Inhaltsverzeichnis

Anwendungen für Fortgeschrittene....2
Schnelles Testen....2
DEBUG aktivieren....2
Den Ablauf der Map definieren....3
Aufträge im Skript....3
Lokalisierung von Aufträgen....4
Angaben für einen Auftrag....4
Interaktive Objekte....5
Nichtspieler-Charaktere....6
Funktionen....6
Behavior....6
Mission-Briefings und Cutscenes....7
Grundlagen für Dialoge und Missionsbeschreibungen....7
Weggablungen in Dialogen....8
Grundlagen für Kameraflüge....9
Relative Positionsangaben....10
Das Burglager....11
Die Nutzung....11
Die Vorteile....12

Anwendungen für Fortgeschrittene

Die QSB kann wie gewöhnt mit dem Assistenten im Karteneditor verwendet werden. Alle Standardfunktionen stehen Dir zur Verfügung. Viele Rosinen bleiben dem Nutzer des Assistenten jedoch verwehrt. Möchtest Du das volle Potential der QSB nutzen, so wirst du Wohl oder Übel dazu übergehen müssen, Aufträge im Skript zu erzeugen.

Schnelles Testen

Nach *jeder* Änderung im Skript müsste man eigentlich das Spiel verlassen, den Editor starten und das Skript neu importieren. Das 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 Dateien, da es zuerst prüft, ob es sich in der globalen oder lokalen Environment[1] 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.

Ein Beispiel für einen angepassten Pfad könnte wie folgt aussehen:

```
local PathToProject = "E:/Maps/Scripts/mf02_gutkirschenessenmitmanfred";
```

Solange der Pfad stimmt und alle Dateien vorhanden sind, wird es funktionieren.

DEBUG aktivieren

Bevor Du in den kreativen Schöpfungsprozess einsteigst, möchtest Du vielleicht den Debug[2] aktivieren, um Fehler schneller entdecken zu können. Der Debug wird mit der Funktion `API.ActivateDebugMode()` aktiviert. Die Argumente sind die gleichen wie bei `Reward_DEBUG` und werden das Gleiche auslösen.

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

Hier werden alle Funktionen, außer der Statusverfolgung, eingeschaltet. Die Statusverfolgung sollte man nur dann Benutzen, wenn man wissen muss, wann welcher Quest seinen Status ändert. Schaue in die Dokumentation, wenn du wissen möchtest, was welcher Parameter bedeutet.

1 Environment: (eng. „Umgebung) Bezeichnet einen für sich abgeschlossenen Bereich

2 Debug: Kommt aus der Sprache der Software-Entwickler. Fehler werden aus Bugs (Käfer) bezeichnet. Fehler beseitigen bedeutet also Ungeziefer auszumerzen, das einem das Leben schwer macht.

Den Ablauf der Map definieren

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[3], 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 und Eigenschaften des Quests weglassen zu können. Dies geht einher mit einer neuen Schreibweise.

Ein Beispiel für die neue Schreibweise:

```
local QuestName = API.CreateQuest {
    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"),
}
```

Jeder Auftrag benötigt wenigstens einen Namen. Man kann jedoch leicht den Überblick verlieren, da es vermutlich eine größere Anzahl an Quests geben wird. Deshalb bietet sich die Einteilung in Abschnitte an. Jeder Auftrag bekommt einen Abschnitts-Präfix vor seinen Namen.

K1Q1_Langvogt_werden
K1Q2_Verlorene_Schafe_finden
K1Q3_Belohnung_Schafe_gefunden
…

Zusätzlich kannst Du, solltest du Nutzer des BBA-Tools sein, das Skript ich mehrere Skripte aufteilen. Diese Skripte können auch mit Skript.Load geladen werden.

```
Script.Load("path/to/your/script.lua");
```

Der Name des Quest weggelassen werden. In diesem Fall wird ein automatischer Bezeichner als Name gewählt. Der Name des Quests wird von API.CreateQuest zurückgegeben und kann auch so auf andere Behavior übertragen werden.

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

```
local QuestName = API.CreateQuest {
    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),
}

local NextQuest = API.CreateQuest {
    Suggestion  = "Ich bin der Abt des Klosters!",
    Sender      = 4,
    Trigger_OnQuestSuccess(QuestName, 10)
}
```

Lokalisierung von Aufträgen

Die QSB kann Texte in Deutsch und Englisch wiedergeben. Die Sprache wird automatisch erkannt, es wird aber nur Deutsch und Englisch unterstützt.

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

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 Auftrag nur ein Goal haben kann. Auslöser, oder auch Trigger, können mehrere vorhanden sein. Sie müssen alle wenigstens einmal ausgelöst haben, damit der Auftrag startet.

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.

Natürlich können interaktive Objekte auch mit Aktivierungskosten versehen werden.

```
CreateObject {
    Name  = "fire1",
    Title = "Signalfeuer",
    Costs = {Goods.G_Wood, 30, Goods.G_Honeycomb, 5},
};
```

Ein interaktives Objekt kann auch eine Belohnung beinhalten.

```
CreateObject {
    Name   = "chest1",
    Title  = "Schatztruhe",
    Reward = {Goods.G_Gold, 3000},
};
```

Die verschiedenen Eigenschaften des Objektes können beliebig gesetzt oder weggelassen werden. Schreibt man sie nicht, werden Standardwerte benutzt. Für eine Liste aller möglichen Eigenschaften, sieh Dir bitte die Dokumentation an.

Nichtspieler-Charaktere

Funktionen

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 auf der Spielwelt glitzert.

```
API.NpcCompose {
    Name     = "hakim",
    Callback = function(_Npc, _Hero)
       -- Hier kann was passieren
    end,
}
```

Dies ist ein Beispiel für einen einfachen NPC.

NPCs haben noch weitere Funktionalitäten. Du kannst einen Helden vorgeben, der den NPC ansprechen muss. Diese Option ist allerdings nur sinnvoll, wenn der Spieler in Deiner Map mehr als einen Helden gleichzeitig steuert. Für die meisten Maps ist diese Angabe daher unnötig.

```
API.NpcCompose {
    Name             = "hakim",
    Hero             = "marcus",
    WrongHeroMessage = "Ich spreche nicht mit dir!",
    Callback         = function(_Npc, _Hero)
       -- Hier kann was passieren
    end,
}
```

Wird der NPC nun mit einem Helden angesprochen, dessen Skriptname nicht `marcus` ist, wird das Callback nicht ausgelöst und der Text von `WrongHeroMessage` als Nachricht am linken Rand auf dem Bildschirm ausgegeben.

Behavior

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

```
Goal_NPC("hakim", "hero")
```

Als erstes wird der Name des NPC angegeben. Als zweites kann optional der Name des Helden folgen, der den NPC ansprechen muss. Je nach dem, ob ein Held vorgegeben ist oder nicht, wird sich das Auftragsfenster anpassen.

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[4] des Spiels einen entsprechenden Kameramodus an, der für die Thronsaalszenen in der Kampagne genutzt wird. Das so genannte Briefing System kann diesen Kameramodus zweckentfremden und es ermöglichen, Missions-Briefings zu erzeugen. Dabei unterteilt es in zwei verschiedene Arten von Briefings. Das Missions- oder Dialog-Briefing für Beschreibungen und der Kameraflug oder Cutscene für szenerische Darstellung.

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. Damit dies funktioniert, muss der Quest angegeben werden, an den das Briefing gehangen wurde.

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

Grundlagen für Dialoge und Missionsbeschreibungen

Damit das BriefingSystem ordnungsgemäß arbeiten kann, muss ein Grundschema eingehalten werden. Es ähnelt dem altbekannten Schema, wurde aber um einige neue Inhalte ergänzt. Das 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 Missions-Briefing, 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

4 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.

übersprungen 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 vereinbart, 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. Du darfst auf keinen Fall das return vergessen, sonst funktioniert es nicht!

Weggablungen in Dialogen

Weggabelungen in Dialogen sind unter der Bezeichnung Multiple Choice bekannt. Man versteht darunter einen Auswahldialog mit mehreren Optionen, aus denen der Spieler eine wählen muss, damit er fortfahren kann. 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 bietet die Möglichkeit zu einer anderen Seite zu wechseln, ohne das ein weiterer Auswahldialog benötigt wird. Als Argument wird die Nummer der Zielseite angegeben.

```
AP(11);
```

Der Auswahldialog ist der wichtigste Teil. Hier wählt der Spieler die Optionen aus. ASMC ähnelt dabei APS, nur das an die Stelle des Callbacks die Optionen getreten sind.

```
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! Versuche nicht mehr als 8 Möglichkeiten pro Seite anzuzeigen.

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. Die Nummer entspricht von oben betrachtet der Position der 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.

Grundlagen für Kameraflüge

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. Für den Anfang wäre es weise diese zu übernehmen. Später spricht nichts dagegen mit den möglichen Einstellungen herum zu experimentieren.Da wir etwas von der Welt sehen wollen, verwenden wir die schmale Dekoration. Der Nutzer soll die Landschafft genießen, daher werden die Seiten automatisch übergehen. Zum Schluss werden noch ein paar Schönheitseinstellungen gemacht. Wer will, kann auch die Spielgeschwindigkeit am Ende des Briefings wiederherstellen lassen. Dies wird notwendig, wenn Du nicht willst, dass Briefings höhere Spielgeschwindigkeiten herabsetzen.

Eine Cutscene verwendet ausschließlich die Funktion AF. Natürlich kann auch AP Cutscenes nutzen, allerdings bietet AF eine übersichtlichere Notation. Du musst Dir keine Gedanken darüber

machen, wie viele Seiten Dein Kameraflug hat. Außerdem willst Du keine permanenten Änderungen vornehmen, wie z.B. Gebiete aufdecken und Markierungen anbringen.

Aus diesem Grund müssen die einzelnen Seiten nicht zugreifbar sein. AF kann aus einer Liste von Positionen eine Bewegung erzeugen, die alle Positionen nacheinander abfährt.

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 Dauer der Anzeige wird mit Duration eingestellt. Alle Stationen eines Fluges, außer dem Startpunkt, teilen sich die Gesamtdauer der Bewegung gerecht untereinander auf.

Hat ein Flight z.B. 5 Punkte (Startpunkt und 4 Stationen) und eine Anzeigedauer von 12 Sekunden, wird erst zum Starpunkt gesprungen und danach jede der 4 Stationen angeflogen. Die Bewegung von einem Punkt zum anderen wird durch 4 geteilt. Also dauert die Anfahrt zum nächsten Punkt immer 3 Sekunden (12 Sekunden Anzeigedauer / 4 Positionen = 3 Sekunden Flugzeit).

```
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,
};
```

Abgeschlossen wird die Cutscene abermals mit ihrem Fuß. Er besteht, wie gewohnt, aus einer Finished-Funktion und dem Startbefehl.

```
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.

Relative Positionsangaben

Mit Koordinaten im dreidimensionalen Raum zu arbeiten kann auf Anfänger abschreckend wirken. Daher gibt es neben der Angabe von XYZ-Koordinaten auch die Möglichkeit, Entities zu verwenden. Man kann entwede die direkte Position eines Entity verwenden, oder die relative Position in einem bestimmten Abstand und Winkel.

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

In diesem Beispiel ist die Blickrichtung absolut und die Position der Kamera relativ. Natürlich geht das auch andersrum. Mit diesen speziellen Einstellungen sind interessante Kameraeffekte möglich. Scheue Dich nicht, es zu probieren!

Das Burglager

In DIE SIEDLER - Aufstieg eines Königreichs ist es nicht vorgesehen endlos viele Rohstoffe und Waren zu hamstern, wie in den Vorgängern der Reihe. Über die Jahre gab es viele Anläufe ein zusätzliches Lager bereitzustellen. Dies wurde geläufig als „Burglager“ bezeichnet. Ein Problem teilten jedoch alle diese Burglager: Sie konnten sich nicht richtig ins Spiel eingliedern. Tribute an andere Spieler oder Aktivierungskosten von interaktiven Objekten konnten nicht direkt aus diesen Lagern beglichen werden.

Oftmals mussten zusätzliche Grafiken mit der Map ausgeliefert werden und die Steuerung der Lager ließ viel zu wünschen übrig. Glücklicher Weise gehören diese Zeiten nun der Vergangenheit an! Symfonia bringt ein eigenes Burglager mit, das alles bisher dagewesene um Längen ins Abseits drängt und in Vergessenheit geraten lassen wird.

Die Nutzung

Das Burglager wird für den angegebenen Spieler initialisiert und kann sofort danach verwendet werden. Diese Funktion muss immer zuerst aufgerufen werden, bevor irgend etwas mit dem Burgtlager gemacht wird.

```
API.CastleStoreCreate(1);
```

Ein Burglager kann natürlich jeder Zeit wieder entfernt werden. Alle Waren im Burglager sind dann jedoch unwiederbringlich verloren.

```
API.CastleStoreDestroy(1);
```

Dem Burglager können auch direkt Waren hinzugefügt oder daraus entfernt werden.

```
API.CastleStoreAddGood(1, Goods.G_Wool, 25);
API.CastleStoreRemoveGood(1, Goods.G_Wool, 25);
```

Die Menge an Waren im Burglager können abgefragt werden.

```
Local WoolAmount  = API.CastleStoreGetGoodAmount(1, Goods.G_Wool);
local TotalAmount = API.CastleStoreGetTotalAmount(1);
```

Der voreingestellte Basiswert zur Berechnung des Lagerlimits kann geändert werden.

```
API.CastleStoreSetBaseCapacity(1, 150);
```

Die Vorteile

Man kann viel über die Vor- und Nachteile philosophieren. Deshalb habe ich einen direkten Vergleich gezogen, den du nachstehender Tabelle entnehmen kannst.

Feature	Symfonia Lager	Sonstige Lager
Einlagerung von Waren aus dem Lagerhaus	✔	✔
Automatische Verwaltung, sobald Platz verfügbar ist	✔	✔
Möglichkeit Waren im Lager zu reservieren	✔	(✔)
Möglichkeit Waren nicht zur Lagerung zuzulassen	✔	(✔)
Lagerzustand eines Warentyps einzeln bestimmen	✔	✘
Keine externen Texturen benötigt	✔	✘
Tribute können aus dem Burglager beglichen werden	✔	✘
Interaktive Objekte mit Waren aus der Burg aktivieren	✔	✘
Gemischte Kostenbegleichung (Lagerhaus und Burg)	✔	✘