

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.....	7
4.1	Startbildschirm.....	7
	Projekt anlegen.....	7
	Projekt laden.....	8
4.2	Projekt-Tab.....	9
	Projekteigenschaften.....	9
	Map-Datei.....	10
	Export.....	10
	Import.....	10
4.3	Karteneinstellungen-Tab.....	11
	Kartenbeschreibung.....	11
	Spielereinstellungen.....	11
	Diplomatie.....	12
	Rohstoffe.....	12
	Wetterset.....	12
	DEBUG.....	13
4.4	Briefingassistent-Tab.....	13
	Briefings.....	13
	Pages.....	13
4.5	Questassistent-Tab.....	15
	Quests.....	15
	Behavior.....	15
5	DEBUG nutzen.....	17
	Cheats.....	17
	Eingabeaufforderung.....	18
6	Beispiele: Questassistent.....	18
	Mein erster Quest.....	18
	KI-gesteuerte Armeen.....	19
	Custom Variables.....	19
	Multiple Choice.....	20
7	Beispiele: Scripting.....	20
	Skripte auslagern.....	20
	MapScriptFunction-Behavior.....	20
	Goals.....	21
	Trigger.....	21
	Rewards und Reprisals.....	21

8 Erweiterungen schreiben.....	22
8.1 Einleitung.....	22
8.2 Behavior definition.....	22
8.3 Behavior programmieren.....	23
Reset.....	24
Debug.....	24
CustomFunction.....	24
AddParameter.....	24
Get-Goal/Reward/Reprisal/Trigger-Table.....	24
Das Type-Attribut.....	24

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 (folgend als Map bezeichnet), welches das BBA Tool nutzt um Maps zu öffnen, zu verändern und zu speichern. Ich war und bin nicht an der Entwicklung des BBA Tool beteiligt. Verantwortlicher für das BBA Tool ist der User yoq.

2 Installation

2.1 Installation von Java

Um die Anwendung nutzen zu können wird ein Java Runtime Enviornment (JRE) benötigt. Ich habe den Assistenten für JRE 8 und JRE 12 entwickelt. Alle gängigen Distributionen der JRE sollten Funktionieren.

Beispiele für Runtime Enviornments:

- 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 installiert ist, kannst Du das Archiv mit der Anwendung in ein beliebiges Verzeichnis kopieren. Das Programm wird durch Doppelklick auf die JAR-Datei gestartet. Sollte dies nicht funktionieren, gibt es eine BAT-Datei mit gleichen Namen, die die Anwendung starten wird.

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 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 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 3 Arten von Seiten:

- Dialog: Eine Seite, die hauptsächlich Text anzeigen soll.
- Choice: Der Spieler muss eine Entscheidung zwischen zwei Optionen treffen.
- Separator: Trennt Dialogpfade nach einer Entscheidung ab.

Am Ende eines Briefings können getroffene Entscheidungen über ein spezielles Behavior abgefragt werden.

3.2 Quest

Ein Quest ist ein Job, der sich nach den angegebenen Behavior entsprechend verhält. Quests können im Hintergrund ablaufen 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 und auch wieder aus. Es kann aber nur maximal 8 sichtbare Quests geben!

Das Verhalten eines Quests, also was der Spieler tun oder lassen soll, wird durch Behavior gesteuert. Behavior sind Schablonen, mit denen definiert wird, was während des Quests passieren soll.

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. Quests können alle möglichen Dinge auslösen. Dazu zählen Briefings, Diplomatieänderung, Rohstoffschenkungen, etc.

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.

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_Time(0)
```

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 machinell nach Deinen Angaben erzeugt.

Du kannst das Kartenskript jedoch auch dafür nutzen um eigene Funktionen zu schreiben. Ein Beispiel dafür sind eigene Behavior, welche als Funktion geschrieben werden müssen, damit sie in den "MapScriptFunction"-Behavior genutzt werden können. Du kannst genauso eigene Briefings schreiben oder Zusatzcode hinzufügen. Auch Includes aus dem Projekt 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

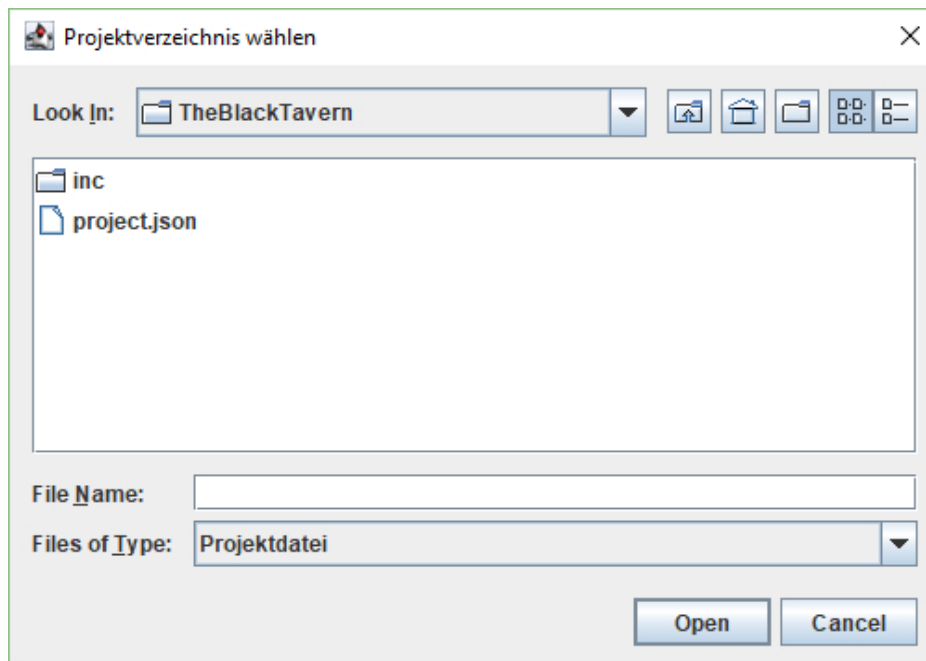
Um an einem Projekt weiterzuarbeiten musst Du es zuerst laden.

Projekt öffnen

Öffne ein Projekt aus einem beliebigen Verzeichnis auf Deinem Computer.

Project öffnen

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

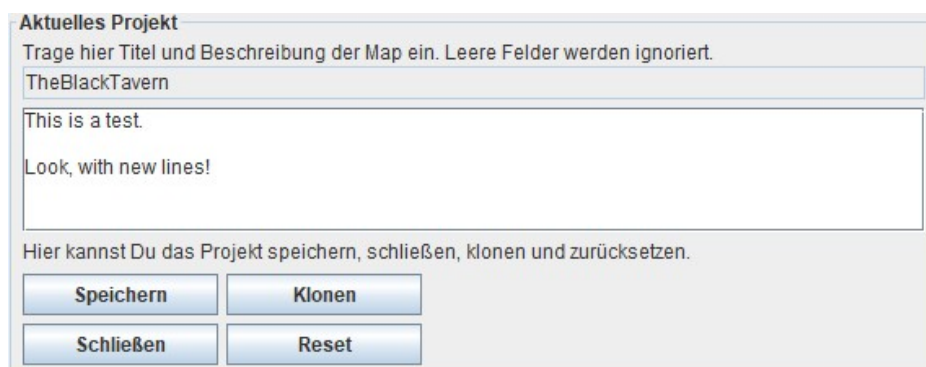
Projekteigenschaften

In den Projekteigenschaften kann die optionale Beschreibung Deines 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. Von dort kannst Du dann z.B. an einem anderen Projekt weiterarbeiten.

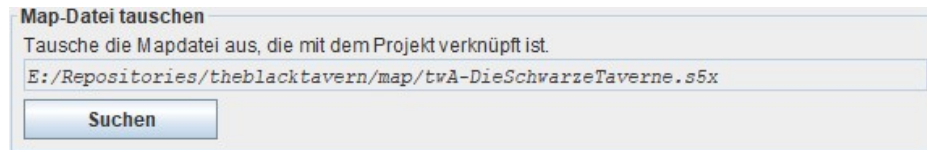
Du kannst Dein Projekt zum letzten Speicherstand zurücksetzen. Dann werden alle Änderungen, die Du bisher gemacht hast, verworfen.

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



Map-Datei

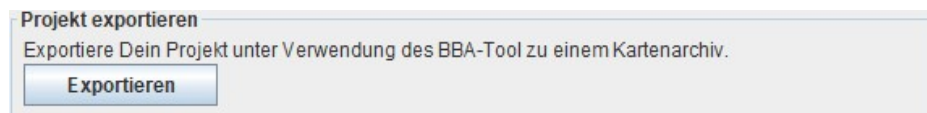
Solltest Du die Map-Datei verschieben, welche im Projekt referenziert ist, kann sie nicht mehr gefunden werden. Export oder das Auslesen von Skriptnamen ist dann nicht mehr möglich. Du kannst jedoch jeder Zeit eine neue Datei hinterlegen. Bei Änderungen an der Map-Datei muss das Projekt neu geladen werden.



The dialog box is titled "Map-Datei tauschen". It contains the instruction "Tausche die Mapdatei aus, die mit dem Projekt verknüpft ist." Below this is a text input field containing the file path "E:/Repositories/theblacktavern/map/twA-DieSchwarzeTaverne.s5x". At the bottom is a button labeled "Suchen".

Export

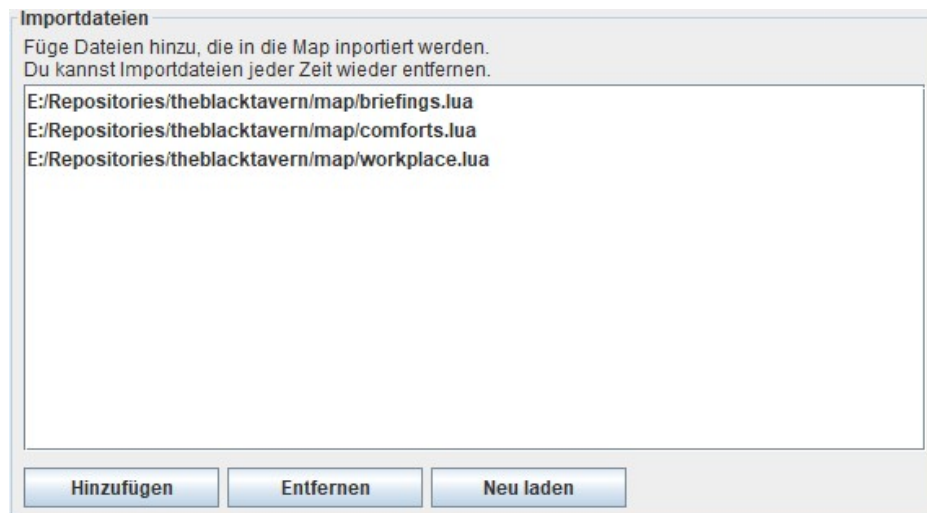
Um aus dem Projekt eine spielbare Map zu machen, muss das Projekt exportiert werden. Beim Klicken auf "Exportieren" wird eine neue Map-Datei im Projektverzeichnis erzeugt. Die referenzierte Map wird nicht verändert.



The dialog box is titled "Projekt exportieren". It contains the instruction "Exportiere Dein Projekt unter Verwendung des BBA-Tool zu einem Kartenarchiv." Below this is a button labeled "Exportieren".

Import

Du kannst weitere Dateien in Dein Projekt importieren. Alle Importe sind sowohl im Projekt als auch später in der Map im Unterverzeichnis `inc` zu finden. Du kannst über das Kartenskript darauf zugreifen.



The dialog box is titled "Importdateien". It contains the instructions "Füge Dateien hinzu, die in die Map importiert werden." and "Du kannst Importdateien jeder Zeit wieder entfernen." Below this is a list box containing three file paths: "E:/Repositories/theblacktavern/map/briefings.lua", "E:/Repositories/theblacktavern/map/comforts.lua", and "E:/Repositories/theblacktavern/map/workplace.lua". At the bottom are three buttons: "Hinzufügen", "Entfernen", and "Neu laden".

Der Assistent erlaubt folgende Dateiformate:

- *.lua, *.luac – Lua-Skripte und Lua-Bytecode
- *.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.

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

Spilereinstellungen

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.

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

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

Taler 0	Lehm 0	Holz 0
Stein 0	Eisen 0	Schwefel 0

Aktualisieren

Wetterset

Wettersets bestimmen, wie die Umgebung aussieht. Du kannst in der Wetterset-Gruppe aus den Standard-sets wählen. Das ausgewählte Set bestimmt dann das Ambiente in dem der Spieler seine Stadt aufbauen wird.

Wetterset
Das Wetterset bestimmt die Belichtungsverhältnisse im Spiel.

SetupHighlandWeatherGfxSet ▼

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

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

aktiv ▼

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.

aktiv ▼

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

abgeschaltet ▼

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

aktiv ▼

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

aktiv ▼

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.

Briefings

Hier kannst Du ein Briefing erstellen, umbenennen oder bearbeiten.

Briefing_HakonTalk ▼

Neu Kopieren Umbenennen 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.

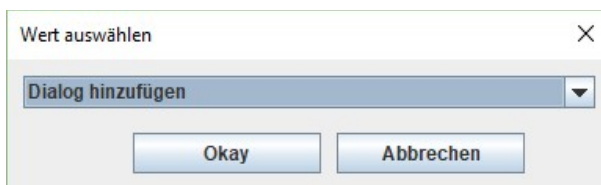
Es gibt 3 Arten von Seiten:

- **Dialogseite:** Die Kamera springt zur Position und zeigt einen Text an. Das Ziel kann entweder aus der Ferne oder aus der Nähe gezeigt werden.
- **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 ein Briefing nach einer Entscheidung zu beenden. Normalerweise wird das komplette Briefing abgespielt. Der Separator sagt, dass an dieser Stelle vorzeitig Schluss ist.



Jeder Seitentyp hat eine Farbe zugeordnet, damit er leichter erkannt werden kann.

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.

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.

0000_InitMission	Sichtbarkeit sichtbar Update Typ SUBQUEST_OPEN Empfänger Spieler 1 Zeitlimit 0 Titel Trautes Heim... Beschreibung 1) Begeht Euch nach Asengarden @cr 2) Nehmt Eure neue Burg in Besitz.
0001_DefeatOnHQDestroyed	
0002_InitBanditArmies	
0003_InitHakonArmies	
0004_InitRakkulArmies	
0005_BanditArmiesDefensive	
0006_VampireArmiesDefensive	
0007_DefeatOnAriKilled	
1001_BriefingArrival	
1002_GoIntoTown	
1003_WelcomeToTheNorth	
1004_TheNewCastle	
1005_FluchtNachVorn	
1010_FindResourcesBriefing	
1012_ResourceRuin1	
1013_ResourceRuin2	
1014_ResourceRuin3	
1015_ResourceRuin4	
1016_FindResourcesFinished	
1020_BuildSomeStuff	

Neu Kopie Umbenennen Löschen

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.

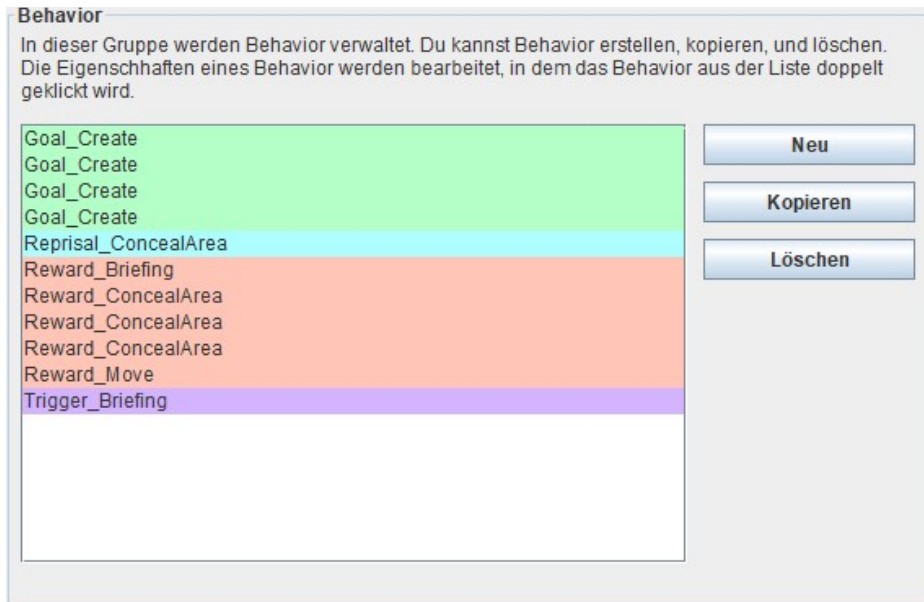
Quests können umbenannt, gelöscht oder kopiert werden. Kopieren bewirkt, dass sie komplett mit Beschreibung und allen Behavior geklont werden. Natürlich muss auch bei einem Quest der Name eindeutig sein.

Wird ein Quest umbenannt, werden auch automatisch alle Erwähnungen im Skript geändert. Du musst nicht von Hand alle Quests auf den alten Namen prüfen.

Behavior

Ein Behavior wird verwendet um den Ablauf eines Quests zusammenzubauen. Du kannst dir die Behavior eine Bausteine vorstellen. Sie können beliebig kombiniert werden und ergeben jedes Mal ein anderes Bild.

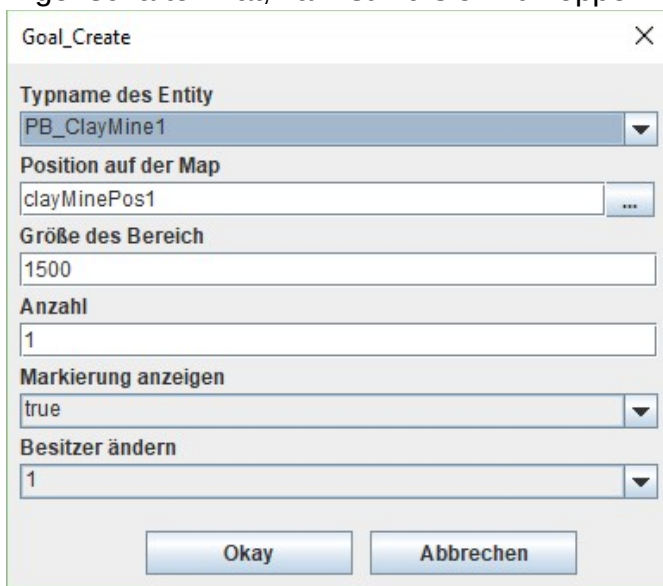
Natürlich gibt es Regeln. Es gibt immer Regeln. Jeder Quest benötigt mindestens ein Behavior vom Typ `Goal` und eins vom Typ `Trigger`. Quests können auch mehrere Behavior eines Typs haben. Beispielsweise sollen 5 Gebäude gebaut werden. Der Auftrag wird erst dann erfolgreich abgeschlossen, wenn der Spieler alle 5 Gebäude gebaut hat. Ebenso werden immer erst alle Trigger geprüft. Belohnungen und Strafen werden dann nach Ergebnis des Quests ausgeführt.



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.

Du kannst Behavior hinzufügen, entfernen oder kopieren. Es ist aber nicht erlaubt sie umzubenennen. Die Reihenfolge in der Liste ist unwichtig. Wenn ein Behavior Eigenschaften hat, kannst Du sie mit Doppelklick auf das Behavior ändern.



Es erscheint ein Dialog, in dem Du die Eigenschaften einstellen kannst. Dabei darf, außer bei Message Feldern, keines der Felder ohne Wert sein.

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 + 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 + G	GUI an-/ausschalten
CTRL + F	Nebel des Krieges an-/ausschalten
CTRL + 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	Kontrollierenden Spieler durchschalten
CTRL + ALT + 1	Bastardschwertkämpfer unter der Mouse spawnen
CTRL + ALT + 2	Schwere Armbrustschützen unter der Mouse spawnen
CTRL + ALT + 3	Schwere Scharfschützen unter der Mouse spawnen
CTRL + ALT + 4	Schwere Kavalerie unter der Mouse spawnen
CTRL + ALT + 5	Belagerungskanone 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.

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 verstört 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)
```

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 Armee Militärbauwerke 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 Schatztruhen suchen lassen.

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

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

Hier würde der Auftrag abgeschlossen, wenn 20 Schatztruhen gefunden sind.

Die Custom Variables ersparen Dir in vielen Fällen das Schreiben eines eigenen Behavior im Kartenscript.

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.

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 aber 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 Callbacks¹ und geben nichts zurück. Alle Statements² werden nacheinander ausgeführt ohne Fragen zu stellen.

¹ Callback: Funktion, die als Reaktion auf ein Ereignis aufgerufen wird.

² Statement: Ein Befehl im Code

Goals

Goals sind Ziele, die der Spieler erreichen muss.

Ob ein Ziel erreicht ist, wird durch eine Fallunterscheidung³ 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, muss der Trigger false zurückgeben.

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

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": [  
    ["NumberInput", "Zahl", []]  
  ]  
}
```

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.

ChoiceName	Ein Textfeld zur Auswahl einer Entscheidung (Multiple Choice). Über den nebenstehenden Button werden alle verfügbaren Entscheidungen angezeigt und eine gewählt werden.
QuestName	Ein Textfeld zur Auswahl eines Quest-Name. Über den nebenstehenden Button werden alle verfügbaren Quests angezeigt und einer 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 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`.