

# Orthos - Anleitung

Grundlagen zur Verwendung des Assistenten

Version 2

by totalwarANGEL

## Inhaltsverzeichnis

1	Einleitung.....	4
1.1	Vorwort.....	4
1.2	Rechtsinformationen.....	4
2	Installation.....	5
2.1	Installation von Java.....	5
2.2	Installation des Programms.....	5
2.3	Hinweise zur Nutzung.....	5
3	Konzepte.....	6
3.1	Briefing.....	6
3.2	Quest.....	6
3.3	Kartenskript.....	7
4	Oberfläche.....	8
4.1	Startbildschirm.....	8
	Projekt anlegen.....	8
	Projekt laden.....	9
4.2	Projekt-Tab.....	9
	Projekteigenschaften.....	9
	Import.....	10
	Map-Datei.....	11
	Export.....	11
4.3	Karteneinstellungen-Tab.....	11
	Kartenbeschreibung.....	11
	Spielereinstellungen.....	11
	Diplomatie.....	12
	Rohstoffe.....	13
	Wetterset.....	13
	DEBUG.....	13
4.4	Briefingassistent-Tab.....	14
	Briefings.....	14
	Pages.....	14
4.5	Questassistent-Tab.....	15
	Quests.....	15
	Behavior.....	16
5	DEBUG nutzen.....	18
	Cheats.....	18
	Eingabeaufforderung.....	19
6	Beispiele: Questassistent.....	20
	Mein erster Quest.....	20
	KI-gesteuerte Armeen.....	21
	Custom Variables.....	21
	Multiple Choice.....	22
	Umgang mit Fragmenten.....	22
7	Beispiele: Scripting.....	23
	Skripte auslagern.....	23
	MapScriptFunction-Behavior.....	23
	Goals.....	24
	Trigger.....	24

Rewards und Reprisals.....	24
8 Erweiterungen schreiben.....	25
8.1 Einleitung.....	25
8.2 Behavior-Definition.....	25
8.3 Behavior programmieren.....	27
Reset.....	28
Debug.....	28
CustomFunction.....	28
AddParameter.....	28
Get-Goal/Reward/Reprisal/Trigger-Table.....	28
Das Type-Attribut.....	28

# 1 Einleitung

## 1.1 Vorwort

Der Einstieg in das Mappen ist für Neulinge und jene, die dem Programmieren nicht mächtig sind, schon immer schwer gefallen. Anders als „Aufstieg eines Königreich“ bietet „Das Erbe der Könige“ kein Questsystem und keinen grafischen Editor. Der Mapper muss alles selbst programmieren (können).

Eine Hürde, vor der viele zurückschrecken. Mit dem Beispiel von Siedler 6 vor Augen, habe ich Questsystem und Questassistent – soweit es möglich war – nach Siedler 5 portiert. Dadurch kannst Du nun die Handlung der Map „zusammenklicken“, anstelle alles Programmieren zu müssen. Es ist natürlich dennoch möglich eigene Funktionen zu schreiben! Meine Intention war es, dem Mapper das Leben zu erleichtern. Wenn Du Dich darauf einlassen willst, dann gibt der Sache eine Chance. Das folgende Dokument soll Dir beim Einstieg in diese neue Welt behilflich sein.

Für die neue Version wurden viele Dinge angefasst und verbessert um die Bedienung zu erleichtern. Natürlich sind auch einige neue Features hinzugekommen. Lasse dich also überraschen!

## 1.2 Rechtsinformationen

"DIE SIEDLER" und "DIE SIEDLER – Das Erbe der Könige" sind eingetragene Marken von Ubisoft Entertainment. Alle Rechte diesbezüglich liegen bei Ubisoft Entertainment.

Das in dieser Anleitung beschriebene Programm ist eine Anwendung zur Erzeugung von Missionen für das Spiel DIE SIEDLER – Das Erbe der Könige“(folgend als Map bezeichnet), welches ein Tool nutzt um Maps zu öffnen, zu verändern und zu speichern. Dieses Tool wird folgend als „BBA Tool“ bezeichnet.

Der Autor der Anwendung und des Manual war und ist nicht an der Entwicklung des BBA Tool beteiligt. Verantwortliche diesbezüglich sind bobby und yoq. Siehe <http://settlers.pro> für mehr Informationen.

## 2 Installation

### 2.1 Installation von Java

Um die Anwendung nutzen zu können wird ein Java Runtime Environment (JRE) benötigt. Ich habe den Assistenten für JRE 8 entwickelt. Alle gängigen Distributionen der JRE ab Version 8 sollten funktionieren.

Beispiele für Runtime Environments:

- AdoptOpenJDK (Enthält JRE)
- Open JDK (Enthält JRE)
- Oracle JRE

Wer noch kein Java auf seinem PC installiert hat, kann sich z.B. die AdoptOpenJDK von dieser Seite herunterladen.

<https://adoptopenjdk.net/installation.html?variant=openjdk12&jvmVariant=hotspot>

Du kannst auswählen, welche Software heruntergeladen wird. Wähle als Plattform Windows x64 JRE aus. Solltest Du noch immer ein 32-Bit-System haben, musst Du stattdessen Windows x86 JRE auswählen.

Folge anschließend den Anweisungen auf der Seite.

### 2.2 Installation des Programms

Sofern Java korrekt installiert ist, kannst Du das Archiv mit der Anwendung in ein beliebiges Verzeichnis kopieren. Das Programm wird durch Doppelklick auf die Executable (\*.exe) gestartet. Du kannst dir auch eine Verknüpfung auf den Desktop oder in das Startmenü anlegen, welche auf die Executable zeigt.

### 2.3 Hinweise zur Nutzung

Die Nutzung der Anwendung erfolgt auf eigene Gefahr. Ich übernehme nicht die Verantwortung, sollte Deine Map Schaden nehmen. Erstelle also immer eine Sicherheitskopie und arbeite nur mit einer Kopie der Map!

Wenn ein Projekt geöffnet wird und wenn ein Projekt zu einer Map exportiert wird, wird das (mitgelieferte) BBA Tool genutzt. Da das Programm nicht mit Administratorprivilegien gestartet wird, kann es nicht in schreibgeschützte Verzeichnisse schreiben. Ist ein Verzeichnis geschützt, kann es zum "einfrieren" kommen.

Beim ersten Start der Anwendung wird in Deinem Nutzerverzeichnis ein Workspace angelegt. Der Workspace liegt für gewöhnlich in diesem Verzeichnis:

`C:\Users\<USERNAME>\MapMaker\workspace`

In diesem Verzeichnis werden Projekte gespeichert. Du kannst natürlich auch in anderen Verzeichnissen speichern, wenn Du das willst. Bei der Auswahl eines Projektes schaut die Anwendung jedoch immer zuerst in diesen Pfad. Die Map wird genutzt um u.a. Skriptnamen auszulesen.

## 3 Konzepte

### 3.1 Briefing

Briefings sind der Hauptkommunikationsweg in DIE SIEDLER – Das Erbe der Könige. Sämtliche Dialoge werden in solchen Briefings abgehandelt. Es ist zwar auch möglich, einfache Nachrichten auf dem Bildschirm auszugeben, doch Briefings bieten Dir mehr Möglichkeiten.

Ein Briefing besteht immer aus mindestens einer Seite. Seiten zeigen Text an und lassen die Kamera zu einem bestimmten Punkt springen. Animationen sind auch möglich, wenn ein Briefing im Skript definiert wird.

Es gibt 4 Arten von Seiten:

- Dialog: Eine Seite, die hauptsächlich Text anzeigen soll.
- Choice: Der Spieler wählt zwischen i.d.R. zwei Optionen.
- Separator: Trennt Dialogpfade nach einer Entscheidung ab.
- Redirect: Ermöglicht springen zu einer anderen Seite.

Nach Abschluss eines Briefings können getroffene Entscheidungen über ein spezielles Behavior in Quests abgefragt werden.

### 3.2 Quest

Ein Quest ist ein Job, der sich nach den angegebenen Behavior (Verhalten) abläuft. Quests können im Hintergrund arbeiten und zur Steuerung der Mission genutzt werden. Sie können jedoch auch sichtbar sein, d.h. im Auftragsbuch als Auftrag angezeigt werden. Dann tragen sie sich selbständig ein. Es kann aber nur maximal 8 sichtbare Quests geben!

Das Verhalten eines Quests wird durch Behavior gesteuert. Es ist auch möglich einen Quest als *Fragment* eines anderen zu definieren. Fragmente tragen sich selbst nicht ins Auftragsbuch ein, sondern werden vom übergeordneten Quest automatisch der eigenen Beschreibung hinzugefügt.

Es gibt 4 Arten von Behavior:

- Goal: Ein Ziel, das vom Spieler erfüllt werden muss. Ein Ziel kann erfolgreich sein, fehlschlagen oder noch unerfüllt sein.
- Reprisal: Eine Aktion, die auf einen Fehlschlag folgt.
- Reward: Eine Aktion, die auf erfolgreichen Abschluss folgt.
- Trigger: Der Auslöser, welcher einen Quest aktiviert.

Quests können verschiedene Status einnehmen. Je nach Status reagiert das Spiel.

Es sind 3 Status möglich:

- inaktiv: Der Quest wartet darauf ausgelöst zu werden
- aktiv: Der Quest ist aktiv und Ziele werden geprüft
- beendet: Der Quest ist abgeschlossen, fehlgeschlagen oder abgebrochen

Außerdem besitzt jeder Quest einen Resultatstypen. Dieser Resultatstyp gibt an, wie ein Quest beendet wurde.

Mögliche Resultatstypen sind:

- unentschieden: Der Quest wurde noch nicht beendet
- abgebrochen: Der Quest wurde ergebnislos beendet
- erfolgreich: Der Quest wurde erfolgreich abgeschlossen
- fehlgeschlagen: Der Quest ist fehlgeschlagen

Der Lebenszyklus eines Quests kann als ewiger Kreislauf gesehen werden. Quests können immer neu gestartet werden und noch einmal ausgeführt werden. Natürlich müssen die Voraussetzungen noch zutreffen.

Als Beispiel für so einen Quest werde ich die Standard-Siegbedingung einer Map hernehmen. Dafür benötigt man i.d.R. 3 Behavior. Wir wollen, dass der Spieler gewinnt, wenn das Haupthaus "HQ2" von Spieler 2 zerstört wird.

```
Goal_DestroyPlayer(2, "HQ2")  
Reward_Victory()  
Trigger_AlwaysActive()
```

Auf diese Weise können beliebige Quests zusammen gesetzt werden. Tiefgründigere Beispiele folgen später im Dokument.

### 3.3 Kartenskript

Das Kartenskript hat eigentlich nur noch die Aufgabe, die vom Assistenten genutzte Bibliothek *Orthos* zu laden und das interne Skript zu starten. Das interne Skript ist komplett maschinell nach Deinen Angaben erzeugt.

Du kannst das Kartenskript jedoch auch dafür nutzen, um eigene Funktionen zu schreiben. Ein Beispiel dafür sind von Dir selbst geschriebene Briefings, die Du nicht im Assistenten zusammen gestellt hast. Das kann nötig sein, wenn Du mehr Funktion brauchst, als der Assistent bereit stellt. Auch Includes aus dem Projekt und Custom Behavior werden hier geladen.

Aber dazu später mehr.

## 4 Oberfläche

Die Oberfläche des Assistenten kann in ihrer Größe verändert werden. Sie ist dafür ausgelegt, gleichzeitig mit dem Mapeditor benutzt zu werden. Du kannst die Größe nach Deinen Wünschen verändern und hast auf modernen Monitoren noch genug Fläche für den Editor übrig.

### 4.1 Startbildschirm

#### Projekt anlegen

Im Startbildschirm können Projekte erstellt werden. Jedes Projekt braucht einen Namen. Der Name darf nur aus Buchstaben, Zahlen und Unterstrichen bestehen. Aus dem Namen des Projektes wird der Name des Verzeichnis abgeleitet.

**Projekt anlegen**

Wähle ein Verzeichnis aus und erstelle ein neues Projekt. Dein Projekt benötigt einen Namen. Im Verzeichnis darf sich kein Ordner mit dem gleichen Namen befinden.

**Verzeichnis**

C:\Users\angermanager\MapMaker\workspace

Durchsuchen

**Projektname**

**Projektbeschreibung**

**Kartenarchiv auswählen**

Durchsuchen

**Legacy-Project importieren**

Durchsuchen

Project erstellen

Außerdem benötigst Du eine Map-Datei (Kartenarchiv) für dein Projekt. Während Du am Projekt arbeitest um Skriptnamen auszulesen, später um Deine Mission in eine spielbare Map umzuwandeln.

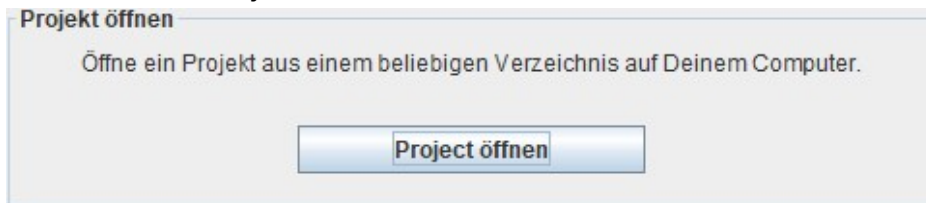


Du kannst außerdem noch eine S5N-Datei, die Projektdatei der alten Version des Assistenten, in Dein Projekt importieren. Somit bist Du nicht gezwungen von Null anzufangen, wenn Du bereits ein Projekt mit der alten Version begonnen hast.

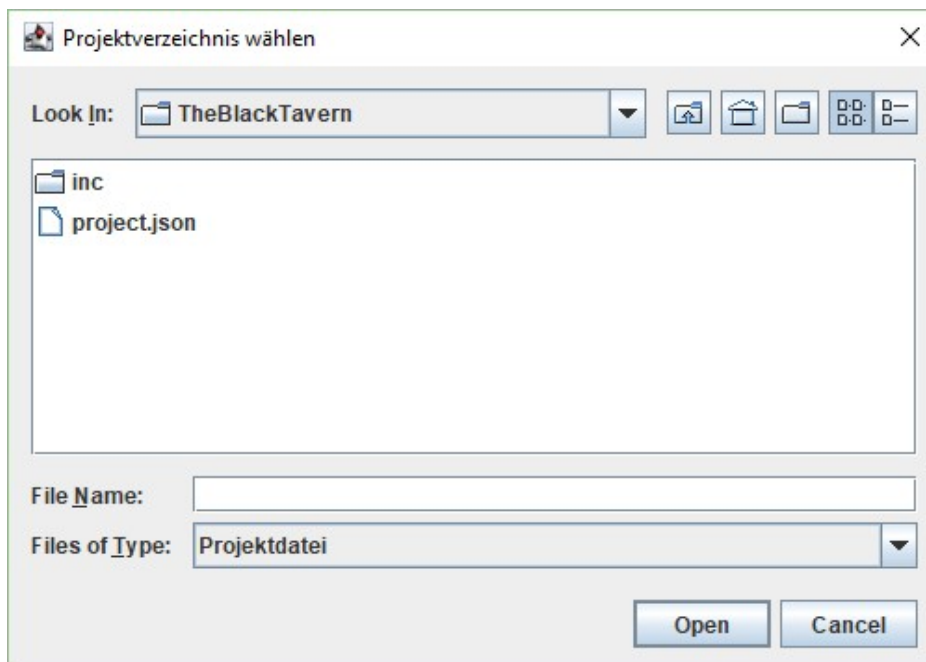
Wenn Du zufrieden bist, kannst Du das Projekt erstellen. Es wird dann im ausgewählten Arbeitsverzeichnis angelegt. Standardmäßig ist das der Workspace, der vom Programm im Benutzerverzeichnis angelegt wird.

## Projekt laden

Um an einem Projekt weiterzuarbeiten musst Du es zuerst laden.



Die Anwendung wird zuerst im Standardarbeitsverzeichnis suchen. Du kannst jedoch zu jedem beliebigen Ordner navigieren. Um ein Projekt zu laden, musst Du eine Datei `project.json` auswählen.



## 4.2 Projekt-Tab

### Projekteigenschaften

In den Projekteigenschaften kann die optionale Beschreibung des Projektes geändert werden. Die Beschreibung wird nicht im Spiel erscheinen. Du kannst hier also schreiben, was Du willst. Nutze die Beschreibung um verschiedene Versionen des Projektes zu unterscheiden.

Wichtiger jedoch ist, dass Du niemals vergisst Dein Projekt zu speichern! Sonst sind alle bisherigen Änderungen an der Projektdatei verloren. Schließen bewirkt, dass Du zum Startbildschirm zurück kehrst.

Du kannst Dein Projekt zum letzten Speicherstand zurücksetzen. Dann werden alle Änderungen, die Du bisher gemacht hast, verworfen.

Es kann auch nützlich sein, sich gelegentlich eine Sicherungskopie anzulegen. Dann kannst Du zu einem vorherigen Stand zurückkehren, sollte es nötig sein.

**Aktuelles Projekt**

Trage hier Titel und Beschreibung der Map ein. Leere Felder werden ignoriert.

TheBlackTavern

This is a test.  
Look, with new lines!

Hier kannst Du das Projekt speichern, schließen, klonen und zurücksetzen.

Speichern Klonen

Schließen Reset

## Import

Du kannst weitere Dateien in Dein Projekt importieren. Skripte sind später in der Map im Unterverzeichnis `script` in der Map zu finden. Grafiken und Sounds werden nach `media` kopiert. Cutscenes befinden sich direkt in der Map. Innerhalb Deines Projektes befinden sich alle Dateien im Ordner `inc`. Solltest Du Versionsverwaltung nutzen, kannst du diesen Ordner ignorieren lassen.

**Importdateien**

Füge Dateien hinzu, die in die Map importiert werden.  
Du kannst Importdateien jeder Zeit wieder entfernen.

E:/Repositories/theblacktavern/map/briefings.lua  
E:/Repositories/theblacktavern/map/comforts.lua  
E:/Repositories/theblacktavern/map/workplace.lua

Hinzufügen Entfernen Neu laden

Der Assistent erlaubt folgende Dateiformate:

- \*.lua, \*.luac – Lua-Skripte und Lua-Bytecode
- \*.mp3 – Sounddateien
- \*.png – Grafiken
- \*.xml – Konfigurationsdateien wie z.B. Cutscenes

Für jede dieser Dateitypen gibt es Anleitungen, wie sie zu nutzen sind. Daher werde ich nicht genauer darauf eingehen. Imports müssen nach einer Änderung neu importiert werden. Du kannst alle Dateien neu laden. Dabei werden auch Dateien entfernt, deren Original nicht gefunden werden kann.

## Map-Datei

Solltest Du die Map-Datei verschieben, welche im Projekt referenziert ist, kann sie nicht mehr gefunden werden. Export oder das Auslesen von Skriptnamen ist dann nicht mehr möglich. Du kannst jedoch jeder Zeit eine neue Datei hinterlegen. Bei Änderungen an der Map-Datei muss die Map neu eingelesen werden.

**Map-Datei tauschen**  
Tausche die Mapdatei aus oder lade sie neu um Skriptnamen zu aktualisieren.  
Erbe der Könige - Gold Edition/extra2/shr/maps/user/twA-TheSevenDeadlySins.s5x  
Durchsuchen Aktualisieren

## Export

Um aus dem Projekt eine spielbare Map zu machen, muss das Projekt exportiert werden. Beim Klicken auf "Exportieren" wird eine neue Map-Datei im Projektverzeichnis erzeugt. Die referenzierte Map wird nicht verändert.

**Projekt exportieren**  
Exportiere Dein Projekt unter Verwendung des BBA-Tool zu einem Kartenarchiv.  
Exportieren

## 4.3 Karteneinstellungen-Tab

Im Karteneinstellungen-Tab kannst Du verschiedene Änderungen vornehmen. Jede Gruppe hat eine eigene Aufgabe und muss per Hand aktualisiert werden.

### Kartenbeschreibung

Die Kartenbeschreibung wird später in die Map hineingeschrieben. Du kannst beide Felder leer lassen. Dann werden sie ignoriert.

**Kartenbeschreibung**  
Titel und Beschreibung der Map.  
Die schwarze Taverne  
Ari Harker antwortet auf die Heiratsanzeige des mysteriösen Lord Rakkuls und reist in den hohen Norden. Doch die Bewohner verhalten sich merkwürdig. Aris zukünftiger Gatte verkehrt an einem schaurigen Ort, um den sich viele Geschichten ranken: die schwarze Taverne. @cr Welche Geheimnisse wird Ari auf ihren Reisen aufdecken?  
Aktualisieren

## Spielereinstellungen

Die unterschiedlichen

Parteien in einer Map benötigen einen Namen, damit sie im Diplomatienmenü eingetragen werden. Wenn Du Felder frei lässt, wird dieser Spieler nicht im Diplomatienmenü angezeigt.

Du kannst außerdem eine Spielerfarbe bestimmen. Du hast die Wahl zwischen mehreren speziellen Farben und der Standardfarbe des jeweiligen Spielers. Achte aber darauf, es nicht zu bunt werden zu lassen.

**Spielereinstellungen**

Hier kannst Du Einstellungen an den Parteien vornehmen. Du kannst ihre Farbe und den Namen im Diplomatienmenü bestimmen.

Spieler 1	Name	Farbe
Spieler 1	Ari Harker	DEFAULT COLOR
Spieler 2	Lord Rakkul	NEPHILIM COLOR
Spieler 3	Isac	ROBBERS COLOR
Spieler 4	Linnea	EVIL GOVERNOR COLOR
Spieler 5	Liam	FRIENDLY COLOR1
Spieler 6	Hakon	ENEMY COLOR1
Spieler 7	Caroline	FRIENDLY COLOR2
Spieler 8		NPC COLOR

**Aktualisieren**

## Diplomatie

Im Spiel gibt es 3 verschiedene diplomatische Beziehungstypen. In der Gruppe für die Diplomatie kannst Du die Beziehungen aller Spieler zueinander bestimmen. Wähle die entsprechende Farbe der gewünschten Beziehung aus. Beziehungen können auch später über Behavior geändert werden.

**Diplomatie**

Hier kannst du die Diplomatie aller Spieler zueinander bei Spielbeginn festlegen. Die Beziehungen werden erst nach Reihe, dann nach Spalte gelesen.

Mögliche Beziehungen:  
■ Neutral ■ Feindlich ■ Verbündet

	S. 1	S. 2	S. 3	S. 4	S. 5	S. 6	S. 7	S. 8
R. 1								
R. 2								
R. 3								
R. 4								
R. 5								
R. 6								
R. 7								
R. 8								

**Aktualisieren**

## Rohstoffe

Du kannst in der Rohstoffe-Gruppe die Startrohstoffe bestimmen, die sofort zu Spielbeginn verfügbar sind. Wenn erst später Rohstoffe verfügbar sein sollen, nutze stattdessen Behavior.

**Rohstoffe**  
Hier werden die Rohstoffe zu Spielbeginn gesetzt.

<b>Taler</b>	<b>Lehm</b>	<b>Holz</b>
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
<b>Stein</b>	<b>Eisen</b>	<b>Schwefel</b>
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

Aktualisieren

## Wetterset

Wettersets sind die Pseudo-Klimazonen von Siedler. Durch Auswahl eines Sets verändert sich das Ambiente. Zusätzlich kannst Du für das Set typische Wetterelemente automatisch generieren lassen.

Während in den Highlands knackig kalt ist und viel Schnee fällt, wird für Sumpf ein warmfeuchtes Klima erzeugt. In der Steppe wird es niemals schneien und selten regnen eben so wie in Elelance. Normales Wetter verhält sich so, wie man es als Europäer erwartet. Es kann aber auch zu seltsamen Konstellationen kommen.

**Wetterset**  
Das Wetterset bestimmt die Belichtungsverhältnisse im Spiel.

Aktiviere diese Option, wenn automatische Wetterwechsel für das ausgewählte Set generiert werden sollen.

**Hinweis:** Wetterwechsel sind zufällig und können zu unvorteilhaften Konstellationen führen!

Aktualisieren

## DEBUG

Über DEBUG kannst Du hilfreiche Zusatzfunktionen wie z.B. spezielle Cheats oder Prüfung der Quests hinzuschalten.

**Debug-Optionen**  
Aktiviere die Prüfung von Quests. Bevor ein Quest getriggert wird, wird er auf Fehler geprüft und ggf. übersprungen.

Aktiviere die Statusverfolgung. Es wird eine Meldung angezeigt, wenn ein Quest seinen Status wechselt.

Aktiviere die Development Cheats um Deine Map schneller testen zu können.

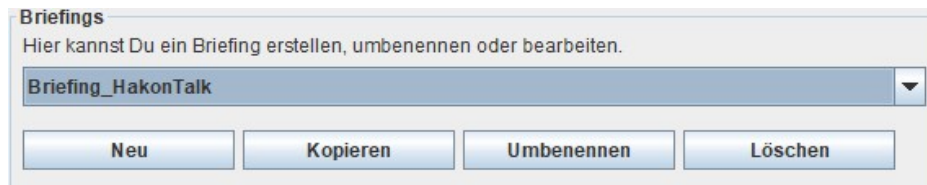
Aktiviere die Konsole um über spezielle Befehle Abläufe zu manipulieren oder nachträglich Skripte zu laden.

Aktualisieren

## 4.4 Briefingassistent-Tab

### Briefings

In der Briefings-Gruppe kannst Du Briefings auswählen, sie erstellen, sie umbenennen, sie kopieren und sie löschen. Wenn Du ein neues Briefing erstellst oder vorhandene Briefings kopierst bzw. umbenennst, musst du einen Namen angeben. Jeder Name darf nur einmal vorkommen.

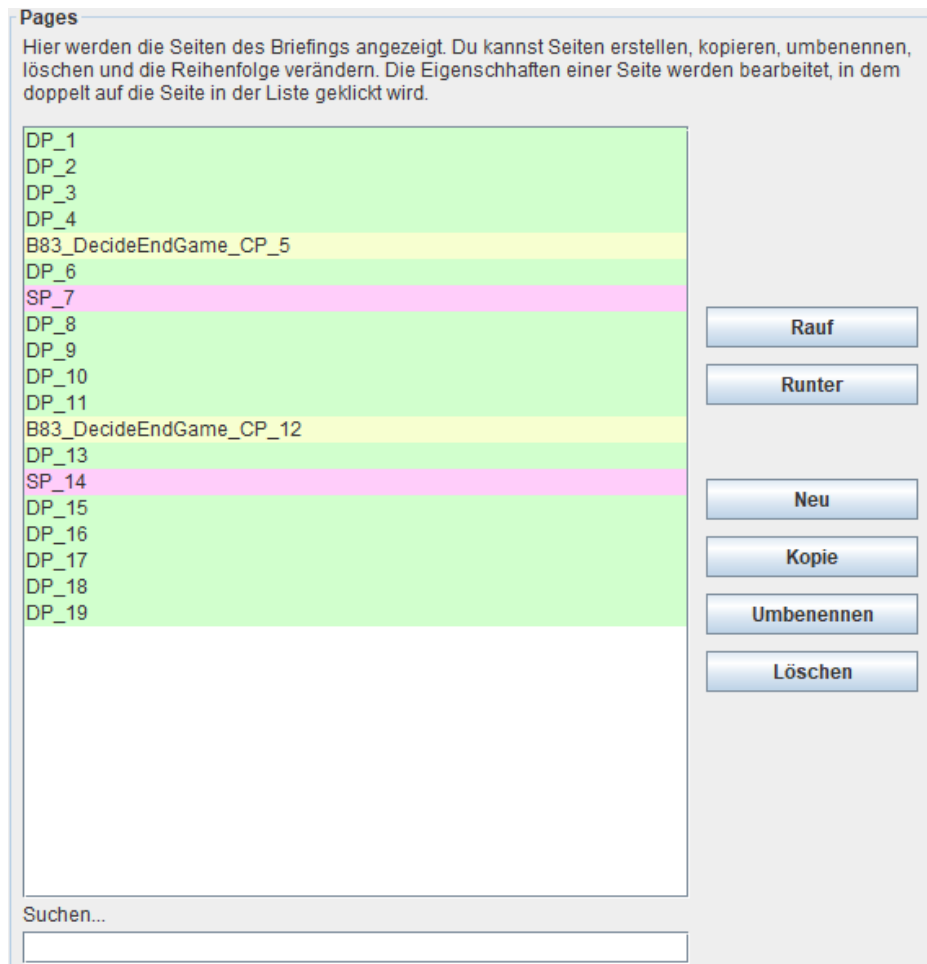


The screenshot shows a window titled "Briefings". Below the title bar, there is a text instruction: "Hier kannst Du ein Briefing erstellen, umbenennen oder bearbeiten." Below this instruction is a text input field containing "Briefing\_HakonTalk" with a dropdown arrow on the right. At the bottom of the window, there are four buttons: "Neu", "Kopieren", "Umbenennen", and "Löschen".

### Pages

Jedes Briefing besteht aus mindestens einer Seite. Diese Seiten zeigen Text an und werden genutzt um die Geschichte der Map zu erzählen oder um den Spieler über seine Mission zu informieren. Bearbeitet werden die Seiten in der Gruppe Pages.

Jeder Seitentyp hat eine Farbe zugeordnet, damit er leichter erkannt werden kann. Normale Seiten sind grün, Entscheidungen sind gelb eingefärbt und Separatoren in rosa hervorgehoben.



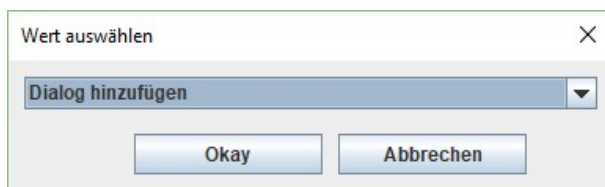
The screenshot shows a window titled "Pages". Below the title bar, there is a text instruction: "Hier werden die Seiten des Briefings angezeigt. Du kannst Seiten erstellen, kopieren, umbenennen, löschen und die Reihenfolge verändern. Die Eigenschaften einer Seite werden bearbeitet, in dem doppelt auf die Seite in der Liste geklickt wird." Below this instruction is a list of pages, each with a colored background: DP\_1 (green), DP\_2 (green), DP\_3 (green), DP\_4 (green), B83\_DecideEndGame\_CP\_5 (yellow), DP\_6 (green), SP\_7 (pink), DP\_8 (green), DP\_9 (green), DP\_10 (green), DP\_11 (green), B83\_DecideEndGame\_CP\_12 (yellow), DP\_13 (green), SP\_14 (pink), DP\_15 (green), DP\_16 (green), DP\_17 (green), DP\_18 (green), and DP\_19 (green). To the right of the list, there are five buttons: "Rauf", "Runter", "Neu", "Kopie", and "Umbenennen", followed by a "Löschen" button. At the bottom of the window, there is a search bar labeled "Suchen..." with an input field.



Es gibt 4 Arten von Seiten:

- Dialogseite: Die Kamera springt zur Position und zeigt einen Text an. Das Ziel kann entweder aus der Ferne oder aus der Nähe gezeigt werden. Die Kamera schaut das Ziel immer an.
- Entscheidungsseite: Bei einer Entscheidung wählt der Spieler zwischen einer von 2 Optionen. Die getroffene Entscheidung kann nach dem Ende des Briefings über `Goal_MultipleChoiceSelection` abgefragt werden.
- Separator: Ein Separator dient ausschließlich dazu die Zweige nach einer Entscheidung abzutrennen und das Briefing zu beenden.
- Redirect: Eine Weiterleitung wird benutzt, um zu einer anderen Stelle im Briefing zu springen.

Alle Seiten erhalten automatisch einen fortlaufenden Namen. Dieser Name muss im ganzen Briefing eindeutig sein. Entscheidungen sind ein Sonderfall. Ihr Name muss im kompletten Projekt eindeutig sein. Die Seiten werden untereinander dargestellt und können in ihrer Reihenfolge vertauscht werden. Natürlich können die Seiten eines Briefings auch kopiert, umbenannt und gelöscht werden.



Wenn Du eine neue Seite anlegst, musst Du Dich für einen Typ entscheiden. Der Typ bestimmt das Verhalten der Seite.

Wenn Du den Namen der Seite veränderst, darf dieser Name nicht schon im Briefing vorkommen. Und im Falle einer Entscheidung nirgends wo im Projekt. Du möchtest vielleicht bei der Standardbenennung verbleiben, es kann aber übersichtlicher sein zumindest Entscheidungen nach dem zu benennen, was entschieden wird.

Briefings werden später über Quests gestartet. Ein Briefing kann entweder auf einen Erfolg oder einen Fehlschlag folgen. Entsprechend können nachfolgende Quests auf ein Briefing getriggert werden.

## 4.5 Questassistent-Tab

### Quests

Quests, also Aufträge, werden benutzt um die Handlung zu steuern. Sie laufen meistens im Hintergrund ab, sodass der Spieler nichts von ihnen mitbekommt. Manche sind sichtbar und tragen sich selbst ins Auftragsbuch ein.

Damit ein Quest sichtbar wird, muss seine Sichtbarkeit geändert werden. Du kannst dann weitere Einstellungen vornehmen. Diese müssen mit dem Button „Update“ bestätigt werden um wirksam zu werden.

Quests können umbenannt, gelöscht oder kopiert werden. Kopieren bewirkt, dass sie komplett mit Beschreibung und allen Behavior geklont werden. Wird ein Quest umbenannt, werden auch automatisch alle Erwähnungen angepasst.

Aufträge werden in der Gruppe Quests bearbeitet.

**Quests**

Hier können Aufträge verwaltet werden. Du kannst Aufträge erstellen, kopieren, umbenennen und löschen. Die Eigenschaften eines Auftrages werden bearbeitet, in dem der Auftrag aus der Liste ausgewählt wird. Um Eigenschaften zu aktualisieren, klicke auf "Update".

Ch02_MainQuest	<input type="button" value="Update"/> <b>Sichtbarkeit</b> sichtbar <b>Empfänger</b> Spieler 1 <b>Zeitlimit</b> 0 <b>Typ</b> FRAGMENTQUEST_OPEN <b>Titel</b> Spricht mit dem Brückenarchitekten. <b>Beschreibung</b> Erkundigt Euch über den Fortschritt auf der Baustelle.
Ch02_Quest01_ChapterIntro	
Ch02_Quest01_FindLittleJohn01	
Ch02_Quest01_FindLittleJohn02	
Ch02_Quest01_FindLittleJohn03	
Ch02_Quest01_FindLittleJohn04	
Ch02_Quest01_FindLittleJohn_Quest	
Ch02_Quest02_BuildCastle01	
Ch02_Quest02_BuildCastle02	
Ch02_Quest02_BuildCastle03	
Ch02_Quest02_BuildCastle04	
Ch02_Quest02_BuildCastle05	
<b>Ch02_Quest02_BuildCastle06</b>	
Ch02_Quest02_BuildCastle_Quest	
Ch02_Quest03_LazyKnight01	
Ch02_Quest03_LazyKnight02	
Ch02_Quest03_LazyKnight03	
Ch02_Quest03_LazyKnight04	
Suchen...	

Suchen...

Über den Questtyp FRAGMENTQUEST\_OPEN wird ein Auftrag als Fragment deklariert. Fragmente tragen sich nicht selbst ins Auftragsbuch ein, sondern werden anderen Quests hinzugefügt, welche in ihren Goals den Status des Quest prüfen. Je nachdem wie ein Fragment-Quest endet, wird er entsprechend eingefärbt.

## Behavior

Ein Behavior wird verwendet um den Ablauf eines Quests zusammenzubauen. Du kannst Dir die Behavior als Bausteine vorstellen. Sie können beliebig kombiniert werden und ergeben jedes Mal ein anderes Bild. Jeder Quest benötigt mindestens ein Behavior vom Typ Goal und eins vom Typ Trigger. Quests können auch mehrere Behavior eines Typs haben. Der Auftrag wird erst dann abgeschlossen, wenn alle Goals ein Ergebnis liefern.

**Behavior**

In dieser Gruppe werden Behavior verwaltet. Du kannst Behavior erstellen, kopieren, und löschen. Die Eigenschaften eines Behavior werden bearbeitet, in dem das Behavior aus der Liste doppelt geklickt wird.

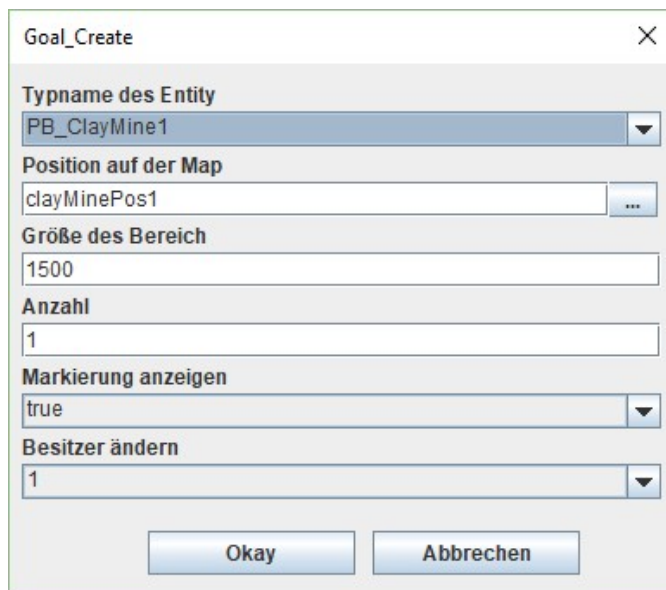
Goal_Create	<input type="button" value="Neu"/> <input type="button" value="Kopieren"/> <input type="button" value="Löschen"/>
Goal_Create	
Goal_Create	
Goal_Create	
Reprisal_ConcealArea	
Reward_Briefing	
Reward_ConcealArea	
Reward_ConcealArea	
Reward_ConcealArea	
Reward_Move	
Trigger_Briefing	



Ähnlich wie die Seiten eines Briefings, haben auch Behavior verschiedene Farben.

Wenn Du ein neues Behavior hinzufügen willst, erscheint ein Dialog. Hier kannst Du aus allen bekannten Behavior wählen. Auch Deine eigenen Custom Behavior im Projekt werden in der Liste erscheinen.

Du kannst Behavior hinzufügen, entfernen oder kopieren. Es ist aber nicht erlaubt sie umzubenennen. Behavior werden alphabetisch sortiert und in dieser Reihenfolge abgearbeitet. Wenn ein Behavior Eigenschaften hat, kannst Du sie mit Doppelklick auf das Behavior ändern.



Goal\_Create

Typname des Entity  
PB\_ClayMine1

Position auf der Map  
clayMinePos1 ...

Größe des Bereich  
1500

Anzahl  
1

Markierung anzeigen  
true

Besitzer ändern  
1

Okay Abbrechen

Es erscheint ein Dialog, in dem Du die Eigenschaften einstellen kannst. Wenn ein Feld ohne Wert ist, wirst Du eine Warnung erhalten.

Manche Felder haben einen Button ... neben sich. Über diesen Button kannst Du dir Vorschläge für das Feld anzeigen lassen. Auf diese Weise kannst Du z.B. bequem ausgelesene Skriptnamen oder Quests wählen.

## 5 DEBUG nutzen

Der DEBUG-Modus ist dafür da, Dir das Testen einer Map zu erleichtern. Du kannst Behavior eines Quests automatisch prüfen, die Developer Cheats aktivieren, den Questverlauf steuern und Statusveränderungen am Bildschirm ausgeben lassen.

### Cheats

Nachstehend folgt eine Liste der verfügbaren Cheats:

Tastenkombination	Auswirkung
CTRL + SHIFT + Num 9	Kameramodus umschalten
CTRL + Num 4	Kamerawinkel erhöhen (nur freie Kamera)
CTRL + Num 1	Kamerawinkel senken (nur freie Kamera)
CTRL + Num 5	Zoom erhöhen (nur freie Kamera)
CTRL + Num 2	Zoom verringern (nur freie Kamera)
CTRL + SHIFT + G	GUI an-/ausschalten
CTRL + SHIFT + F	Nebel des Krieges an-/ausschalten
CTRL + SHIFT + Y	Himmel an-/ausschalten
CTRL + SHIFT + 1	FPS anzeigen
CTRL + H	Entity unter der Mouse Schaden zufügen
SHIFT + H	Entity unter der Mouse heilen
SHIFT + 1 ... 8	Entity unter der Mouse dem Spieler übergeben
SHIFT + ALT + 1 ... 8	Kontrollierenden Spieler durchschalten
CTRL + ALT + 1	Bastardschwertkämpfer unter der Mouse spawnen
CTRL + ALT + 2	Arbelestenschützen unter der Mouse spawnen
CTRL + ALT + 3	Hellebariere unter der Mouse spawnen
CTRL + ALT + 4	Schwere Kavalerie unter der Mouse spawnen
CTRL + ALT + 5	Leichte Kavalerie unter der Mouse spawnen
CTRL + ALT + 6	Eisenkannone unter der Mouse spawnen
CTRL + ALT + 7	Belagerungskanone unter der Mouse spawnen
CTRL + ALT + 8	Schwere Scharfschützen unter der Mouse spawnen
CTRL + F1	+100 Taler
CTRL + F2	+100 Lehm
CTRL + F3	+100 Holz
CTRL + F4	+100 Stein
CTRL + F5	+100 Eisen
CTRL + F6	+100 Schwefel
CTRL + F7	+100 Glaube
CTRL + F8	+100 Wetterenergie

## Eingabeaufforderung

Die Eingabeaufforderung wird mit ^ (Zirkumflex) geöffnet. Du kannst durch Verkettung von Befehlen die Abfolge der Quests beeinflussen und noch ein paar andere nützliche Dinge tun. Befehle können beliebig mit & und && kombiniert werden. Dabei verkettet && Befehle und & Parameter.

Hier eine Liste der möglichen Kommandos:

Kommando	Beschreibung
help <subject>	Zeigt Hilfetexte zu den Cheats an. Nutze <code>help cheats</code> um die Kategorien anzuzeigen
load <path>	Läd das Skript vom Pfad ins Spiel (/ statt \ benutzen)
win <quest>	Gewinnt den Quest
fail <quest>	Lässt den Quest fehlschlagen
stop <quest>	Unterbricht den Quest
start <quest>	Startet den Quest
reset <quest>	Setzt den Quest zurück.
wakeup <hero>	Erweckt einen Helden (durch ReplaceEntity)
diplomacy <p1> <p2> <state>	Ändert die Diplomatie zwischen den Parteien
clear	Entfernt alle Nachrichten

## 6 Beispiele: Questassistent

Ich möchte Dir noch ein paar Beispiele auf den Weg mitgeben, wie Du das System nutzen kannst um eine interessante Map aufzubauen. Es folgen ein paar Beispiele um die Funktion zu demonstrieren.

### Mein erster Quest

Als ersten einfachen Auftrag nehmen wir an, dass der Spieler ein Gebäude eines Verbündeten für eine bestimmte Zeit schützen muss. Außerdem soll er Verteidigungsanlagen bauen. Um dies zu realisieren werden wir einige Behavior benötigen.

1. Goal\_Protect: Der Spieler muss ein Entity beschützen. Wenn das Entity zerstört wird (oder der Held bewusstlos ist), scheitert das Ziel.
2. Goal\_Create: Der Spieler muss einen Entitytyp erzeugen.
3. Reprisal\_Defeat: Der Spieler verliert bei einem Fehlschlag.
4. Trigger\_AlwaysActive: Der Auftrag wird sofort aktiviert.

Die Behavior-Konfiguration könnte dann z.B. so aussehen:

#### **Quest\_GuardVillageCenter**

```
Goal_Protect("villageCenter1"),  
Goal_Create("PB_Tower2", "defPos1", 1000, 3, true, 1),  
Reprisal_Defeat(),  
Trigger_AlwaysActive()
```

Zusätzlich musst Du nun eine Zeit einstellen, bis zu der alle Aufgaben erledigt sein müssen. Wenn der Spieler dann zwar das Dorfzentrum beschützt hat aber die Türme nicht gebaut hat, verliert er trotzdem.

Das ist dumm sagst Du? Ist es auch. Du solltest lieber zwei getrennte Quests nutzen. Wenn das Dorfzentrum erfolgreich verteidigt wurde, müssen keine Türme mehr gebaut werden. Deine Quests würden dann wie folgt aussehen:

#### **Quest\_BuildTowers**

```
Goal_Create("PB_Tower2", "defPos1", 1000, 3, true, 1),  
Trigger_AlwaysActive()
```

#### **Quest\_GuardVillageCenter**

```
Goal_Protect("villageCenter1"),  
Reprisal_Defeat(),  
Reward_QuestInterrupt("Quest_BuildTowers")  
Trigger_AlwaysActive()
```

Der ursprüngliche Quest wird um einen Abbruchbefehl ergänzt, der den Quest für den Bau der Türme abbricht, sobald er erfolgreich abgeschlossen wurde. Die Aufgabe die Türme zu bauen wird hingegen ausgelagert. Quests können mit anderen Quests verbunden werden oder andere Quests beeinflussen.

## KI-gesteuerte Armeen

Die Erzeugung von Armeen und deren Kontrolle wurde immens vereinfacht. Es gibt einen eingebauten Controller, der alle Armeen der KI bewegen kann. Alles was Du noch tun musst, ist ihr Verhalten durch Patrouillien und Angriffsziele zu beeinflussen. Ließ dir dafür die Beschreibungen der entsprechenden Behavior durch.

Willst Du nun eine Armee erzeugen, musst Du zuerst einen KI-Spieler erzeugen.

```
Reward_AI_CreateAIPlayer(2, 4, 8, true, true)
```

Hier z.B. wird Spieler 2 erzeugt und sein Technologie-Level auf Maximum gesetzt.

Nun kannst du eine Armee erzeugen.

```
Reward_AI_CreateArmy("ArmyOne", 2, 8, "base", 3000, "City")
```

Hier wird eine Armee erzeugt, die zufällige Einheiten rekrutiert. „City“ bedeutet, dass diese Einheiten normale Soldaten sind. Um Soldaten zu rekrutieren, benötigt die KI Militärgebäude in der Nähe ihrer Basis. Es gibt auch Armeen, die ihre Einheiten spawnen lassen, solange ihr Lebensfaden noch existiert.

Diese Armeen werden dann über ihre Angriffsziele und Patrouillien gesteuert. Zusätzlich gibt es noch Behavior, die den Armeen Angriff und Patrouillieren explizit zu verbieten. Dann werden diese Punkte ignoriert.

## Custom Variables

Custom Variables sind spezielle Behavior, die Dich eine Variable definieren lassen, die Du dann abfragen kannst. Diese Variablen müssen immer numerisch sein. Dafür kannst Du jedoch unbegrenzt viele von ihnen erzeugen.

Eine Custom Variable muss zuallererst mit einem Reward oder einen Reprisal initialisiert werden.

```
Reward_CustomVariable("MyVariable", "=", 0)
```

Danach können Variablen in darauffolgenden Quests beliebig abgefragt und verändert werden.

```
Goal_CustomVariable("MyVariable", "==", 3)
```

```
Trigger_CustomVariable("MyVariable", "<=", 5)
```

Mit Hilfe dieser Variablen kannst Du verschiedene Bedingungen zusammen stellen. Zum Beispiel den Spieler Ruinen plündern lassen.

```
Reward_CustomVariable("RuinsLooted", "+", 1)
```

```
Goal_CustomVariable("RuinsLooted", "==", 20)
```

Hier würde der Auftrag abgeschlossen, wenn 20 Ruinen geplündert sind.

Die Custom Variables ersparen Dir in vielen Fällen das Schreiben eines MSF-Behavior im Kartenscript oder eines Custom Behavior im Projekt.

## Multiple Choice

Multiple Choice ist eine Möglichkeit die Handlung einer Map in verschiedene Pfade aufzuteilen oder dem Spieler für richtige Entscheidungen zu belohnen. Die Möglichkeiten sind unbegrenzt. Über das Behavior `Goal_MultipleChoiceSelection` kann eine Entscheidung in einem Briefing überprüft werden. Dabei musst Du angeben, welche von zwei Optionen gewählt werden soll.

## Umgang mit Fragmenten

Einen Quest als Fragment eines anderen zu definieren, erspart Dir einiges an Schreibarbeit. Für gewöhnlich trägt sich ein Quest in das Auftragsbuch ein, wenn er startet und wieder aus, sobald er beendet ist. Willst Du nun mehrere Stages für einen Auftrages haben, so musst du den Beschreibungstext mehrfach schreiben. Mit Fragmenten musst Du dies nicht mehr tun.

Ein Quest fügt die Beschreibungen aller Fragmente seiner eigenen hinzu, wenn er einen Fragment-Quest über z.B. `Goal_WinQuest` abfragt. Dann wird die Beschreibung des Fragmentes in den Auftrag integriert, sobald das Fragment aktiv oder abgeschlossen ist. Ein aktives Fragment wird mit der vollständigen Beschreibung angezeigt (Titel + Beschreibung). Abgeschlossene Fragmente zeigen nur den Titel und werden nach ihrem Resultat eingefärbt.

- Rot: Fragment-Quest ist fehlgeschlagen.
- Grün: Fragment-Quest wurde erfolgreich abgeschlossen
- Grau: Fragment-Quest wurde abgebrochen

Auf diese Weise kannst Du einen Quest als übergeordneten Auftrag (Master) verwenden und den Status seiner Untergebenen (Slaves) prüfen. Die Beschreibung wird sich immer automatisch anpassen.

## 7 Beispiele: Scripting

Früher oder später wirst Du in die Situation kommen, dass die Möglichkeiten des Assistenten den Anforderungen nicht mehr genügen. Wenn Du an diesem Punkt angelangt bist, ist es Zeit sich mit dem Schreiben von Skripten zu befassen. Der Assistent macht nichts weiter, als ein Lua-Skript zu generieren. Die zugrunde liegende Bibliothek funktioniert vollkommen autonom vom Assistenten.

Funktionen werden im Kartenskript `mapscript.lua` geschrieben. Du kannst so eigene Behavior und Briefings schreiben oder vollkommen andere Dinge umsetzen.

### Skripte auslagern

Damit Du nicht jedes mal das Spiel beenden musst, damit eine Änderung im Skript wirksam wird, kannst du das Skript „auslagern“. Das bedeutet, dass das Kartenskript nur aus einem Ladebefehl besteht, der die eigentliche LUA-Datei lädt.

```
Script.Load("Pfad/zum/Skript.lua")
```

Je mehr Du im Skript selbst schreibst, desto mehr wirst Du testen müssen.

### MapScriptFunction-Behavior

Manchmal Reichen die Standardbehavior nicht aus. In diesem Fall wirst Du in die Situation kommen, ein Behavior selbst schreiben zu müssen. Das ist überhaupt nicht schwer, wenn man sich dabei an ein paar Regeln hält.

1. Behavior haben 4 verschiedene Typen: Goal (Ziel), Reprisal (Strafe), Reward (Belohnung), Trigger (Auslöser). Ebenso unterschiedlich wie sie sich verhalten, werden sie auch im Skript geschrieben. Wobei Reprisal und Reward identisch aufgebaut sind.
2. Ein Goal kann 3 Zustände haben: unbestimmt, erfüllt und fehlgeschlagen. Diese Zustände werden durch einen Wahrheitswert bestimmt, den das Goal zurückgeben muss. Dabei wird für erfüllt `true` zurückgegeben, für fehlgeschlagen `false` und für unbestimmt nichts.
3. Ein Trigger hat 2 Zustände: ausgelöst und inaktiv. Diese Zustände werden ebenfalls durch Wahrheitswerte repräsentiert und müssen zurückgegeben werden. Um auszulösen muss `true` zurückgegeben werden. Für inaktiv entweder `false` oder einfach nichts.
4. Ein Reprisal oder ein Reward sind simple Callbacks<sup>1</sup> und geben nichts zurück. Alle Statements<sup>2</sup> werden nacheinander ausgeführt ohne Fragen zu stellen.

<sup>1</sup> Callback: Funktion, die als Reaktion auf ein Ereignis aufgerufen wird.

<sup>2</sup> Statement: Ein Befehl im Code

## Goals

Goals sind Ziele, die der Spieler erreichen muss.

Ob ein Ziel erreicht ist, wird durch eine Fallunterscheidung<sup>3</sup> überprüft und durch Rückgabe eines Wahrheitswertes dem Questsystem bekannt gemacht.

```
function HausGebaut()  
    if Logic.IsConstructionComplete(HAUS_ID) == 1 then  
        return true;  
    end  
end
```

Selbstgeschriebene Goals werden mit Goal\_MapScriptFunction eingebunden.

Goal\_MapScriptFunction("HausGebaut")

## Trigger

Ein Auslöser bestimmt, wenn ein Quest startet.

Auch hier wird wieder eine Fallunterscheidung durchgeführt. Damit ein Quest startet,

```
function SollQuestGestartetWerden()  
    if Logic.GetTime() > 5000 then  
        return true;  
    end  
end
```

muss der Trigger false zurückgeben.

Selbstgeschriebene Trigger werden mit Trigger\_MapScriptFunction eingebunden.

Trigger\_MapScriptFunction("SollQuestGestartetWerden")

## Rewards und Reprisals

Ein Reprisal bzw. ein Reward sind Aktionen, die für Fehlschlag bzw. Erfolg ausgeführt werden. Hier müssen i.d.R. keine Fälle unterschieden werden.

```
function DarioBewegen()  
    local x, y, z = Logic.EntityGetPos(GetID("destination"));  
    Logic.MoveSettler(GetID("dario"), x, y);  
end
```

Selbstgeschriebene Rewards und Reprisals werden auf ähnliche Weise in einen Quest eingebunden.

Für einen Reward:

Reward\_MapScriptFunction("DarioBewegen")

Für ein Reprisal:

ReprisalMapScriptFunktion("DarioBewegen")

3 Fallunterscheidung: Es wird, je nach Situation, entsprechend reagiert. Beispiel: Für Fall 1 wird Aktion A ausgeführt, für Fall 2 Aktion B. Fallunterscheidungen sind die wichtigsten Werkzeuge der Programmierung.



## 8 Erweiterungen schreiben

### 8.1 Einleitung

Mit den Grundlagen zum Schreiben eines Behavior bist Du nun auch im Stande von Dir oft verwendete MapScriptFunction-Behavior zu integrieren. Von Dir integrierte Behavior befinden sich innerhalb des Projektes, in dem Du sie nutzen möchtest. Die entsprechenden Dateien kannst Du in weitere Projekte kopieren. Diese Behavior werden, wenn vorhanden, automatisch genutzt.

Du benötigst zwei Dateien:

- behaviors.json  
Beinhaltet die Definition der Behavior für den Assistenten.
- behaviors.lua  
Enthält den von Dir geschriebenen Lua-Code, der im Spiel ausgeführt wird.

### 8.2 Behavior-Definition

Damit der Assistent versteht, was Du im sagen möchtest, musst Du seine Sprache sprechen. Dazu ist es notwendig, die Syntax von JSON zu kennen. Das ist eine beschreibende Sprache, ähnlich zu XML. Jedes Behavior wird als JSON-Objekt geschrieben. Ein Behavior kann z.B. so aussehen:

```
"Goal_Example": {  
  "Description": "Das ist ein Beispiel-Behavior",  
  "Parameter": [  
    ["NumberSelect", "Zahl", ["-12", "580", "9999"],]  
  ]  
}
```

Das Objekt wird mit dem Namen des Behavior eingeleitet. Danach folgt die Beschreibung und die Liste der Parameter. Parameter bestehen immer aus Typ, Namen und einer Liste von Auswahlmöglichkeiten. Parametertypen ohne Auswahlmöglichkeit haben eine leere Liste.

Folgende Parametertypen sind möglich:

Parametertyp	Beschreibung
MessageInput	Ein Nachrichtenfenster zur Eingabe von langen Texten. Es wird für Nachrichten benutzt und löst keine Warnung aus, wenn es leer ist.
NumberInput	Ein Feld zur Eingabe einer positiven ganzen Zahl.
StringInput	Ein Feld zur Eingabe eines kurzen Strings.
BooleanSelect	Ein Dropdown für einen Wahrheitswert (true/false)
NumberSelect	Ein Dropdown zur Auswahl von vordefinierten Zahlen.
StringSelect	Ein Dropdown zur Auswahl von vordefinierten Strings.
BriefingName	Ein Textfeld zur Eingabe eines Briefing-Namens, Mit einem Button kann eine Liste aller Briefings angezeigt und ein Name ausgewählt werden. Der gewählte Name wird in das Textfeld kopiert.

ArmyName	Ein Textfeld zur Auswahl einer im Assistenten konfigurierten Armee. Über den nebenstehenden Button werden alle verfügbaren Armeen angezeigt und eine kann gewählt werden.
ChoiceName	Ein Textfeld zur Auswahl einer Entscheidung (Multiple Choice). Über den nebenstehenden Button werden alle verfügbaren Entscheidungen angezeigt und eine kann gewählt werden.
QuestName	Ein Textfeld zur Auswahl eines Quest-Name. Über den nebenstehenden Button werden alle verfügbaren Quests angezeigt und einer kann gewählt werden.
ScriptName	Ein Textfeld zur Auswahl eines Skriptnamen. Skriptnamen werden automatisch aus der Map geladen und können über den nebenstehenden Button ausgewählt und eingefügt werden.
ValueName	Ein Textfeld zur Auswahl einer Custom Variable. Über den nebenstehenden Button werden alle verfügbaren Entscheidungen angezeigt und einer kann gewählt werden.

Bedenke, dass auch bei Zahlen die vorgegebenen Werte stets als String zu schreiben sind. Alle Parameter müssen Zeichenketten sein!

## 8.3 Behavior programmieren

Um ein Behavior im Spiel verwenden zu können, musst Du auch eine neue Klasse in Lua schreiben. Eine solche Klasse besteht aus den Methoden AddParameter, CustomFunction, Debug und Reset.

Je nach Art des Behavior gibt es entsprechende Funktionen GetGoalTable, GetTriggerTable, GetRewardTable oder GetReprisalTable. Außerdem hat jede Klasse auch einen Konstruktor, welcher genauso heißt wie das Behavior. Die Klasse wird als globale Variable gespeichert. Konvention hier ist, dass auch der Name verwendet wird, allerdings mit b\_ vorangestellt.

Ein Beispiel:

```
function Goal_MyBehavior(...)
    return b_Goal_MyBehavior:New(unpack(arg));
end

b_Goal_MyBehavior = {
    Data = {
        Name = "Goal_MyBehavior",
        Type = Objectives.MapScriptFunction
    },
};

function b_Goal_MyBehavior:AddParameter(_Index, _Parameter)
    if _Index == 1 then
        self.Data.Number = _Parameter;
    end
end

function b_Goal_MyBehavior:GetGoalTable()
    return {self.Data.Type, {self.CustomFunction, self}};
end

function b_Goal_MyBehavior:CustomFunction(_Quest)
    -- Aktion
end
```

```
function b_Goal_MyBehavior:Debug(_Quest)
    if type(self.Data.Number) != "number" or self.Data.Number < 100 then
        dbg(_Quest, self, "Number is to low!");
        return true;
    end
    return false;
end

function b_Goal_MyBehavior:Reset(_Quest)
end

RegisterBehavior(b_Goal_MyBehavior);
```

## Reset

Mit der Reset-Methode können im Behavior gespeicherte Daten zurückgesetzt werden. Das ist hilfreich, wenn Du Daten in der Instanz veränderst.

## Debug

Mit der Debug-Methode kannst Du bei aktiven Debug-Mode verhindern, dass ein Quest gestartet wird, wenn im Behavior ein Fehler vorliegt. Es ist auch hilfreich für Dich, eine Fehlermeldung auszugeben, damit Du weißt, was passiert ist.

## CustomFunction

Die CustomFunction-Methode führt den eigentlichen Code aus. Hier gelten für die verschiedenen Typen von Behavior die gleichen Regeln wie bei den zuvor behandelten MapScriptFunction-Behavior.

## AddParameter

Die AddParameter-Methode wird genutzt um die Parameter in der Instanz zu speichern. Du kannst beliebige Namen für die Parameter verwenden, achte jedoch darauf nichts zu überschreiben, was schon in Data steht.

## Get-Goal/Reward/Reprisal/Trigger-Table

Diese Funktionen sind immer gleich aufgebaut und müssen auch nicht verändert werden. Es darf nur die jeweils entsprechende Methode vorhanden sein.

## Das Type-Attribut

Mit dem Typ-Attribut wird der Typ des Behavior bestimmt.

Schreibe für ein Goal `Objective.MapScriptFunction`, für einen Trigger `Triggers.MapScriptFunction` und für Rewards und Reprisals gleichermaßen `Callbacks.MapScriptFunction`.