
    disableGlobalInvulnerability = false,
    restoreCamera = true,
    restoreGameSpeed = false,
    skipPerPage = true,
    hideFoW = true,
    showSky = true,
    hideBorderPins = true
};
local AP, ASP, ASMC = AddPages(briefing)
```

Dies ist der Kopf des Briefings. Hier werden die Eigenschaften des Briefings notiert, ähnlich wie bei den Aufträgen. Dies ist die empfohlene Grundkonfiguration für ein Missionsbriefing, die Du getrost ohne zu Fragen übernehmen kannst. Wir wollen nicht viel von der Spielwelt zeigen, daher werden die breiten Balken verwendet. Der Nutzer soll lesen, daher muss jede Seite einzeln weggeklickt werden. Zum Schluss werden noch ein paar Schönheitseinstellungen gemacht und die Kameraposition vor dem Briefing gespeichert. Wer will, kann auch die Spielgeschwindigkeit am Ende des Briefings wiederherstellen lassen. Dies wird notwendig, wenn Du nicht willst, dass Briefings höhere Spielgeschwindigkeiten herabsetzen.

Anschließend folgen die Deklarationen der Seiten. Für ein Dialog-Briefing empfehle ich diese Kurzschreibweise: Entity, Name des Sprechers, gesprochener Text, Dialogsicht an/aus, Action. Dies sind die Paramater für einen Aufruf von ASP. Natürlich steht es Dir frei auch die alte Schreibweise mit AP zu verwenden und dies nach belieben frei zu mischen.

```
ASP("alandra", "Alandra", "Hallo Marcus, schön Dich zu sehen.", true);
ASP("marcus", "Marcus", "Danke! Dann frisch ans Werk!", true);
ASP("alandra", "Alandra", "Lasst uns keine Zeit verlieren.", true);
```

Schlussendlich fehlt noch der Fuß.

```
briefing.finished = function()
end
return StartBriefing(briefing);
```

Im Fuß wird eine Funktion deklariert, die am Ende des Briefings ausgeführt wird. In dieser Funktion können beliebige Dinge geschehen, sie kann aber auch weggelassen werden. Wichtiger ist die letzte Zeile der Funktion. StartBriefing führt das Briefing aus und gibt die ID zurück. Diese ID wird für die Anbindung an das Questsystem benötigt.

Multiple Choice

Unter Multiple Choice versteht man einen Auswahldialog mit mehreren Optionen, aus denen der Spieler eine wählen muss. Um die einzelnen Zweige eines solchen Briefings zu trennen, werden Leerseiten und Sprünge verwendet.

Eine Leerseite ist ein Aufruf von AP ohne Argumente. Dies signalisiert, dass das Briefing zu Ende ist. Andernfalls würde einfach die nächste Seite angezeigt.

```
AP();
```

Ein Sprung bedeutet zu einer anderen Seite zu wechseln, sobald die Antwort gewählt wurde. Dazu wird die Nummer der Seite angegeben, zu der gesprungen werden soll.

```
AP(11);
```

Der Auswahldialog ist der wichtigste Teil. Hier wählt der Spieler die Optionen aus.

```
local CP = ASMC("hero", "", "Wie entscheidest Du Dich?", true,
    "Gut, ich werde es tun!", 5
    "Niemals, vergesst es!", 10
);
```

In diesem Beispiel wird zur Seite 5 gesprungen, wenn der Spieler die erste Antwort auswählt, und zur Seite 10, wenn der Spieler die zweite Antwort auswählt. Die Anzahl an Optionen ist unbegrenzt, übertreibe es aber nicht!

Anschließend an das Briefing möchtest Du vielleicht wissen, welche Antwort gewählt wurde. Dies geschrieht mit der Funktion MCGetSelectedAnswer. Ihr wird als Argument die erzeugte Seite übergeben, die von ASMC zurückgegeben wurde. Du erhälst die Nummer der gewählten Antwort, beginnend bei 1 für die erste Antwort.

```
if  MCGetSelectedAnswer(CP) == 1 then
    -- Mach was!
Else
    -- Mach was anderes!
end
```

Auf diese Weise kannst Du, basierend auf der gewählten Antwort, auf die Entscheidung des Spielers reagieren. Wozu Du dies nutzt bleibt deiner Fantasie überlassen.

Das Cutscene-Gerüst

Unter einer Cutscene versteht man einen effektvollen Kameraflug. Cutscenes sind *nicht* dafür gedacht, Dialoge abzubilden. Zudem stehen einige Funktionen nicht zur Verfügung, wie z.B. das Setzen von Markierungen auf der Map oder Multiple Choice. Diese sind Briefings vorbehalten. Dafür können aufwendige Bewegungsabläufe erzeugt werden.

```
local cutscene = {
    barStyle = "small",
    disableGlobalInvulnerability = false,
    restoreCamera = false,
    restoreGameSpeed = false,
    skipAll = true,
    hideFoW = true,
    showSky = true,
    hideBorderPins = true
};
local AF = AddFlights(cutscene)
```

Dies ist der Kopf einer Cutscene mit den empfohlenen Einstellungen.

Eine Cutscene verwendet ausschließlich die Funktion AF.

```
AF {
    {
        Position = {X= 12300, Y= 23000, Z= 3400},
        LookAt   = {X= 22000, Y= 34050, Z= 200},
        Text     = "Das ist ein Text....",
    },
    {
        Position = {X= 12300, Y= 23000, Z= 3400},
        LookAt   = {X= 22500, Y= 31050, Z= 350},
        Text     = "Das ist ein Text....",
    },
    FadeOut  = 0.5,
    FadeIn   = 0.5,
    Duration = 24,
};
```

Diese Funktion erspart etwas Schreibarbeit, weil sie beliebig viele Punkte innerhalb eines Flight abfahren kann. Ein Flight besteht immer aus mindestens zwei Punkten: Einem Startpunkt und einer Position, zu der sich bewegt werden soll. Jeder einzelne Punkt kann einen eigenen Text anzeigen. Die Anzeigezeit Duration wird auf alle Punkte (exklusiv dem Startpunkt) aufgeteilt

Relative Positionsangaben

Neben der Angabe von XYZ-Koordinaten können auch Entities verwendet werden. Hier ist es außerdem möglich, sich in einer gewissen Entfernung relativ im Winkel zur Entity zu bewegen.

```
-- {Entity, Kamerahöhe, Distanz, Winkel}
LookAt      = {"Blickpunkt1", 200},
Position    = {"Kameraposition1", 450, 3000, 25},
```

Erneut fehlt nur noch der Fuß.

```
cutscene.finished = function()
end
return StartCutscene(cutscene);
```

Im Fuß wird eine optionale Funktion deklariert, die am Ende der Cutscene ausgeführt wird. Wichtiger ist die letzte Zeile der Funktion. StartCutscene führt die Cutscene aus und gibt wieder die ID zurück. Diese ID wird für die Anbindung an das Questsystem benötigt.

Interaktive Objekte

Mit den erweiterten interaktiven Objekten kannst Du neue Elemente in Deine Maps bringen. Helden können z.B. versteckte Schalter finden um geheime Gänge zu öffnen. Der große Vorteil ist, dass sich so ziemlich jedes Entity in ein interaktives Objekt umwandeln lässt. Zum Beispiel könnte einem Schloss der richtige Schlüssel fehlen.

```
CreateObject {
    Name                = "trigger1",
    Title               = "Versteckter Schalter",
    ConditionUnfulfilled = "Dir fehlt der benötigte Schlüssel!",
    Condition           = function(_Data)
        return HasFoundKey == true;
    end,
    Callback            = function(_Data)
    end,
};
```

Diese Konfiguration ist ein Beispiel für ein versteckten Schalter, der aber erst aktiviert werden kann, wenn die Variable `HasFoundKey` gesetzt ist. Vorher bekommt der Spieler eine Meldung, dass er einen Schlüssel benötigt.

Es gibt natürlich noch mehr Eigenschaften, als die, die in dem Beispiel gezeigt werden. Für mehr Informationen verweise ich auf die Dokumentation.

Nichtspieler-Charaktere

Nichtspieler-Charaktere (NPC) sind Siedler, die von einem Helden explizit angesprochen werden müssen, damit sie etwas auslösen. Du kannst sie Dir als lebende interaktive Objekte vorstellen. Ein NPC ist dadurch zu erkennen, dass er blinkt.

```
Local NPC = NonPlayerCharacter:New("settler")
                 :SetCallback(VeryImportantBriefing)
                 :Activate();
```

Dies ist ein Beispiel für einen einfachen NPC.

Du kannst Deine NPCs auch an Quests anbinden, anstelle sie entkoppelt zu verwenden, indem Du die entsprechenden Behavior verwendest.