RPG Maker XP Kurs
Teil 2

Inhaltsverzeichnis

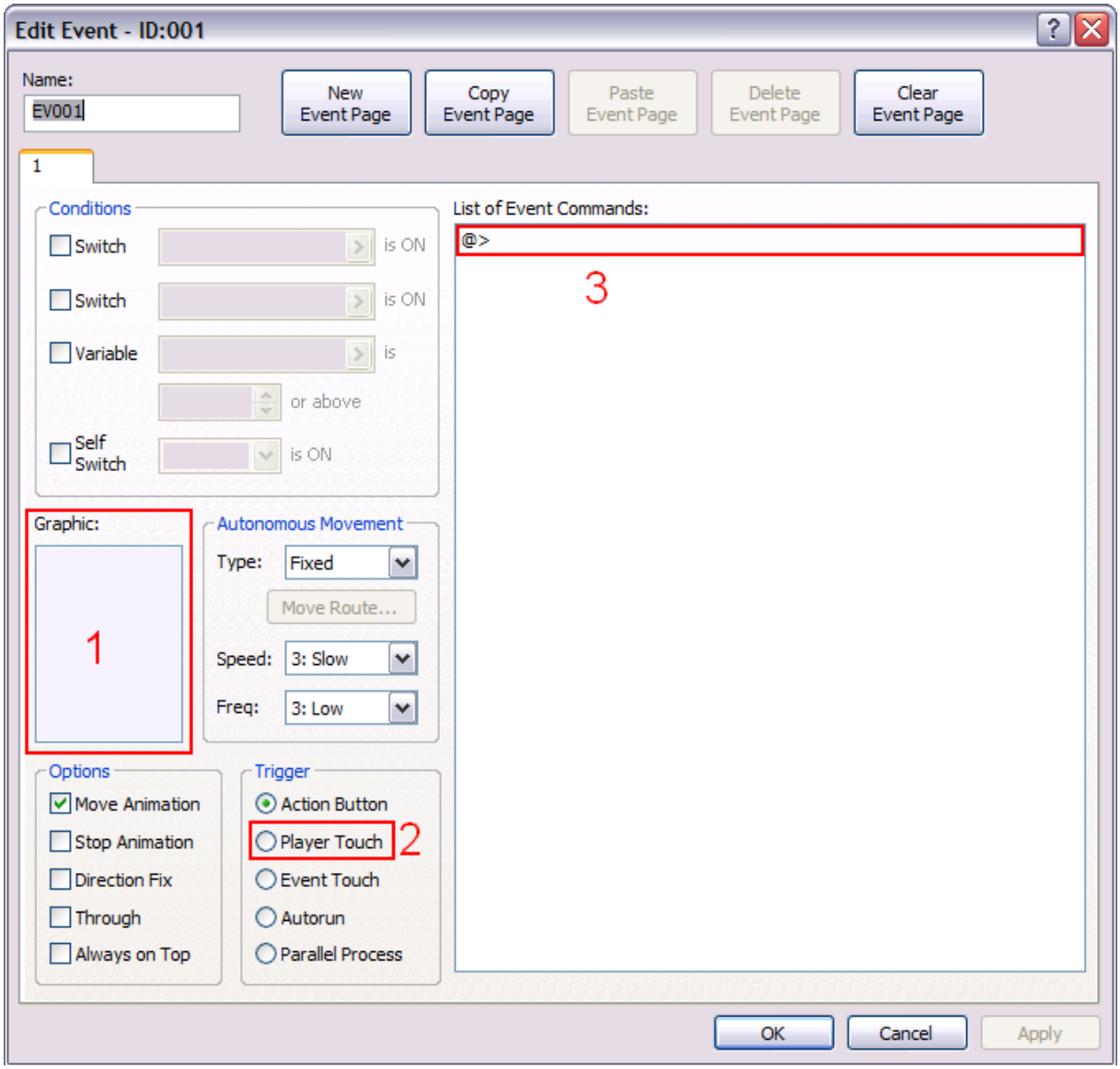
- **Einleitung**
- **Mapübergänge**
- **Einkaufs- und Übernachtungsmöglichkeiten**
  - **Einbauen von Textnachrichten**
  - **Komplexe Codestrukturen durch Fallunterscheidungen**
  - **Oft gebrauchte Eventbefehle**
- **Aufgaben**
  - **Vorbereitung der Aufgabe**
  - **Lokale Schalter als Informationsträger**
  - **Mehrseitige Events**
  - **Variablen als Informationsträger**
  - **Scriptübersicht**
  - **Wiederholung von Schaltern und Variablen**
- **Schlusswort**
- **Register**

In zweiten Teil des RPG-Maker XP Kurses beschäftigen wir uns näher mit den Eventcommands und mit typischen Anwendungsbeispielen. Ziel dieses zweiten Kurses soll sein, dass ihr bereits ein eigenes Spiel mit allen typischen Facetten erschaffen könnt. Natürlich setze ich voraus, dass ihr den ersten Kurs bereits gelesen habt. Denn wir werden häufig auf bereits erlerntes Wissen zurückgreifen.

Mapübergänge

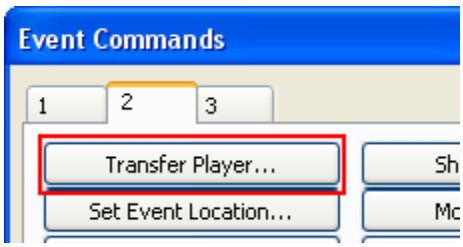
Als erstes beschäftigen wir uns mit dem Erstellen von **Teleportevents**. Im ersten Kurs habt ihr bereits mehrere Maps erstellt. Doch ist euer Held noch nicht in der Lage, von einer Map zur Nächsten zu wechseln. Hierfür verwenden wir Teleportevents. Was ein Event ist, wurde ja bereits besprochen: Ein besonderes Ereignis auf der Map, z.B. Ein NPC (Nichtspielercharakter), eine Schatzkiste, eine Tür oder eben ein Mapübergang.
Geht in eine eurer Maps und wechselt durch einen Klick auf den

 Eventebene-Button, oder über die F8-Taste in den Eventmodus. Auf eurer Map erscheint nun ein Gitternetz. Ein Event kann immer auf einem der Felder im Gitternetz platziert werden. Klickt ein Feld am Rande eurer Map doppelt an, und es öffnet sich das Event-Fenster.

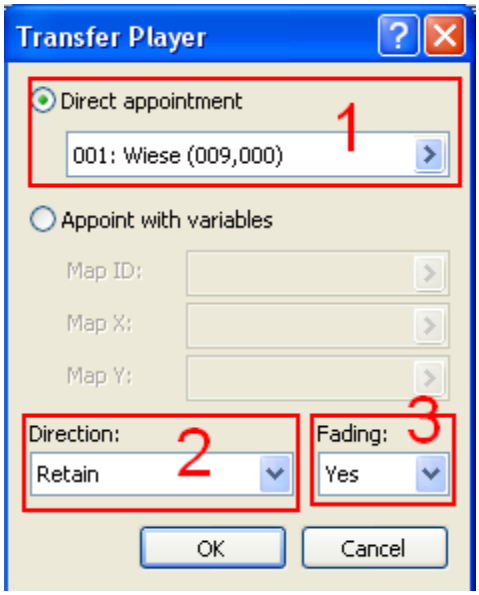


Der Aufbau des Eventfensters sollte euch ja schon aus dem letzten Tutorial bekannt sein. Bei „Graphic“ (1) wählen wir die Grafik des Teleportevents. Nun, normalerweise ist solch ein Event unsichtbar. Deshalb belassen wir die Grafik des Events auf „(None)“.
Bei „Trigger“ (zu deutsch: Auslöser) können wir einstellen, wie das Event ausgelöst bzw. aktiviert wird. Wir wollen einen Teleporter an den Rand unserer Map setzen. Sobald der Spieler diesen Teleporter berührt, soll er sich in die nächste Map teleportieren.
Deshalb wählen wir „Player Touch“ (2). Dadurch wird das Event dann aktiviert, wenn der Spieler das

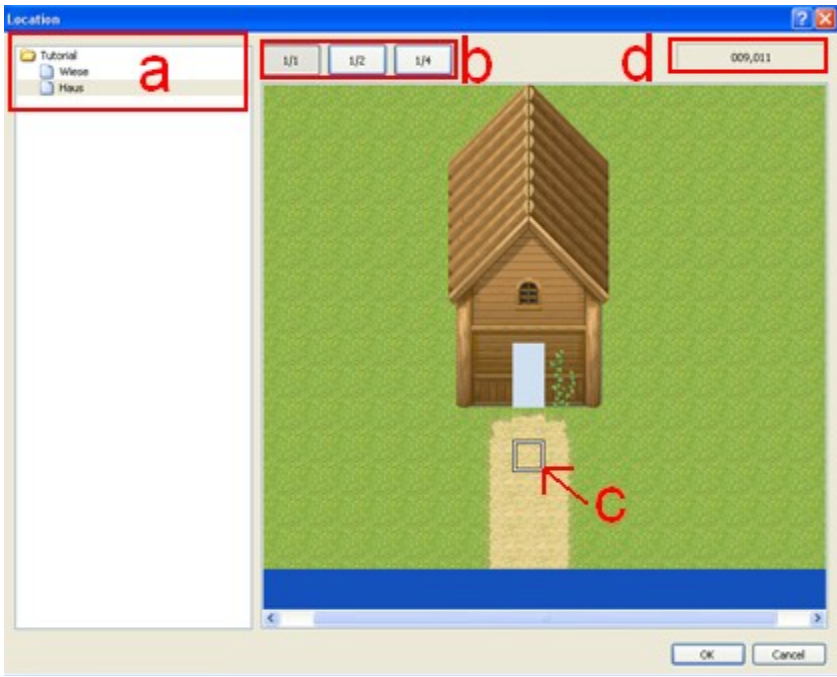
Event berührt.
Nun wenden wir uns dem eigentlichen Teil der Eventerstellung zu: Den *Event Commands* (Ereignisbefehle).
Klickt doppelt auf die weiße, leere Fläche rechts (2). Nun öffnet sich die Liste mit den *Event Commands*.



Zum Teleportieren des Spielers benutzen wir den Befehl „Transfer Player“. Er befindet sich auf Seite 2, ganz oben links. Sobald ihr den Button „Transfer Player“ anklickt, erscheint ein neuer Dialog. Nun gilt es festzulegen, wo der Spieler hinteleportiert werden soll.



Das Dialogfenster bietet verschiedene Auswahlfunktionen, von denen wir jedoch vorerst nicht alle behandeln wollen. „Direct appointment“ (1), was soviel wie „Direkte Bestimmung des Zielorts“ bedeutet, ermöglicht es euch den genauen Ort zu bestimmen, an dem der Spieler teleportiert werden soll. Klickt das Feld also doppelt an. Nun erscheint ein neues Fenster. Links erkennt ihr eine Liste (a) aller im Spiel vorhandenen Maps. Sobald ihr eine der Maps auswählt, erscheint eine Vorschau im rechten Teil des Fensters. Wählt nun also die Map aus, zu der der Held sich teleportieren soll. Die drei Buttons rechts neben der Liste (b) sind zum ran- und wegzoomen der Map. Ihr kennt sie sicherlich noch aus dem vorherigen Tutorial, denn sie fanden auch bei der Maperstellung ihre Anwendung.



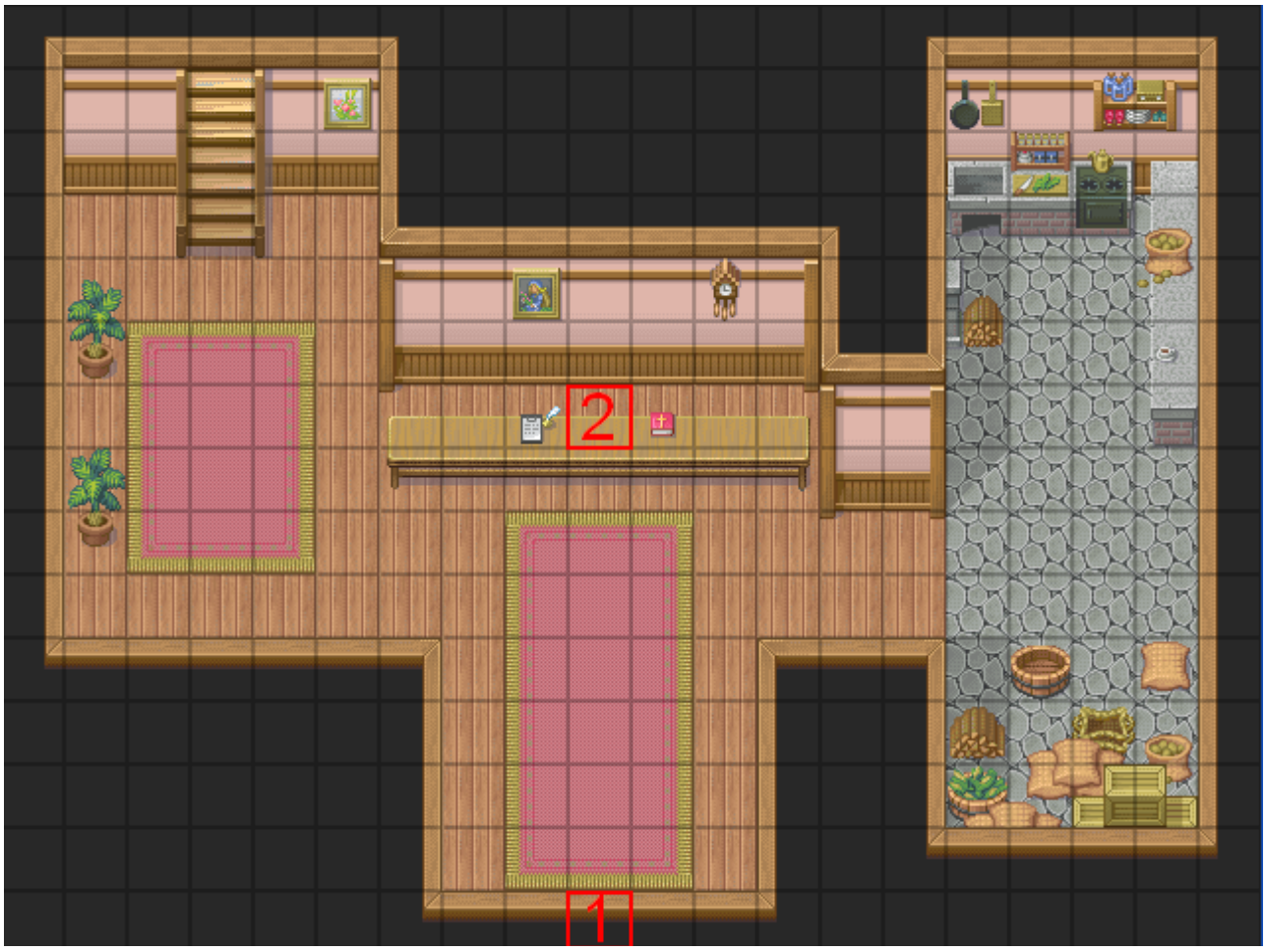
Sobald ihr ein Feld auf der Mapvorschau anklickt, erscheint auf eben diesem ein weiß umrahmtes Kästchen (c). Es gibt die genaue Stelle an, zu der der Held teleportiert werden soll. Oben rechts werden zudem noch die genauen Koordinaten des Zielortes angegeben (d). Auf diese Weise könnt ihr konkret den Zielort der Teleportation wählen. Klickt auf OK, und ihr gelangt zurück zum vorherigen Dialog. Das mittlere Kontrollfeld „Appoint with variables“ werden wir vorerst auslassen. Erst später werden wir uns den Variablen zuwenden. Doch die unteren beiden Felder sind für die Teleportation doch relativ praktisch. Bei „Direction“ (2), was soviel wie Richtung bedeutet, könnt ihr angeben, in welche Richtung der Spielcharakter gucken soll, nachdem er teleportiert wurde. Ihr könnt entweder die vier Richtungen „down“ (unten), „left“ (links), „right“ (rechts), „up“ (oben) wählen, oder aber „Retain“ (beibehalten). Letzteres bewirkt, das der Spieler in dieselbe Richtung blickt wie vor dem Teleport. Der Einfachheit halber können wir die *Direction* stets bei *Retain* belassen.
„Fading“ (3), auf deutsch Überblendung, ist reine grafische Spielerei. Habt ihr „yes“ (ja) ausgewählt, so wird der Mapwechsel durch einen fließenden Übergang realisiert. Bei „no“ (nein) geschieht die

Teleportation abrupt. Natürlich sieht es bei „yes“ doch deutlich schöner aus, weshalb ich raten würde, dieses anzuwählen.
Nachdem alle Einstellungen getätigt wurden, klickt auf Okay. Ihr könnt euren neuen Teleport gleich über Testplay ausprobieren.
Durch die Teleportfunktion seid ihr nun in der Lage, Verbindungen zwischen den Maps herzustellen.

Einkaufs- und Übernachtungsmöglichkeiten

Einbauen von Textnachrichten

Die typischen Städte in einem RPG bestehen aus Wohnhäusern, Waffen-, Rüstungs- und Itemläden, sowie einem Gasthaus zum übernachten (also ein INN).
Glücklicherweise verfügt der RPG-Maker XP bereits über Standardbefehle, um derartige Einrichtungen leicht zu erzeugen. Wir erstellen also eine neue Map mit dem Namen „Gasthaus“



So beispielsweise könnte das Gasthaus aussehen. Das Feld, welches mit der Nummer 1 gekennzeichnet ist, steht für den Ausgang des Gasthauses. Hier setzen wir also ein Teleportevent hin, welches den Spieler zurück ins Dorf bringt.
Das Feld mit der 2 soll von einem Hotelier belegt werden, welcher das freie Zimmer für den Helden vermietet.
Wir setzen an diese Stelle also ein neues Event, wählen die Grafik „112-Civilian12“ und belassen die übrigen Standardeinstellungen des Events.

Ein Doppelklick auf die große weiße Fläche rechts im Event-Fenster öffnet wieder die Liste mit den *Eventcommands*.
Dieser Hotelier soll den Spieler fragen, ob er im Gasthaus übernachten möchte. Die Übernachtung wird 20 Goldmünzen kosten. Doch wenn der Spieler zustimmt, wird er komplett geheilt und ihm werden die 20 Münzen abgezogen. Lehnt er ab, soll der Hotelier sich vom Spieler mit einem enttäuschten „Auf Wiedersehen“ verabschieden.
Um eine Textnachricht anzuzeigen bedienen wir uns der Funktion **„Show Text“**. Sie ist gleich die allererste Funktion der Event Befehle, also auf Seite 1 oben links.
Sofort öffnet sich ein Dialogfenster, in dem wir den Text, den der NPC sprechen soll, eingeben können. Uns stehen aber auch einige Formatierungsmöglichkeiten offen, um diesen Text auszuarbeiten. Diese Formatierungsbefehle ruft man mit so genannten **Kontrollzeichen** auf. Es gibt folgende Kontrollzeichen:

- \n[x] wird durch den Namen des Helden mit der Nummer x ersetzt
- \v[x] wird durch den Wert der Variable mit der Nummer x ersetzt.
- \g öffnet ein Fenster, welches den Geldbetrag des Spielers anzeigt
- \c[x] verändert die Farbe des Textes.
- \ zeigt das nachfolgenden Kontrollzeichen als Text an, statt es auszuführen

Kontrollzeichen sind eine wirklich einfache Sache um einige Besonderheiten in euren Text einzubauen. Zur Übung werden wir versuchen unserem Hotelier so viele Kontrollzeichen wie möglich zu geben.
Standardmäßig könnte der Text zum Beispiel folgendermaßen lauten:
Hotelier: Hallo, möchten sie ein Zimmer für 20 Münzen?
Doch was wäre wenn der Hotelier den Helden zufällig kennt? Vielleicht wohnt der Held ja im gleichen Dorf wie der Hotelier. In diesem Fall müsste der NPC den Helden auch mit Namen ansprechen können. Das funktioniert, wie bereits gesagt, über die \n[x] Funktion. Im ersten Teil des Tutorials haben wir den Barbarenhelden Alex erstellt. Er müsste in der Datenbank unter „Actors“ die Nummer 9 haben. Deshalb können wir durch den Befehl \n[9] in der Textnachricht den Namen des Helden mit der Nummer 9, also unseren Alex, anzeigen lassen. Sollte Alex bei euch eine andere Nummer haben, so gebt diese statt der 9 ein.
Unsere Nachricht heißt also nun:
Hotelier: Hallo \n[9], möchten sie ein Zimmer für 20 Münzen?
Das \n[9] wird im Spiel automatisch durch „Alex“ ersetzt.
Als nächstes wollen wir den Text noch farblich etwas verschönern. So könnte doch der Abschnitt „Hotelier:“ in einer anderen Farbe geschrieben werden, als das Gesprochene. Hierfür benutzen wir das Kontrollzeichen \c[X]. Statt X können wir eine Zahl von 0 bis 7 eingeben. Diese Zahlen stehen für folgende Farben:

Wir wollen den Namen „Hotelier:“ in grüner Schrift anzeigen, während wir den eigentlichen Text in weiß geschrieben haben wollen.
Deshalb schreiben wir in die „Show Text“ Box:
\c[3]Hotelier: \c[0]Hallo \n[9], möchten sie ein Zimmer für 20 Münzen?

Zum Schluss wollen wir noch, dass am oberen Bildschirmrand der Geldbetrag des Spielers angezeigt wird. Denn bevor der Spieler leichtfertig 20 Münzen ausgibt, möchte er sicher erst einmal prüfen, ob er überhaupt soviel Geld hat. Hierfür verwendet man das Kontrollzeichen \g. Das \g kann dabei an eine beliebige Stelle des Textes gesetzt werden.

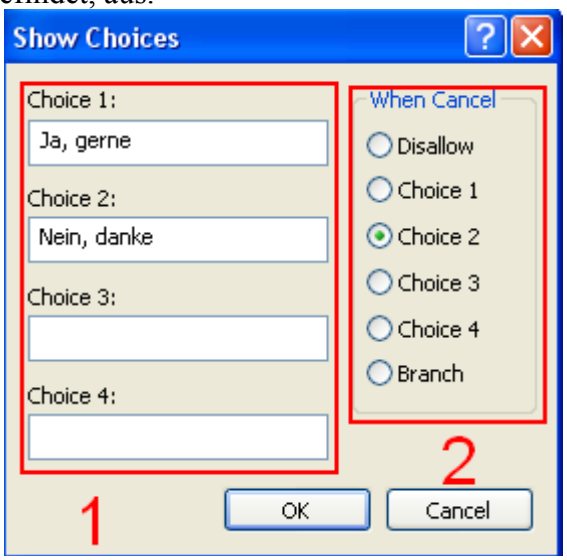
Unser fertiger Text sieht also folgendermaßen aus:
\c[3]Hotelier: \c[0]Hallo \n[9], möchten sie ein Zimmer für 20 Münzen? \g

Die übrigen zwei Kontrollzeichen werden wir in diesem Fall wohl nicht gebrauchen können. Dennoch möchte ich ihre Bedeutung kurz erläutern. \v[x] funktioniert ähnlich wie die Heldennamenanzeige \n[x]. Nur werden statt Heldennamen Variablenwerte angezeigt. Was Variablen sind, dazu kommen wir in diesem Tutorial noch.

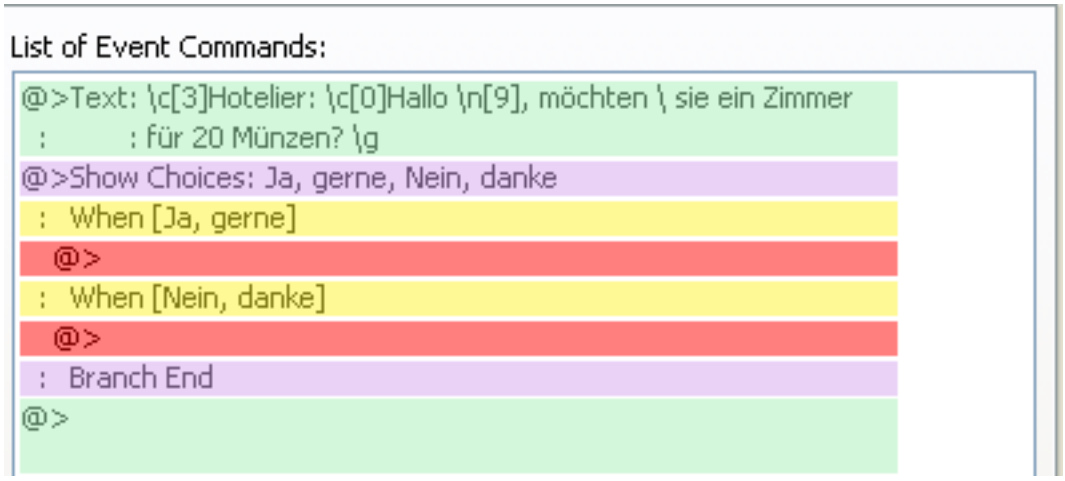
\ dient zum Anzeigen eines Kontrollzeichens im Text. Nehmen wir einmal an, wir wollen ein Tutorial-Spiel entwerfen (was für eine verrückte Idee). Wir erstellen dort einen NPC, der dem Spieler die Kontrollzeichen erklärt.
„Hallo, durch das Kontrollzeichen \g lässt sich der Geldbetrag darstellen“
Was würde im Spiel geschehen? Probiert es doch einmal aus.
Das Kontrollzeichen \g würde aus dem Text verschwinden, und stattdessen würde der Geldbetrag am oberen Bildschirmrand angezeigt werden. In diesem Fall ist das aber nicht erwünscht. Wenn man vor einem Kontrollzeichen \ schreibt, wird dieses nicht beachtet, und normal ausgeschrieben.
„Hallo, durch das Kontrollzeichen \\g lässt sich der Geldbetrag darstellen“
Diese kleine Änderung würde bewirken, das \g im Text normal angezeigt, und kein Geldbetragsfenster geöffnet wird.
Doch für unser Gasthaus ist das ohnehin nicht von Bedeutung. Wenden wir uns also wieder unserem Hotelier zu.

Komplexe Codestrukturen durch Fallunterscheidungen

Nun werden wir unser Wissen ***Show Choice*** (Entscheidungsmöglichkeiten anzeigen) aus dem vorherigen Tutorial wieder anwenden. Denn unser Hotelier hat dem Spieler schließlich eine Frage gestellt, und er wartet sicherlich schon auf eine Antwort.
Ein *Show Choice* erstellt verschiedene Antwortmöglichkeiten, von denen der Spieler eine auswählen kann. In diesem Fall gäbe es zwei Antwortmöglichkeiten: „Ja, gerne“ und „Nein, danke“.
Wir erstellen durch Doppelklick auf die weiße Fläche unter dem Show Text einen neuen Eventbefehl. Diesmal wählen wir die „Show Choice“ Funktion, welche sich direkt unter dem „Show Text“ befindet, aus.

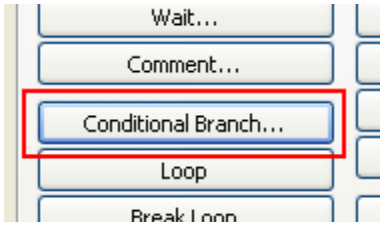


Wie ein *Show Choice* funktioniert, wurde im ersten Tutorial ja bereits geklärt. Dennoch eine kurze Wiederholung: In den einzelnen *Choice*-Feldern (1) werden die verschiedenen Antwortmöglichkeiten eingetragen. Das „When Cancel“ Feld (2) gibt an, was passiert, wenn der Spieler den *Choice* per ESC abbrechen will. „Disallow“ ignoriert die ESC-Taste. *Choice* 1-4 geben an, das beim Abbruch durch ESC automatisch die gewählte Antwortmöglichkeit gewählt wird. „Branch“ erstellt einen Sonderfall für den Abbruch per ESC-Taste. Wir füllen den Dialog wie im Screenshot gezeigt aus.
Im Eventfenster sollte die Eventbefehlsliste nun folgendermaßen aussehen:

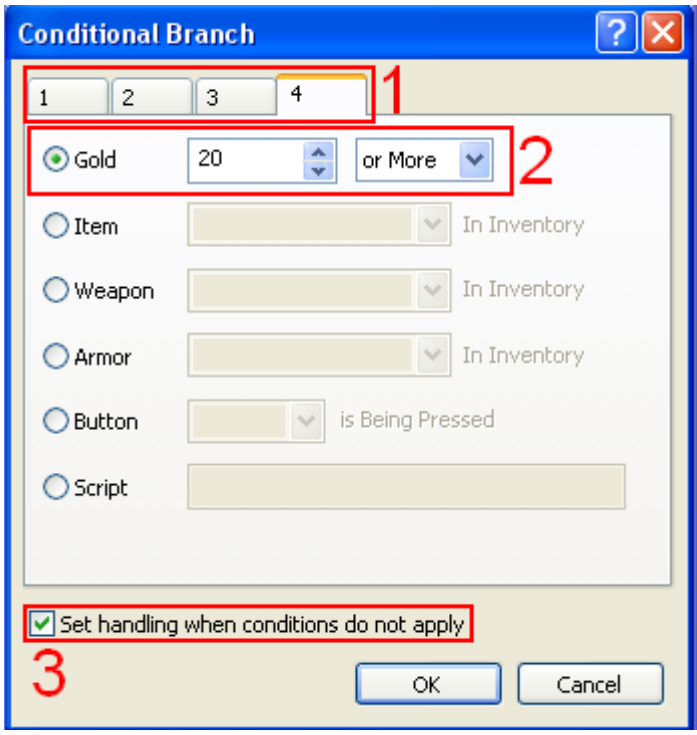


Zur Verdeutlichung der Eventbefehlslistenstruktur habe ich den Screen etwas eingefärbt.
Grün stellt den normalen Ablauf der Eventbefehlsliste dar. Am Anfang das Anzeigen der Textnachricht, am Ende dann eine noch leere Befehlszeile. Dazwischen jedoch wurde die Eventstruktur durch eine Entscheidungsmöglichkeit stark verändert. Violett stellt dabei den „Show Choice“ Befehl dar. Alles was zwischen den beiden violetten Streifen ist, ist Teil der neuen Struktur, welche durch den *Show Choice* geschaffen wurde. Das klingt nun etwas kompliziert, ist aber recht einfach. Die gelben Streifen kennzeichnen die einzelnen Entscheidungsmöglichkeiten: „Ja, gerne“, und „Nein, danke“. Die roten Streifen unter diesen Entscheidungsmöglichkeiten geben an, was passiert, wenn der Spieler sich für diese entscheidet.
Fangen wir also mit dem einfacheren an. Wenn der Spieler „Nein, danke“ sagt, soll der Hotelier sich verabschieden. Wir klicken also auf den (hier) roten Streifen unter dem „Nein, danke“ doppelt drauf. Natürlich gibt es bei euch keinen roten Streifen, daher orientiert euch an dem „When [Nein, danke]“. Die Eventbefehle, die ihr nun einfügt, werden nur dann ausgeführt, wenn der Spieler sich für „Nein, danke“ entscheidet.
Wir fügen also ein „Show Text“ mit dem Inhalt
\c[3]Hotelier: \c[0]Das ist aber schade... Auf Wiedersehen!
ein.
Dieser Text wird also nur dann angezeigt, wenn der Spieler sich für die zweite Auswahlmöglichkeit entscheidet. Probiert es doch einmal aus! Startet euer Testspiel und redet den Hotelier an. Wenn ihr „Ja, danke“ anklickt, sollte nichts passieren. Klickt ihr stattdessen „Nein, danke“ an, so wird sich der Hotelier von euch verabschieden.
Die Entscheidungsmöglichkeiten gehören zu jenen Befehlen, welche die natürliche, lineare Struktur der Eventbefehlsliste durch Fallunterscheidungen ersetzen. Wir werden später noch andere solche Funktionen kennen lernen, so zum Beispiel *Cycles* (Schleifen) und *Conditional Branchs* (Bedingungen).
Doch beim Testplay sollte euch noch etwas anderes aufgefallen sein: Die Textnachricht und die Entscheidungsmöglichkeit wird in ein und demselben Textfenster angezeigt!
Das sieht natürlich weitaus schöner aus, als wenn es in zwei nacheinander folgenden Textfenstern angezeigt werden würde. Doch wie kommt der Maker dazu?
Nun, wenn eine Entscheidungsmöglichkeit direkt nach einer Textnachricht folgt und kein anderer Eventbefehl dazwischen liegt (und sei es auch nur ein Kommentar), so versucht der Maker beides in einer Textbox anzuzeigen. Das geht jedoch nur, wenn Textnachricht und Auswahlmöglichkeit zusammen nicht mehr als 4 Textzeilen einnehmen. Unser Text sollte ungefähr zwei Zeilen lang sein. Unsere Entscheidungsmöglichkeit hat ebenfalls nur zwei Entscheidungen. Beides zusammen ergeben vier Textzeilen. Würden wir auch nur eine Textzeile oder eine Entscheidung mehr einbauen, so würden beide Befehle in separaten Fenstern angezeigt werden.

Nun wenden wir uns der Antwort „Ja, gerne“ zu, und werden hierbei schon wieder etliche neue Eventfunktionen kennen lernen. Darunter auch der oben genannte ***Conditional Branch***.
Der Spieler entscheidet sich also, im Gasthaus für 20 Münzen zu übernachten. Demzufolge müssen wir zuallererst abfragen, ob der Spieler überhaupt genug Geld hat. Wenn dies nicht der Fall ist, kann er auch nicht übernachten. Sollte er jedoch genügend Geld haben, so muss der Bildschirm schwarz werden, eine Übernachtungsmusik muss eingespielt werden, schließlich sollte der Bildschirm wieder hell und die Heldentruppe wieder vollständig geheilt werden.



Um abzufragen ob der Spieler genügend Geld hat, benutzen wir so genannte *Conditional Branchs* (Bedingungen).
Die Eventfunktion „Conditional Branch“ befindet sich auf der ersten Seite, in der linken Spalte der Eventbefehle.
Erstellt diese Funktion unter dem „When [Ja, gerne]“, also in unserem fiktiven roten Streifen. Es sollte sich nun ein neuer, vierseitiger Dialog mit unzähligen Funktionen öffnen. Keine Sorge, das sieht komplexer aus als es ist. Gehen wir kurz den Aufbau des Dialogs und dann die einzelnen Felder durch:



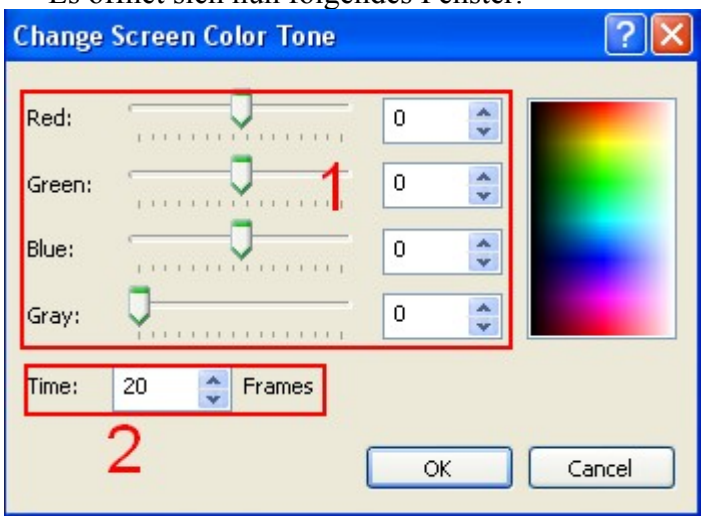
Die oberen, mit den Nummern 1-4 beschrifteten, Buttons (1) teilen den Dialog in 4 Seiten ein, auf die man durch Drücken dieser Buttons zugreifen kann. Die einzelnen Felder (z.B. 2) enthalten die jeweilige Bedingung. Dabei kann in jedem *Conditional Branch* nur eine Bedingung gestellt werden.
„Set handling when conditions do not apply“ ermöglicht euch das Einfügen eines Sonderfalls, wenn die gestellte Bedingung nicht erfüllt wird. Aber dazu später.
Seite 1 ist wohl der komplizierteste Teil des *Conditional Branchs*, denn hier erscheinen sofort wieder diese Variablen und *Switchs*, mit denen wir bisher ja noch nicht gearbeitet haben. Wir werden aber noch in diesem Tutorial darauf eingehen und einen *Conditional Branch* erstellen, der auch Variablen und *Switchs* abfragt.
Doch die zweite Seite ist dagegen schon sehr einleuchtend. Hier lassen sich Bedingungen zu euren Helden stellen. Bei „Hero“ wird der jeweilige Held eingestellt, dem ihr die Bedingung widmet. Wählen wir doch testweise unseren Barbarenhelden Alex aus.
Bei „In the party:“ wird abgefragt, ob der Held Alex überhaupt in der Heldentruppe ist. Wenn dies der Fall ist, so ist die Bedingung erfüllt. Ansonsten ist sie eben nicht erfüllt.
Bei „Name:“ wird abgefragt ob der gewählte Held einen bestimmten Namen trägt. Wir werden später noch Funktionen kennen lernen, die dem Spieler eine Namenseingabe ermöglichen. Nehmen wir einmal ein, für bauen einen Cheat ein, der ersten Helden eine starke Waffe gibt, wenn dieser den Namen „Cheater“ erhält. Um diesen speziellen Namen abzufragen, bedarf es einem *Conditional Branch* mit dem Inhalt Hero [0001:Alex] name is „Cheater“ applied.
Unter „Skill“ kann man abfragen lassen, ob der Held den ausgewählten *Skill* gelernt hat.
Durch „Weapons“ und „Armors“ lässt sich die getragene Ausrüstung des Helden in eine Bedingung stellen.
Letztlich bleibt noch „State“, wodurch der Status des Helden abgefragt wird. Beispielsweise könnte ein Held, der mit dem Status „Fluch“ belegt ist, von Hotelier mit den Worten „Wir lassen hier keine verfluchten Menschen übernachten!“ abgewiesen werden. Auch dies stellt wieder eine Fallunterscheidung dar.

Seite 3 bezieht sich auf die Gegner in einem Kampf, sowie die Events und die vom Spieler gesteuerte Spielfigur auf der Map.
Natürlich kann man einen *Conditional Branch*, der sich auf einen Gegner bezieht, nur während eines Kampfes stellen. Da wir jedoch in diesem Teil des Kurses noch nicht auf Kampfereignisse eingehen wollen, überspringen wir diesen Punkt einfach.
Bei „Character“ lassen sich Abfragen bezüglich der Blickrichtung, in die der ausgewählte Charakter blickt, stellen. Wollen wir also abfragen, ob der Held nach oben schaut, so müssen wir auf diese Funktion des *Conditional Branchs* zurückgreifen.
In der vierten Seite befinden sich die restlichen Abfragen. Bei Gold kann man eine Bedingung über die Geldmenge des Spielers stellen. „Item“, „Weapon“ und „Armor“ sind diesmal dazu da, um abzufragen, ob der Spieler einen bestimmten Gegenstand im Inventar hat.
Bei „Button“ wird abgefragt, ob eine bestimmte Taste gedrückt wird. Auch hierzu kommen wir in einem anderen Teil des Kurses noch. Der letzte Punkt „Script“ bezieht sich auf das im RPG-Maker XP integrierte Ruby. Dieses Tutorial bezieht sich nur auf die Grundlagen, deshalb werde ich auf Ruby nicht weiter eingehen.
Wir brauchen ohnehin nur den Punkt „Gold“. Hier geben wir die Zahl 20, sowie „or more“ (oder mehr) ein, und verlassen den Dialog mit einem OK. Beachtet zuvor, dass ihr ein Häkchen bei „Set handling when conditions do not apply“ gesetzt habt.
Unser Code sollte nun folgendermaßen aussehen:

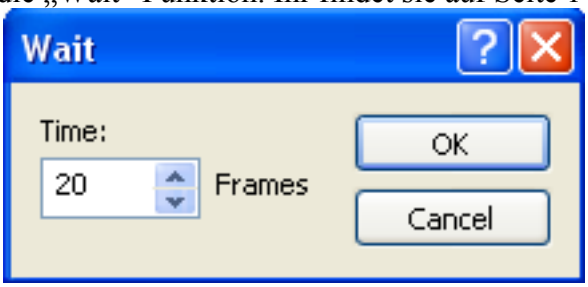
Wieder habe ich die einzelnen Strukturen mit verschiedenen Farben gekennzeichnet. Grün steht für die ganze Eventbefehlsliste. Violett hervorgehoben den Teil des Codes, der durch den *Show Choice* beansprucht wird. Die einzelnen Entscheidungsmöglichkeiten wurden in rot dargestellt. Der *Conditional Branch* nimmt den orangen Teil des Codes ein. Die türkisen Leisten kennzeichnen die möglichen Fälle der Bedingung. Wenn der Spieler 20 Gold oder mehr hat (also die Bedingung eintritt), so wird die obere türkise Leiste ausgeführt. Hat der Spieler weniger Gold, so wird die untere türkise Zeile unter dem „Else“ (ansonsten) ausgeführt.
In die zweite türkise Zeile fügen wir nun eine neue Textnachricht mit dem Inhalt
\c[3]Hotelier:\c[0] Ihr habt nicht genug Geld. Verschwindet!
ein. Dadurch wird der Spieler, sollte er die Übernachtung nicht bezahlen können, unfreundlich fortgejagt.

Oft gebrauchte Eventbefehle

Nun fehlt nur noch das eigentliche Übernachtungsscript. Wiederholen wir noch einmal den Ablauf: Es soll dunkel werden, eine Musik soll abgespielt werden, die Heldentruppe wird geheilt, der Bildschirm wird wieder hell, die Musik stoppt, stattdessen wird die alte Musik wieder abgespielt, der Spieler verliert 20 Goldmünzen.
Wir müssen nun nur noch diese ganzen Befehle als Eventfunktionen nacheinander einfügen.
Damit der Bildschirm schwarz wird, verwenden wir die Funktion „Change Screen Color tone“. Ihr findet sie auf der zweiten Seite, linke Spalte, drittes von unten.
Es öffnet sich nun folgendes Fenster:



Bei den vier Reglern (1) wird die Farbe eingestellt, die der Bildschirm annehmen soll. Wir drehen die ersten drei Regler auf -255. Dadurch wird der Bildschirm schwarz.
Das untere Feld „Time:“ (2) gibt an, wie lang der Übergang von der aktuellen Bildschirmfarbe zur neuen Bildschirmfarbe sein soll. Dieser Übergang wird in *Frames* angegeben. 20 *Frames* ergeben ungefähr eine Sekunde. Eine geringe Übergangszeit führt zu einem abrupten Farbwechsel. Eine lange erzeugt ein sanftes Übergehen in die neue Farbe. Wir geben 20 *Frames* ein.
Die nächste Funktion, der wir uns bedienen, ist die „Wait“ Funktion. Ihr findet sie auf Seite 1 über dem *Conditional Branch*.

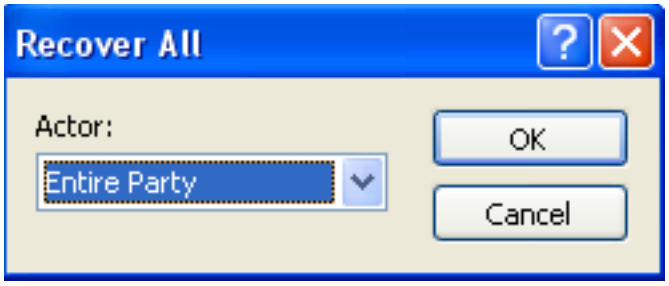


Die „Wait“ Funktion hält den Eventcode für eine kurze Weile an. Wie lange der Code gestoppt werden soll, wird in dem Feld „Time:“ eingegeben. Damit die Musik erst dann abgespielt wird, wenn der Bildschirm vollständig schwarz ist, geben wir 20 *Frames* ein. Der Bildschirm wird jetzt also schwarz gefärbt, und die nächsten Befehle werden erst dann ausgeführt, wenn der *Wait*, welcher genauso lange andauert wie die Färbung des Bildschirms, abgelaufen ist.
Für das Abspielen von Soundeffekten muss man die vier verschiedenen Arten von Audiodaten im Maker differenziert betrachten.

- BGM --> Backgroundmusic (Hintergrundsmusik) ist eine sich stets wiederholende Musik
- BGS --> Backgroundsounds (Hintergrundgeräusche) sind Geräusche die sich stets wiederholen (z.B. Das Pfeifen des Windes oder die Wellengeräusche in einer Hafenstadt)
- ME --> Musiceffects (Musikeffekte) sind Musikstücke, die nur einmal zu einer besonderen Situation abgespielt werden. Während ein Musikeffekt abgespielt wird, wird die Hintergrundmusik pausiert. (z.B. Die Musik, die beim Gewinnen eines Kampfes abgespielt wird)
- SE --> Soundeffects (Geräuscheffekte) sind Geräusche, die nur einmal zu einer besonderen Situation abgespielt werden (z.B. Das Geräusch eines Feuerzaubers während des Kampfes)

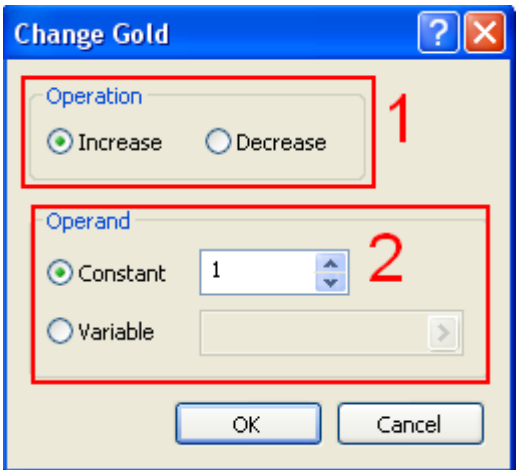
Die ersten beiden Audioarten werden normalerweise ständig im Hintergrund abgespielt. Die letzten beiden nur zu besonderen Situationen. Unser Übernachten wäre z.B. eine solche Situation. Es handelt sich also um einen ME (Musikeffekt).
Für die Audioeffekte sind die Eventbefehle auf der rechten Spalte auf Seite 2 verantwortlich. „Play BGM“, „Play BGS“, „Play ME“ sowie „Play SE“ sind für das Abspielen solcher Audiodateien zuständig. Mit „Fade Out BGM“ und „Fade Out BGS“ wird ein(e) Hintergrundsmusik/Geräusch ausgeblendet (in dem es immer leiser wird). Soundeffekte werden per „Stop SE“ abrupt abgebrochen. Durch „Memorize BGM/BGS“ merkt sich der Maker, welche(s) Hintergrundsmusik/Geräusch

gerade abgespielt wird. Wird danach „Restore BGM/BGS“ benutzt, so wird die zuvor gemerkte Musik/Geräusch abgespielt.
In unserem Inn reicht es, wenn wir die Funktion „Play ME“ wählen, und in der dortigen Liste den Musikeffekt „014-Inn01“ wählen.
Damit der Code noch einmal pausiert, während der Musikeffekt gespielt wird, setzen wir einen neuen *Wait*. Diesmal jedoch mit 60 *Frames*.
Danach wird der Bildschirm wieder auf den normalen Farbton gesetzt. Erneut benutzen wir dafür die „Change Screen Color Tone“ Funktion, setzen aber nun die oberen drei Regler auf 0.
Hinterher wieder ein *Wait* mit 20 *Frames*.



Zu guter Letzt noch der eigentliche Sinn der Übernachtung: Die Heilung der Heldentruppe. Dies erreicht ihr mit dem Eventbefehl „Recover All“, welchen ihr auf der dritten Seite in der linken Spalte findet. Bei „Actor“ könnt ihr auswählen, welcher Held geheilt werden soll. „Entire Party“ heilt die komplette Heldentruppe, weswegen wir dies auch auswählen.

Nun hätten wir fast das Bezahlen der Übernachtung vergessen. Der Spieler muss ja noch um 20 Goldmünzen erleichtert werden. Dies geschieht über die Funktion „Change Gold“. Sie befindet sich auf der ersten Seite in der rechten Spalte. Es öffnet sich nun ein neues Dialogfenster.



Bei „Operation“ (1) wird eingestellt, ob die Heldentruppe Geld verlieren (decrease), oder gewinnen (increase) soll.
Wir wählen natürlich „Decrease“ aus, denn unser Held muss schließlich 20 Münzen bezahlen.
Die Anzahl des Geldes wird bei „Operand“ festgelegt. Hierbei gibt es zwei Möglichkeiten: Eine feste Zahl eingeben (bei Constant), oder den Wert einer Variable übernehmen. Nunja, wie schon etliche Male erwähnt: Zu den Variablen kommen wir noch.
Wählt also Constant aus, und gebt die Zahl 20 ein.
Nun verliert der Spieler 20 Goldmünzen.

Damit wäre unser Inn komplett. Hier noch einmal der komplette Code.

```
@>Text: \c[3]Hotelier: \c[0]Hallo \n[9], möchten \ sie ein Zimmer
 :       : für 20 Münzen? \g
@>Show Choices: Ja, gerne, Nein, danke
 : When [Ja, gerne]
  @>Conditional Branch: Gold 20 or more
    @>Change Screen Color Tone: (-255,-255,-255,0), @20
    @>Wait: 20 frame(s)
    @>Memorize BGM/BGS
    @>Play ME: '014-Inn01', 100, 100
    @>Wait: 60 frame(s)
    @>Change Screen Color Tone: (0,0,0,0), @20
    @>Wait: 20 frame(s)
    @>Recover All: Entire Party
    @>Change Gold: - 20
    @>
   : Else
    @>Text: \c[3]Hotelier:\c[0] Ihr habt nicht genug Geld.
     :       : Verschwindet!
    @>
   : Branch End
  @>
 : When [Nein, danke]
  @>Text: \c[3]Hotelier: \c[0]Das ist aber schade... Auf
   :       : Wiedersehen!
  @>
 : Branch End
```

Außerdem habe ich das Script noch einmal in einer Strukturskizze zusammengefasst, um den Ablauf des Events zu verdeutlichen:

<table>
<tr><td colspan="3">Frage ob Spieler übernachten will</td></tr>
<tr><td colspan="2">Ja, gerne</td><td>Nein, danke</td></tr>
<tr><td colspan="2">Wenn mindestens 20 Gold</td><td rowspan="2">Verabschieden</td></tr>
<tr><td>Bildschirm verdunkeln
Musik abspielen
Bildschirm aufhellen
20 Gold abziehen
vollständig heilen</td><td>Spieler fortjagen</td></tr>
</table>

Unser Inn ist nun fertig. Probiert es einmal aus. Sollte es nicht funktionieren, so vergleicht euren Code mit dem hier Abgebildeten und korrigiert den Euren.
Als nächstes werden wir einen Itemshop erstellen. Keine Sorge, diesmal werden wir gar nicht erst auf Bedingungen, Entscheidungen und diversem zurückgreifen müssen. Denn der Maker bietet bereits eine fertige Option für Itemläden: „Shop Processing“. Ihr findet diese Funktion auf der dritten Seite in der linken Spalte weit oben. Erstellt ein neues Event, gibt ihm eine passende Grafik, und fügt diese „Shop Processing“ Funktion in seiner Eventbefehlsliste ein.
Beim anklicken der Funktion öffnet sich folgender Dialog:

Das Erstellen eines Verkäufers ist relativ leicht. Klickt doppelt auf die erste Zeile der weißen, leeren Tabelle unter „Goods“ (Waren) und Price (Kosten).
Es öffnet sich nun ein neues Fenster:

Ihr könnt nun aus drei Listen ein beliebiges Objekt auswählen. Unter Items befinden sich alle Gegenstände, die in der Datenbank unter der Kategorie Items erstellt wurden. Bei „Weapons“ und „Armors“ befinden sich die Waffen und Rüstungen, welche in der Datenbank in den Kategorien „Weapon“ und „Armor“ konfiguriert werden.

Ihr wählt das Objekt aus, das der Verkäufer anbieten soll und klickt auf OK. Danach könnt ihr die nächste leere Zeile durch ein Objekt füllen, so dass euer *Shop Processing* Fenster bald nicht mehr so leer aussieht.
Links seht ihr nun immer die Waren abgebildet, rechts die Kosten der jeweiligen Güter. Wie ihr die Preise verändert, haben wir im vorherigen Tutorial ja bereits gelernt. Hierfür müssen lediglich die Einstellungen in der Datenbank verändert werden.
Nachdem ihr die Funktion ausgefüllt habt, seid ihr mit dem Erstellen eines Verkäufers auch schon fertig.
Ihr seht: Nicht immer ist soviel Aufwand wie für unser Inn erforderlich. Doch unser Gasthaus hatte einen ganz anderen Vorteil: Ihr habt die Anwendung von „Conditional Branch“ und „Show Choice“ erlernt. Diese Funktionen werden wir nun beim Erstellen eines *Quests* verinnerlichen.

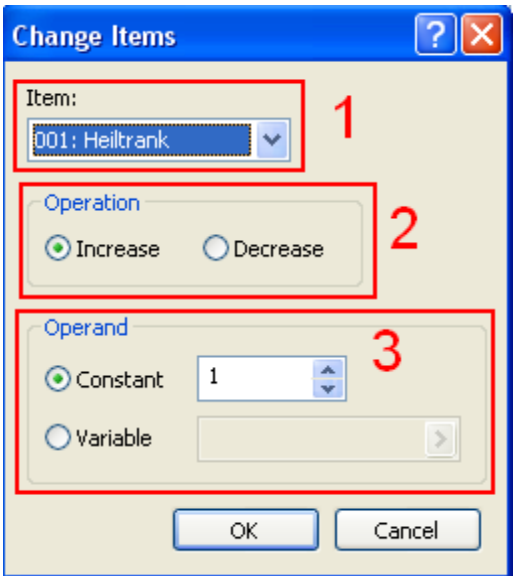
Aufgaben

Vorbereitung der Aufgabe

Aufgaben, oder wie oft auch genannt: *Quests*, werden das Thema dieses Kapitels sein. Was wäre ein Dorf ohne NPCs, die den Helden bitten die arme Katze vom Baum zu holen, oder den kleinen Buben vor dem bösen Drachen zu retten.
Wir beginnen vorerst mit einem eher einfachen *Quest* für unseren Barbarenhelden:
In der Stadtapotheke hat der vergessliche Apotheker seine Heiltränke in alle möglichen Regale gestopft. Dumm, das er sie nun nicht wieder findet. Also muss unser Held ihm helfen. 5 Heiltränke sind in der Apotheke versteckt. Erst wenn wir alle fünf Heiltränke dem Apotheker geben, kann dieser sein Geschäft weiterführen. Zum Dank wird er uns 2 Heiltränke abgeben. Außerdem wird er uns seine Waren anbieten. Na, solch ein Angebot kann man doch nicht ablehnen, oder?

Hier ein Screenshot von meiner Apotheke. Natürlich könnt ihr eure ganz anders mappen.
Die rot eingerahmten Felder habe ich mit einem Grafikprogramm zusätzlich hinzugefügt. Sie kennzeichnen die Positionen, an denen wir Events platzieren müssen.
Die beiden blauen Felder unten stehen für die zwei Teleportevents, welche aus der Apotheke herausführen. Das türkise Feld deutet die Position des verzweifelten Apothekers an.
Die grünen Felder sind die Verstecke für die Heiltränke.
Beginnen wir zuerst mit den Heiltränken. Sobald der Spieler ein Heiltrankversteck anklickt, soll eine Nachricht erscheinen, dass ein Heiltrank gefunden wird. Danach soll dem Inventar ein Heiltrank hinzugefügt werden. Schließlich muss das Event irgendwie verschwinden. Denn sonst könnte man es ja unendlich oft anklicken, und so eine Unzahl von Heiltränken anhäufen.

Zum Anzeigen einer Textnachricht verwenden wir die bereits bekannte „Show Text“ Funktion. Das Hinzufügen eines neuen Items ähnelt in seiner Ausführung stark dem Ändern des Geldbetrages der Truppe. Die Funktion hierfür heißt „Change Items“. Sie befindet sich auf der ersten Seite in der rechten Spalte, direkt unter der „Change Gold“ Funktion.

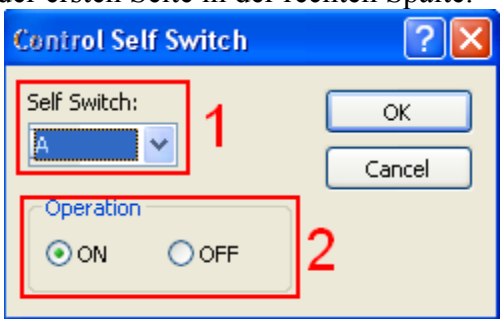


Der Aufbau des Fensters ähnelt ebenfalls der „Change Gold“ Funktion. Bei Item (1) wird das jeweilige Item, das hinzugefügt oder abgenommen werden soll. Falls ihr das Item „Heiltrank“, welches wir im ersten Kurs erstellt haben, nicht mehr habt, so verwendet stattdessen das Item „Potion“.
Bei Operation (2) stellt ihr ein, ob das Item hinzugefügt (increase) oder abgenommen (decrease) werden soll. Bei Operand (3) wird die Menge der Items eingestellt. Entweder geschieht dies wieder über einen konstanten Wert, oder einer Variable. Wir wählen den konstanten Wert 1.
Schließlich kommen wir zum eigentlichen Problem: Wie sagen wir einem Event, das es nach einmaliger Anwendung endgültig verschwinden soll?

Lokale Schalter als Informationsträger

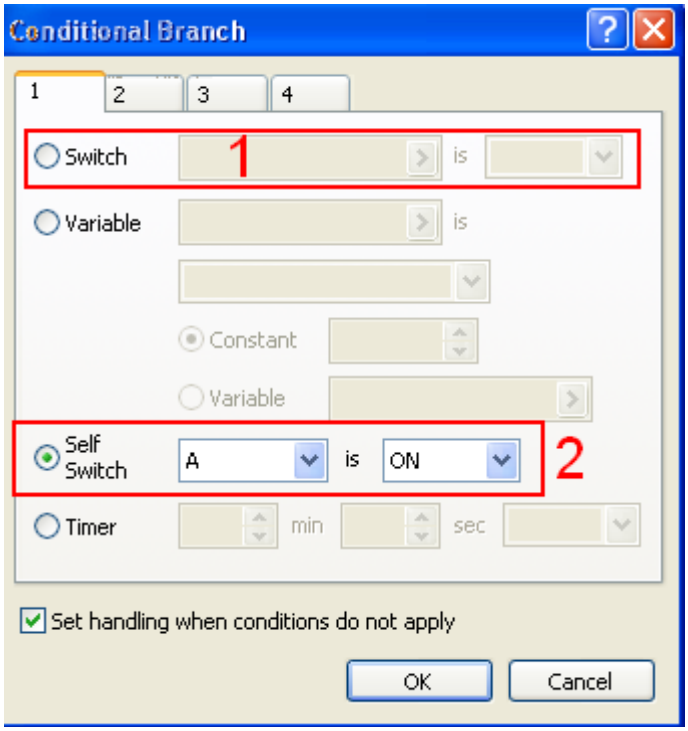
Nun, hierfür werden wir uns den **SelfSwitchs** (lokale Schalter) bedienen. Erst einmal: was ist ein Switch, bzw. ein Schalter, wie wir ihn vorerst nennen wollen? Ein Schalter dient der Speicherung von Informationen. Hierbei kann ein Schalter genau zwei Informationen speichern: ON (an) und OFF (aus). Gehen wir einmal von einem Lichtschalter aus. Der Lichtschalter kann an- und ausgeschaltet sein. Er merkt sich hierbei die Information, ob Licht scheinen darf, oder nicht.
In unserem Fall müssen wir dem Maker also sagen: „Merke dir, das wenn der Spieler den Heiltrank genommen hat, er diesen nicht noch einmal nehmen darf!“
Im Maker unterscheiden wir zwischen den **globalen Schaltern** (einfach Schalter bzw. Switch genannt) und den **lokalen Schaltern** (auf Englisch: SelfSwitchs).
Globale Schalter geben ihre Information an alle Instanzen des Makers weiter. Lokale Schalter dagegen, geben ihre Information nur an das Event weiter, zu dem sie gehören.
Gehen wir wieder einmal von dem Lichtschalter aus. Ein Lichtschalter gehört zu dem Event „Lampe“. Nur die Lampe darf erfahren, ob ihr Lichtschalter auf AN oder AUS gestellt ist. Der Kühlschrank ein Stockwerk tiefer kann die Lampe so oft bedrängen wie er will, er wird nicht in Erfahrung bringen ob der Lichtschalter AN oder AUS ist, denn der Lichtschalter ist ein lokaler Schalter. Er gibt seine Information nur an die Lampe weiter.
In unserem Fall hat also jeder Trank einen lokalen Schalter, der festhält, ob der Trank genommen wurde, oder ob er noch herumliegt. Dabei darf immer nur das jeweilige Trankevent wissen, ob ihr Schalter auf AN oder AUS gestellt ist.
Aber was bringen diese lokalen Schalter dann? Warum nicht gleich globale Schalter nehmen?
Um diese Frage zu beantworten, müssen wir uns etwas näher mit lokalen Schaltern beschäftigen.
Jedes Event hat genau vier lokale Schalter: A, B, C und D. Diese vier Schalter heißen in jedem Event gleich. Die Lampe kann also einen Lichtschalter namens A haben, der ihr sagt, ob das Licht AN oder AUS ist. Der Kühlschrank dagegen hat ebenfalls einen Schalter A. Dieser sagt jedoch, ob jemand die Kühlschranktür offen gelassen hat.
Obwohl beide Schalter die gleichen Namen tragen, sind sie doch grundverschieden. Würde der Kühlschrank den Schalter A der Lampe sehen dürfen, wie könnte er dann diesen Schalter A von seinem Eigenen unterscheiden?
Im Gegenzug sind globale Schalter für jedes Event gültig. Der Strom wäre zum Beispiel ein globaler Schalter. Ist er AUS, so läuft sowohl der Kühlschrank, als auch die Lampe nicht mehr.

Beide müssen ständig wissen, ob der Strom AN oder AUS ist. Da dieser Schalter also für mehrere Events gültig ist, muss er global sein.
Doch beziehen wir uns lieber auf unser *Quest*. Unsere Tränke verwenden lokale Schalter. Um einen lokalen Schalter umzustellen, verwenden wir den Befehl „Control SelfSwitch“. Er befindet sich auf der ersten Seite in der rechten Spalte.



Das Fenster ist eigentlich recht einfach aufgebaut. Bei „Self Switch“ (1) wählt man den lokalen Schalter aus. Wie gesagt, gibt es vier verschiedene: A, B, C und D.
Wir nehmen einfach den Schalter A.
Bei Operation (2) wird schließlich noch eingestellt, ob der *Switch* auf ON (an) oder OFF (aus)gestellt werden soll. Wir setzen ihn auf ON.

Doch wie kann dieser Schalter es bewirken, dass unser Event nicht erneut abgespielt wird? Nun, ein Schalter ist nur eine Information. Der Maker merkt sich, dass der Trank bereits genommen wurde. Wie kann man Informationen abfragen? Richtig, mit Bedingungen. Wir werden also unser Trankevent in einen *Conditional Branch* einfügen.



Diesmal werden wir die erste Seite des *Conditional Branchs* verwenden. Hier stehen gleich zweimal die Wörter *Switch*.
Einmal oben (1), was sich auf globale Schalter bezieht. Wir aber haben einen Lokalen verwenden. Deshalb nutzen wir das Feld weiter unten wo „Self Switch“ steht (2). Dort wählen wir unseren Schalter A aus. Schließlich müssen wir noch einstellen, ob dieser ON oder OFF sein soll. Wählt OFF aus. Beachtet, das unten ein Häkchen bei „Set handling when conditions do not apply“ gesetzt wurde.

Nun fügen wir unseren bisherigen Code in den *Conditional Branch* ein. Eine Textnachricht mit dem Inhalt
Hier ist nichts mehr...
wird unter den Else (Ansonsten) gesetzt.

Unser Code sollte nun folgendermaßen aussehen:

```
@>Conditional Branch: Self Switch A == OFF
  @>Text: Du findest einen Heiltrank!
  @>Change Items: [Heiltrank], + 1
  @>Control Self Switch: A =ON
  @>
 :  Else
  @>Text: Hier ist nichts mehr
  @>
 :  Branch End
@>
```

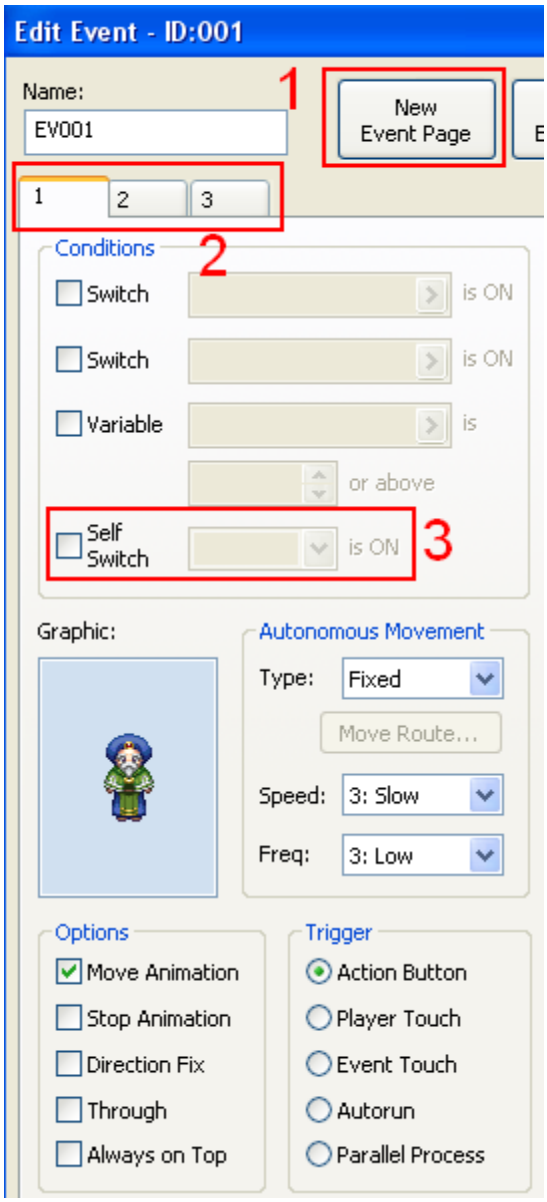
Nun testet euer Projekt. Klickt das Heiltrankversteck an. Beim ersten Mal werdet ihr einen Heiltrank erhalten. Beim zweiten Mal dagegen wird die Nachricht „Hier ist nichts mehr“ eingeblendet.
Hier noch eine Skizze, die den Ablauf erläutert:

Durchlauf	***Was passiert?***	***Switch A***
1	Weil A OFF ist, wird Item hinzugefügt, eine Nachricht eingeblendet und A auf ON gesetzt	OFF
2	Weil A ON ist, wird nur eine Nachricht eingeblendet.	ON

Bei unserem nächsten Schritt, werdet ihr den wahren Vorteil von lokalen Schaltern kennen lernen.

Klickt euer neues Heiltrankversteckevent mit der rechten Maustaste an. Nun öffnet sich eine Liste mit verschiedenen Standardbefehlen, welche das Editieren, Löschen und Kopieren des Events, sowie da das Setzen einer Heldenstartposition ermöglichen. Klickt nun auf „Copy“ (Kopieren).
Nun klickt ihr im *Eventlayer* (Eventebene) auf die verschiedenen Verstecke und kopiert euer Heiltrankversteckevent dort hinein, indem ihr wieder die rechte Maustaste klickt und diesmal „Paste“ (Einfügen) auswählt.
Ihr könnt auf diese Weise euer Heiltrankversteck vermehren, statt stets ein neues Event zu erstellen. Das tolle daran ist, das ihr die Schalter nie ändern müsst. Wären es globale Schalter, so müsstet ihr den Namen des Schalters für jedes Ereignis ändern. Das eine Versteck müsste dann den Schalter „Versteck1“ besitzen, das zweite den Schalter „Versteck2“ etc.
Bei lokalen Schaltern ist dies nicht der Fall. Jedes Versteck hat äußerlich denselben Schalter: A. Doch intern sind es völlig verschiedene Schalter, die stets nur im jeweiligen Versteck wirken.
Das Erstellen der Verstecke ging recht schnell. Auch unser Apotheker wird nicht sonderlich schwer. Diesmal werden wir zusätzlich der *SelfSwitchs* auch auf **Eventseiten** eingehen.

Mehrseitige Events



Auch dies wird nicht sonderlich kompliziert sein. Ihr müsst wissen, dass ein Event mehrere Seiten haben kann. Stets wird aber nur eine Seite ausgeführt, während die anderen Seiten inaktiv sind. Als Beispielsweise eine hübsche Frau, die sich plötzlich in einen Dämonen verwandelt. Auf Seite 1 ist sie noch eine Jungfrau, doch sobald man sie stört, wird Seite 2 aktiviert, und sie wird zu einem gefährlichen Dämonen. Keine Sorge, unser Apotheker soll kein Dämon werden. Doch auch hier werden drei verschiedene Seiten zum Einsatz kommen:
Seite 1: Der Apotheker schildert uns sein Problem. Danach wird Seite 2 aktiviert.
Seite 2: Der Apotheker überprüft, ob wir seine Heiltränke schon gefunden haben. Ist dies der Fall, so wird der Spieler belohnt und Seite 3 aktiviert, ansonsten meckert er den Spieler ungeduldig an.
Seite 3: Der Apotheker begrüßt seinen Helden freundlich, und bietet ihm seine Waren an.
Nun bleibt wohl nur noch die Frage, wie eine Seite aktiviert und deaktiviert wird. Ganz einfach: Über Informationen. Und eine der wichtigsten Informationsträger haben wir ja eben kennen gelernt. Die lokalen Schalter.
Erstellt also ein neues Event für den Apotheker und gebt ihm die Grafik „130-Noble05“.
Nun klickt auf den Button „New Event Page“ (neue Ereignisseite), welcher mit der Nummer 1 gekennzeichnet ist.
Sobald wird unter dem Eventnamen eine neue Seite angezeigt (2). Erstellt eine weitere Seite, so dass ihr ein dreiseitiges Event habt. Auf den anderen beiden Seiten setzt ihr dieselben Einstellungen (unter anderem auch dieselbe Grafik).
Falls ihr euch fragt was die anderen Buttons neben „New Event Page“ bedeuten, hier eine kurze Erklärung:
„Copy Event Page“ kopiert ein bestehende Eventseite.
„Paste Event Page“ erstellt eine neue Eventseite, welche genau dieselben Einstellungen enthält wie die zuvor kopierte Eventseite.
„Delete Event Page“ entfernt eine Seite wieder.
„Clear Event Page“ setzt alle Einstellungen der Seite auf standard.

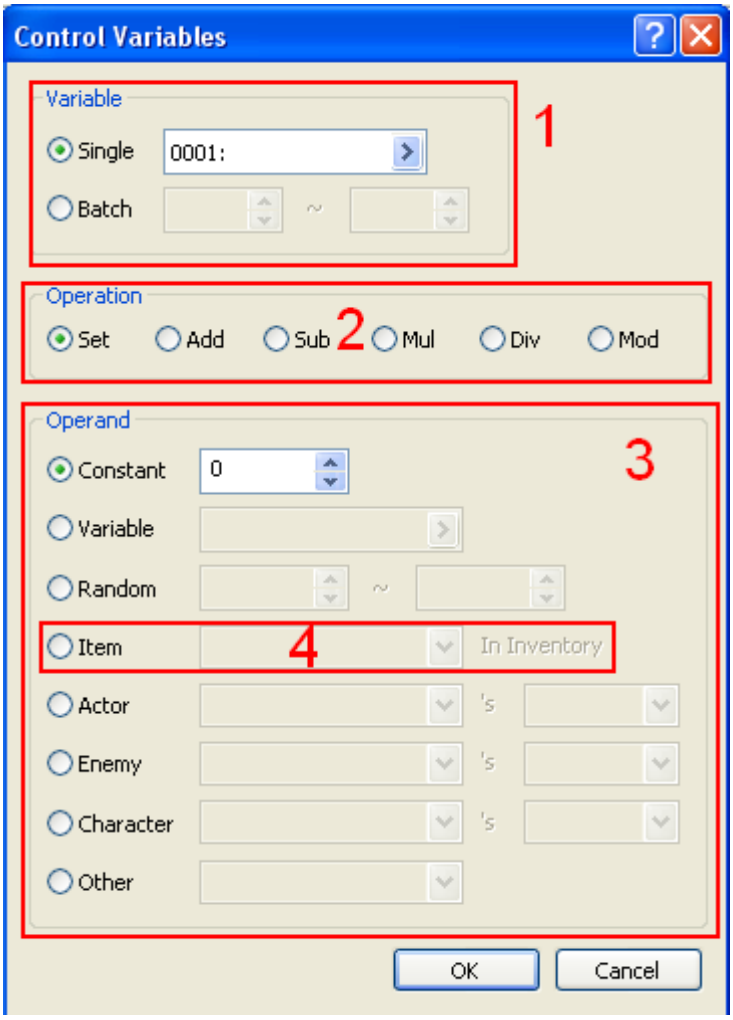
Als nächstes betrachten wir uns das große Feld „Conditions“ (Bedingungen) näher. Hier wird eingestellt, welche Bedingungen erfüllt sein müssen, damit eine Seite aktiviert wird. Grundsätzlich gilt: Wenn mehrere Seiten existieren, welche alle die Bedingungen erfüllen, so wird stets die Seite mit höherer Seitennummer ausgeführt.
Aus diesem Grund brauchen wir für die Erste Seite auch keine Bedingung.

Die erste Seite wird immer dann ausgeführt, wenn die anderen beiden Seiten nicht die Bedingungen erfüllen.
In die Eventbefehlsliste der ersten Seite fügen wir nun den Show Text ein:
\c[5]Apotheker: \c[0]Seid gegrüßt! Ich finde in dieser Unordnung meine fünf Heiltränke nicht. Was für ein Unglück... ob ihr sie vielleicht auftreiben könnt?
Nachdem der Apotheker dies sagt, soll die zweite Seite aktiviert werden. Hierfür müssen wir, wie bereits angedeutet, einen lokalen Schalter aktivieren. Wählt also die Funktion „Control Self Switch", wählt den lokalen Schalter A und setzt diesen auf ON.
Nun suchen wir die zweite Seite auf. Hier setzen wir als Bedingung, dass der lokale Schalter A auf ON steht. Diese Bedingung setzen wir in das Feld „Self Switch" (3) in „Conditions".
Die zweite Seite wird also erst dann ausgeführt, wenn die erste zuvor aktiviert wurde. Nun, das ist doch ähnlich wie zuvor unsere Heiltränke... Tatsächlich sind sich Conditions und *Conditional Branchs* gar nicht so unähnlich. Erstere aktivieren eine Seite eines Eventcodes, letztere aktivieren einen Abschnitt eines Eventcodes.
Doch nun stehen wir einem neuen Problem gegenüber: Wir müssen abfragen, ob der Spieler fünf Heiltränke im Inventar hat. Nun, das ließe sich mit *Conditional Branchs* doch leicht erledigen. Oder...? Leider kann man mit *Conditional Branchs* zwar abfragen, ob man ein bestimmtes Item besitzt, doch man kann nicht abfragen ob man eine bestimmte Menge an Items besitzt.
Wir müssen also die Menge unserer Heiltränke auf einen Informationsträger speichern, welcher vom *Conditional Branch* abgefragt werden kann. Doch unsere Schalter kommen dafür nicht in Frage. Denn die können nur AN und AUS annehmen. Wir bräuchten eine Art Schalter, der ganze Zahlenwerte annehmen kann. Und hier kommen die **Variablen** ins Spiel.

Variablen als Informationsträger

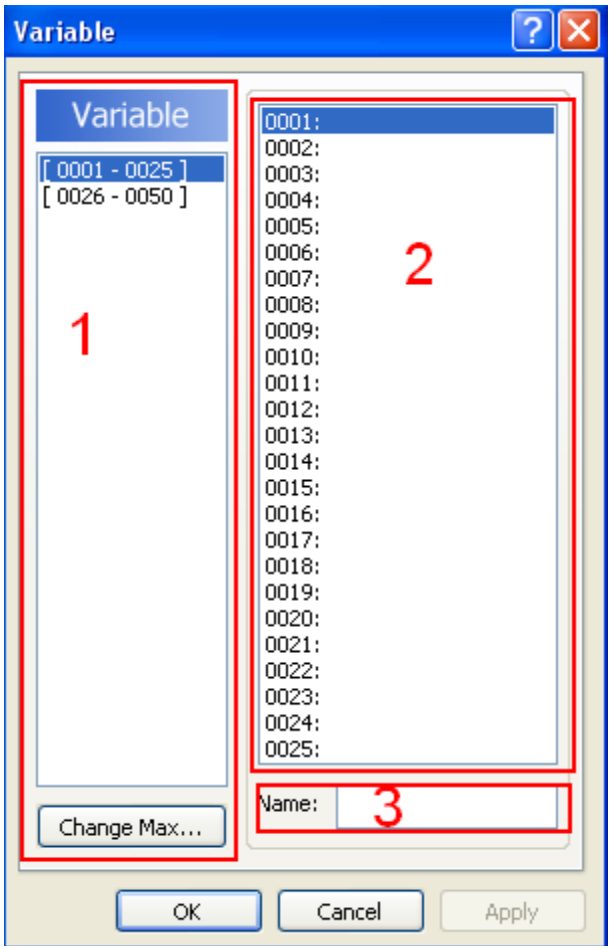
Nun, wer Variablen bereits aus der Mathematik kennt, wird es hier sicherlich einfacher haben. Doch ich werde versuchen es auch für jene verständlich zu machen, die das Wort Variable zuvor noch nie gehört haben.
In der Mathematik tauchen Variablen in Form von Buchstaben auf, die in einer Formel stehen.
z.B. $y = 3 + x$
Im Maker können die Variablen richtige Namen und Nummern erhalten, ebenso wie globale Schalter. Variablen sind auch Informationsträger. Nur während ein Schalter die Information AN und AUS speichert, merkt sich eine Variable einen bestimmten Zahlenwert.
Wenn ich also sage Variable „ZAHL" = 5, dann merkt sich der Maker das diese Variable den Wert 5 enthält. Ich kann diesen Wert jederzeit wieder abfragen. Und zwar von überall aus. Denn Variablen sind grundsätzlich global. Lokale Variablen gibt es nicht.
Der nächste Vorteil von Variablen ist, dass sie den Wert von vielen Informationen des Makers annehmen können. So kann man dem Maker beispielsweise sagen, er soll die Lebenspunkte des Helden Alex in die Variable „Leben" speichern.
Des Weiteren können wir mit Variablen rechnen. Ihr könnt sie addieren, subtrahieren, multiplizieren usw.
Ihr seht, Variablen haben ein gewaltiges Potenzial. In diesem Teil des Tutorials werden wir uns nur oberflächlich mit ihnen beschäftigen. Doch in den nächsten Kursen werden wir komplexe Scripts behandeln, welche nur durch Variablen funktionieren.
Doch möchte ich euch nicht abschrecken. Sehen wir uns doch einmal die Anwendung von Variablen an.
Variablen werden über die Funktion „Control Variables" gesteuert. Diese befindet sich auf der ersten Seite, rechten Spalte, direkt über „Control SelfSwitch".

Das folgende Fenster wird sich nun öffnen:

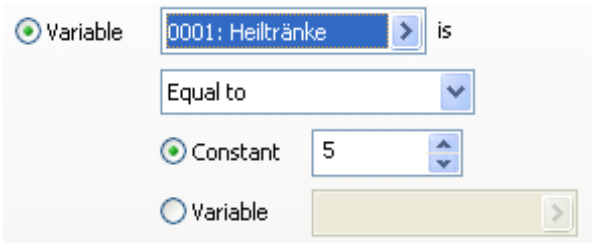


Oben bei „Variable“ (1) könnt ihr die Variable wählen, welche ihr verwenden wollt. Mit „Single“ (Einzeln) wählt ihr eine einzige. Doch ihr könnt auch mit mehreren Variablen auf einmal rechnen. Hierfür müsstet ihr „Batch“ auswählen. Doch gewöhnlicherweise greift man stets nur auf eine Variable zu.
Bei „Operation“ (2) wird ausgewählt, welche Rechenoperation mit der Variable ausgeführt wird.
Mit Set gibt man einer Variable einen neuen Operationswert.
Mit „add“ addiert man den momentanen Wert einer Variable mit einem neuen Operationswert.
„Sub“ steht für subtrahieren, also für das Abziehen eines neuen Wertes vom momentanen.
„Mul“ heißt multiplizieren. Hierbei wird der momentane Wert der Variable, mit dem neuen Operationswert multipliziert.
„Div“ ist die Abkürzung von Dividieren. Der momentane Wert wird durch den neuen Operationswert geteilt.
„Mod“ steht für Modulo, eine Rechenoperation die viele wohl nur noch aus der Grundschule kennen werden: Das Teilen mit Rest. Hierbei nimmt die Variable den Restwert an, der übrig bleibt, wenn man den momentanen Wert durch den neuen Operationswert teilt.
Genaue Beispiele für die einzelnen Rechenoperationen werden in den kommenden Tutorials zu Genüge vorhanden sein.
Das dritte große Fenster trägt die Bezeichnung „Operand“ (3). Hier wird der neue Operationswert eingestellt, mit dem die Variable rechnen soll.
Bei „Constant“ kann direkt eine Zahl eingestellt werden. Will man zum Beispiel, das die Variable „Name“ den Wert 8 annimmt, so muss man bei Variable Single die Variable „Name“ auswählen, bei „Operation“ das Set anwählen und schließlich bei „Operand“ in dem Feld neben „Constant“ die Zahl 8 eingeben.
Bei „Variable“ könnt ihr eine Variable als neuen Wert eingeben. Will man also zwei Variablen addieren, so muss man als Operation „add“ auswählen, und bei Operand auf „Variable“ klicken.
„Random“ ist ein Zufallsgenerator. Er nimmt als Operationswert eine zufällige Zahl im Bereich der beiden neben „Random“ eingegebenen Zahlenwerte. Wollt ihr also ein Kasino erstellen, so ist ein solcher Zufallsgenerator von großer Wichtigkeit. Wenn es soweit ist, werden wir auf den *Random* Befehl zurückkommen.
Bei Item, „Actor“, „Enemy“, „Character“ und „Other“ könnt ihr eine bestimmte Information aus dem Maker (beispielsweise die aktuellen Lebenspunkte des Helden Alex) als Operationswert der Variable bestimmen. Eigentlich sind die einzelnen Informationen/Parameter selbsterklärend. In den künftigen Scripts werde ich versuchen, jeden Parameter zumindestens einmal näher zu erläutern.
In diesem Fall ist es wohl der Parameter Item, welcher besondere Beachtung verdient.
Denn schließlich wollen wir doch die Anzahl unserer Heiltränke in eine Variable speichern.

Klickt hierfür zuerst doppelt auf das Feld bei „Variable“ neben Single.



Es erscheint nun folgender Dialog:
Der Dialog erinnert entfernt an die Kategorien in der Datenbank. Rechts befindet sich wieder eine Liste mit verschiedenen Variablenbereichen. Über „Change Max“ könnt ihr die maximale Anzahl eurer Variablen erhöhen. Macht dies nur, wenn alle bisherigen Variablen aufgebraucht sind.
In der rechten Spalte (2) sind die einzelnen Variablen. Jede Variable hat grundsätzlich eine ID, also eine Identifikationsnummer. Der Maker unterscheidet Variablen nur anhand ihrer ID. Der Name der Variable ist für den Maker also völlig unwichtig. Dennoch solltet ihr stets einen aussagekräftigen Namen wählen, um die Übersicht zu bessern.
Wählt also eine beliebige Variable aus der Liste aus (am besten ihr fangt mit der 1 an). In dem unteren Feld „Name:“ (3) könnt ihr der Variable einen Namen geben.
Wählt den Namen „Heiltränke“ und klickt auf OK. Anfangs enthalten alle Variablen in eurer Liste den Wert 0.
Wir wollen unserer Variable „Heiltränke“ nun einen neuen Wert zuweisen. Dies geschieht über die Rechenoperation Set. Wählt bei „Operation“ also Set aus.
Bei „Operand“ klicken wir nun das Feld Item an, und wählen in der Liste daneben das Item Heiltrank aus. Durch diese Funktion nimmt unsere Variable „Heiltrank“ automatisch den Wert der Menge der Heiltränke im Inventar an.
Hat der Held alle Heiltränke in der Apotheke gefunden, so wird die Variable „Heiltränke“ also den Wert 5 annehmen.
Also zurück zu unserem Apotheker. Auf der zweiten Seite fügen wir also die Funktion „Control Variables“ ein, welche die Variable „Heiltränke“ auf die Menge der Heiltränke im Inventar einstellt.
Nun können wir durch einen *Conditional Branch* auch problemlos abfragen, wie groß die Variable Heiltränke (und damit die Menge der Heiltränke im Inventar) ist.
Setzt also einen *Conditional Branch*, wählt dort auf Seite 1 das Feld „Variable“.



In dem ersten Feld neben „is“ sucht ihr euch eure Variable „Heiltrank“ heraus. In dem darunterliegenden Feld könnt ihr folgende Optionen auswählen:
„equal to“ (gleich), fragt ab, ob die Variable „Heiltrank“ genauso groß wie der Vergleichswert ist
„greater than or equal to“ (größer gleich), fragt ab, ob die Variable genauso groß, oder sogar größer als der Vergleichswert ist.
„Less than or equal to“ (kleiner gleich) fragt wiederum ab, ob unsere Variable genauso groß, oder kleiner als der Vergleichswert ist.
Bei „Greather than“ (größer als) ist die Bedingung dann erfüllt, wenn die Variable größer als der Vergleichswert ist. „Less than“ (kleiner als) fragt ab, ob die Variable kleiner als der Vergleichswert ist.
Zum Schluss gibt es noch „Other“ (ungleich). Hier ist die Bedingung nur dann erfüllt, wenn die Variable anders ist als der Vergleichswert.

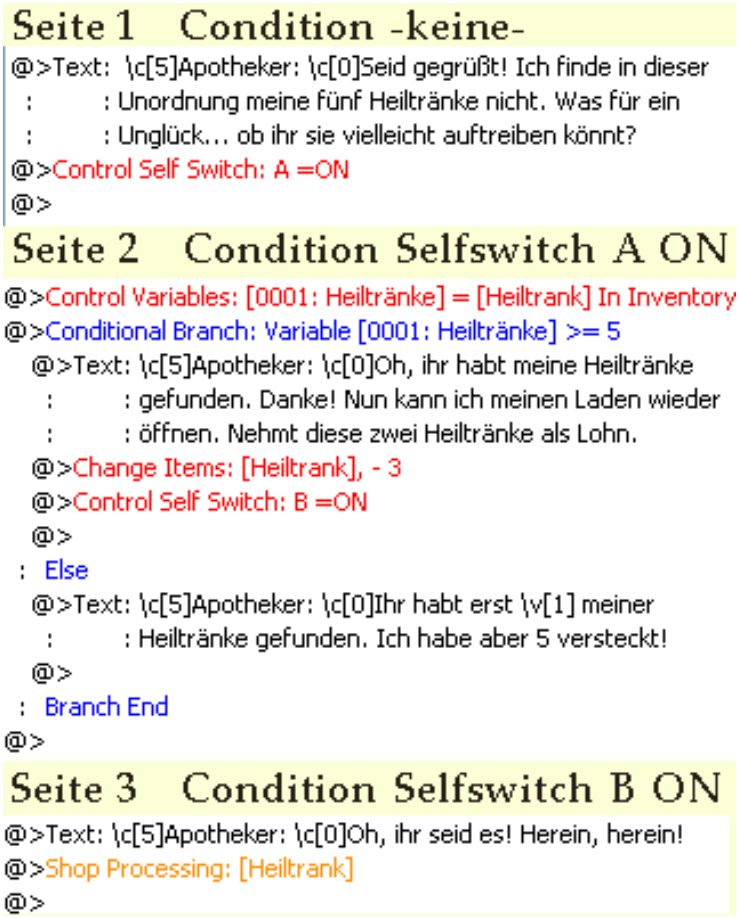
Wir wollen abfragen, ob der Spieler mindestens 5 Heiltränke gefunden hat. Hierfür verwenden wir also „Greater than or equal to“. Warum nicht nur „equal“? Weil dann die Bedingung auch dann nicht eintreten würde, wenn der Spieler mehr Heiltränke im Inventar hat (wenn er zum Beispiel in einem anderen Laden bereits einen Heiltrank gekauft hat).
Die letzten zwei Felder geben den Vergleichswert an. Entweder wählt man bei „Constant“ eine konstante Zahl, oder man wählt bei „Variable“ eine Variable als Vergleichswert.
Wir nehmen die konstante Zahl 5.
Unsere Bedingung heißt also: Wenn der Spieler mindestens 5 Heiltränke hat, soll die Bedingung erfüllt sein.
Innerhalb des *Conditional Branchs* setzen wir nun die Textnachricht
\c[5]Apotheker: \c[0]Oh, ihr habt meine Heiltränke gefunden. Danke! Nun kann ich meinen Laden wieder öffnen. Nehmt diese zwei Heiltränke als Lohn.
Nun ziehen wir dem Spieler drei Heiltränke ab. Wieso nur drei? Nun, der Ladenbesitzer will dem Spieler ja Zweie überlassen.
Das Abziehen der Items funktioniert auch über die „Change Item“ Funktion. Diesmal wählen wir jedoch „decrease“ als Operation und „Constant“ 3 als Operand.
Schließlich setzen wir noch den lokalen Schalter B auf ON. Dieser soll immerhin die dritte Seite aktivieren.
Nutzt hierfür wieder die Funktion „Control SelfSwitch“.
Unter dem Else setzen wir nun eine Textnachricht, welche die Enttäuschung des Apothekers ausdrückt. In etwa so:
\c[5]Apotheker: \c[0]Ihr habt noch nicht meine Heiltränke gefunden? Hinfort!
Wir könnten natürlich auch Gebrauch von unseren Kontrollzeichen machen, über die wir zuvor gesprochen haben.
Sicherlich erinnert ihr euch noch an das Zeichen \v[x], welches den Wert einer Variable anzeigt. Ich hatte gesagt, wir würden dies erst später im Tutorial behandeln. Nun, jetzt ist, denke ich, der Zeitpunkt gekommen.
Statt x muss man die ID der Variable hinschreiben. Das Kontrollzeichen wird im Spiel automatisch durch den Wert der jeweiligen Variable ersetzt.
Schreibt also stattdessen unter dem Else-Fall:
\c[5]Apotheker: \c[0]Ihr habt erst \v[1] meiner Heiltränke gefunden. Ich habe aber 5 versteckt!

Nun wird der Apotheker den Spieler immer informieren, wie viele Heiltränke er bereits gefunden hat.

Ende der Aufgabe

Schließlich kommen wir zu letzten Seite unseres Events. Nachdem der Apotheker seine fünf Heiltränke erhalten hat, soll er seinen Laden wieder eröffnen.
Geht also nun auf die dritte Seite und fügt bei „Conditions“ bei „SelfSwitch“ den lokalen Schalter B gleich ON ein. Der Inhalt der dritten Seite ist glücklicherweise simpel.
Eine Textnachricht begrüßt den Spieler
\c[5]Apotheker: \c[0]Oh, ihr seid es! Herein, herein!
Und danach wird über die „Shop Processing“ Funktion der Verkaufsladen aufgerufen.

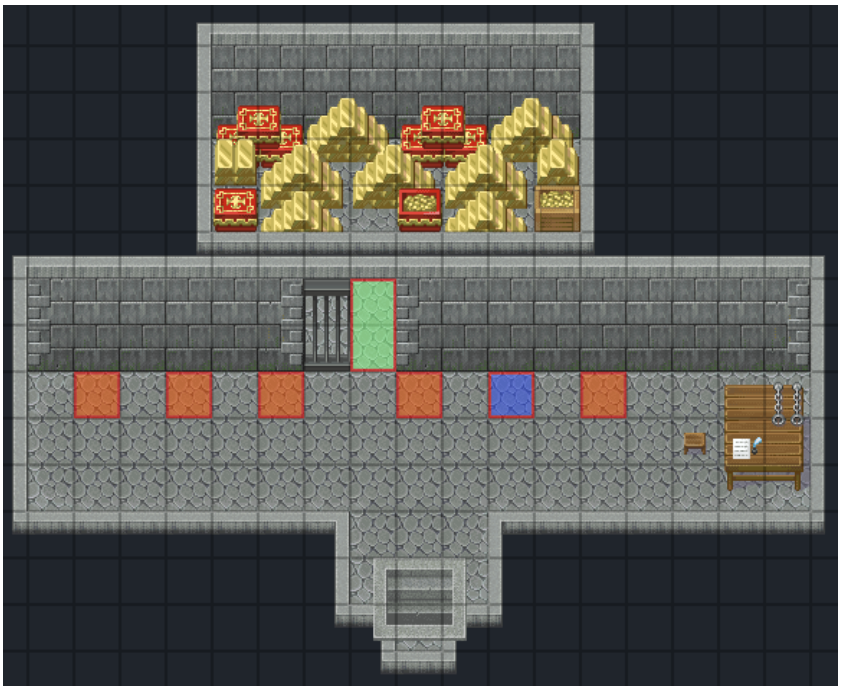
Zur Kontrolle noch einmal die Codes der drei Seiten im Überblick:



In dieser Übersicht wird zudem noch einmal erkenntlich, wie der Maker die Eventseiten abarbeitet. Er versucht stets die höchste Seitennummer auszuführen. Das wäre die Seite 3. Doch da die Seite 3 als Condition den lokalen Schalter B auf ON verlangt, kann die Seite nicht ausgeführt werden.
Also versucht der Maker die zweithöchste Seitennummer abzuspielen. Dies wäre die Seite 2. Da jedoch die Seite 2 als Condition den lokalen Schalter A auf On verlangt, geht auch dies nicht. Die nächst höhere Seitennummer wäre die Seite 1. Diese hat keine Bedingung und kann somit ausgeführt werden.
Sobald der Spieler den Apotheker anklickt, wird der lokale Schalter A auf ON gesetzt. Nun kann die Seite 2 ausgeführt werden. Da sie eine höhere Seitennummer hat als die Seite 1, wird nun nur noch Seite 2 ausgeführt. Hat der Spieler die 5 Heiltränke eingesammelt und zum Apotheker gebracht, so wird der lokale Schalter B auf ON gesetzt. Nun kann die dritte Seite ausgeführt werden. Und da sie die höchste Seitennummer hat, werden die ersten beiden Seiten nicht länger beachtet und es wird nur noch die dritte Seite ausgeführt.

Wiederholung: Schalter und Variablen

Zuletzt wollen wir an einem Script die Bedeutung von Variablen, lokalen sowie globalen Schaltern noch einmal wiederholen.
Der Spieler befindet sich in einem *Dungeon*. Hier befinden sich sechs verschiedene Hebel. Der Spieler muss fünf Hebel umdrehen, damit sich das Tor zur Schatzkammer öffnet. Doch einer der sechs Hebel ist eine Falle. Wird er benutzt, so werden alle Hebel deaktiviert und können nicht mehr umgedreht werden.

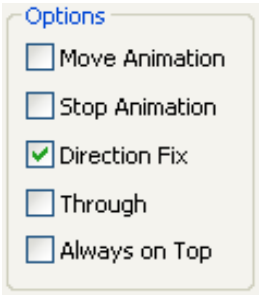


Hier wieder ein Screen von meiner Map. Benutzt wurde das *Tileset* „027: Castle Dunegeon“.
Das grüne Feld symbolisiert den Standort des Portals, welches den Weg zur Schatzkammer versperrt. Die orangen Felder stehen für die Hebel, welche das Tor öffnen. Das blaue Feld steht für die Falle, welche die Hebel deaktiviert.

Damit das Tor geöffnet wird, müssen 5 Hebel umgedreht sein. Es handelt sich hierbei um einen Zahlenwert, darum bietet es sich an, eine Variable zu benutzen. Jeder Hebel erhöht die Variable um 1. Hat die Variable den Wert 5 erreicht, so wird das Tor geöffnet.
Nachdem ein Schalter aktiviert wurde, darf er nicht erneut umgedreht werden. Sonst könnte man fünfmal hintereinander denselben Schalter betätigen, auf dass sich das Tor öffnet.
Hierbei handelt es sich um eine lokale Information. Der Hebel merkt sich selber, dass er nicht noch einmal benutzt werden darf. Deshalb wird hierfür ein lokaler Schalter verwendet.
Wird der falsche Schalter umgelegt, so werden alle Hebel deaktiviert. Hierbei handelt es sich also um eine Information, die alle Schalter erfahren müssen. Ein globaler Schalter ist genau das Richtige dafür.
Nach diesen Überlegungen steigen wir direkt ins Script ein. Wir beginnen mit den Hebeln.
Erstellt auf jedes (hier orange gekennzeichnete) Feld ein neues Event. Als Grafik wählt ihr „179-Switch02“. Ihr erstellt drei Seiten, alle drei Seiten mit derselben Grafik. Doch wählt ihr für die erste Seite diesen Hebel: 

Für die zweite Seite diesen Hebel: 

Und für die dritte Seite schließlich diesen: 



Bei den Eventeinstellungen müsst ihr lediglich beachten, dass bei „Options“ alle Haken entfernt sind, bis auf „Direction Fix“.

Die Seite 1 stellt den noch nicht aktivierten Hebel dar. Wenn man diesen Hebel anklickt, so soll ein *Klick*-Geräusch ertönen, die Variable „aktive Hebel“ um 1 erhöht werden und schließlich ein lokaler Schalter A auf ON gestellt werden, welcher die zweite Seite aktiviert.
Den Code hierfür solltet ihr mittlerweile beherrschen.
Das *Klick*-Geräusch ist ein Soundeffekt. Er wird also durch die Funktion „Play SE“ ausgeführt. Dabei wählen wir den Soundeffekt „034-Switch03“.
Das Erhöhen der Variable geschieht über den Befehl „Control Variables“.
Wählt ihr die Variable „0002: aktive Hebel“. Bei Operation stellt ihr „Add“ (addieren) ein. Als Operand dient die „Constant“ (Konstante) 1.
Mit einem „Control SelfSwitch“ könnt ihr letztendlich noch den lokalen Schalter A auf ON stellen.
Der fertige Code sollte nun folgendermaßen aussehen:

```
@>Play SE: '034-Switch03', 80, 100
@>Control Variables: [0002: aktive Hebel] += 1
@>Control Self Switch: A =ON
@>
```

Nun ist die zweite Seite an der Reihe. Sie wird aktiviert, sobald der lokale Schalter A auf ON steht. Also setzen wir bei „Conditions“ die Bedingung: SelfSwitch A is ON.
Nun, was passiert wenn man einen Hebel erneut umdreht? Richtig, er wird wieder zurückgesetzt. Wir fügen also denselben Code wie auf Seite 1 ein, nur mit zwei Unterschieden:

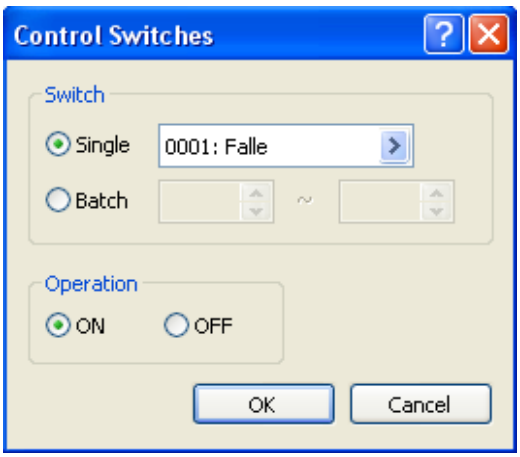
1. Die Variable wird nicht addiert, sondern subtrahiert („sub“)
2. Der lokale Schalter A wird auf OFF gesetzt

Schließlich sieht der Code der zweiten Seite so aus:

```
@>Play SE: '034-Switch03', 80, 100
@>Control Variables: [0002: aktive Hebel] -= 1
@>Control Self Switch: A =OFF
@>
```

Die dritte Seite wird aktiviert, sobald die Falle ausgelöst wurde. Nun funktionieren alle Hebel nicht mehr. Wie bereits erwähnt, ist die Falle ein globaler Schalter. Wir wählen als Condition diesmal die Bedingung: Switch 0001: is ON
Da unser Schalter mit der ID 1 noch keinen Namen hat, bezeichnen wir ihn als „Falle“.
Einen Inhalt braucht diese Seite nicht. Schließlich ist der Schalter ja kaputt.

Als nächstes bearbeiten wir die Falle.
Dieser Schalter erhält zwei Seiten, mit denselben Grafiken wie die ersten zwei Seiten der anderen Hebel.
Seite 1 soll die Falle ein *Klick*-Geräusch erzeugen und danach den Schalter „Falle“ auf ON setzen.
Der Code für das *Klick*-Geräusch ist derselbe wie bei den übrigen Hebeln. Für den globalen Schalter jedoch verwenden wir die *Control Switch* Funktion.



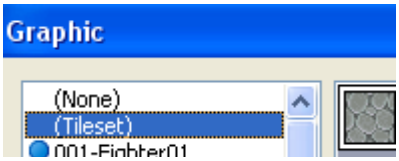
Diese Funktion ist ähnlich aufgebaut wie die „Control Variables“ Funktion. Bei „Switch“ könnt ihr entweder unter „Single“ (einzeln) einen einzelnen Schalter, oder über „Batch“ (Stapel) mehrere Schalter gleichzeitig umschalten.
Klickt auf „Single“ und wählt den Switch „0002: Falle“ aus.
Bei „Operation“ wird entschieden, ob der Schalter auf ON oder OFF gesetzt werden soll.
Wir wählen für unsere Falle natürlich ON.

Die zweite Seite erhält bei „Conditions“ die Bedingung: Switch „0001: Falle“ is ON.
Warum wir hier keinen lokalen Schalter verwenden?
Nun, sicherlich kann die Falle sich über einen lokalen Schalter selbst merken, ob sie aktiviert wurde, oder nicht. Doch alle anderen Schalter müssen auch darüber informiert werden. Wir können deshalb an dieser Stelle einen Schalter sparen, und den globalen Schalter „Falle“ für zwei Aufgaben verwenden:

- er soll alle anderen Hebel deaktivieren
- er soll die Falle deaktivieren, damit sie nicht erneut ausgelöst wird

Ansonsten bleibt die Seite 2 leer. Denn nachdem die Falle ausgelöst wurde, darf sie nicht wieder ausgeschaltet werden.

Als nächstes wenden wir uns der Tür zu. Da sie zwei Felder der Map einnimmt, müssen wir entweder ein Event mit zwei Felder großer Grafik, oder aber zwei Events mit je ein Feld großer Grafik auswählen.
Wir entscheiden uns für letzteres und erstellen zwei Events mit je zwei Seiten.
Als Grafik für die Events wählen wir ein Tile aus der Map
.
Hierfür klickt beim Auswählen der Grafik in der großen Liste auf „(Tileset)“ (welches direkt an zweiter Stelle steht).
Nun könnt ihr ein beliebiges Tile aus eurem *Tileset* als Eventgrafik auswählen.



Das Tile für die erste Seite des unteren Türevents

Das Tile für die erste Seite des oberen Türevents

Das Tile für die zweite Seite des unteren Türevents

Das Tile für die zweite Seite des oberen Türevents

Ansonsten gestaltet sich das Türevent recht einfach. Belasst bei den „Options“ auf jeder Seite sämtliche Einstellungen ohne Haken.
Dadurch übernimmt das Event automatisch die Einstellungen des jeweiligen *Tiles*, dessen Grafik es trägt. Das heißt: Wenn das Event die Grafik eines unbegehbaren *Tiles* besitzt, so ist das Event automatisch auch unbegehbar.
In unserem Fall heißt dies, das die erste Seite des unteren Türevents unbegehbar ist. Alle anderen Seiten des unteren und oberen Events sind begehbar.
Wir müssen nun nur noch bei „Conditions“ auf der zweiten Seite beider Türevents die Bedinung einstellen: Variable „0002: aktive Hebel“ is 5 or above

Dadurch wird die zweite Seite erst dann aktiviert, wenn die Variable „aktive Hebel“ den Wert 5, oder einen höheren Wert angenommen hat. Die Tür geht also erst auf, wenn alle fünf Hebel umgeschaltet wurden.

Das wäre dann auch schon alles.
Dieses Script sollte noch einmal als Wiederholung für die drei Informationsträger des Makers dienen. Ihr habt noch einmal deutlich gesehen, an welchen Stellen man lokale Schalter verwendet, wo globale Schalter nötig sind, oder wo man lieber zu Variablen greifen sollte.

Damit neigt sich dieses Tutorial auch schon dem Ende zu. Die wohl wichtigsten Themen dieses Kurses (Bedingungen und Informationsträger) solltet ihr euch noch einmal verinnerlichen, bevor der dritte Teil des Kurses beginnt.

Schlusswort

In diesem Tutorial haben wir uns mehr auf das Erstellen von typischen Makerskripten konzentriert. Dabei wurden die typischen Kontrollstrukturen des Makers wie *Conditional Branch* und *Show Choice* besprochen, ebenso wie manche Standardbefehle. Ich denke mittlerweile sollten die Leser in der Lage sein, über das Tutorial hinaus bereits ein eigenes RPG zu entwickeln.
Natürlich wird auch nächsten Monat wieder ein Tutorial erscheinen. Dort werden wir uns tiefgründiger mit Variablen beschäftigen, sowie einige neue Kontrollstrukturen (*Labels* und *Cycles*) behandeln. Des Weiteren werden wir diese Neuerungen stets an typischen Scripten wie einem Tag/Nacht-System, sowie einer Uhrzeit demonstrieren.
Geplant ist darüber hinaus noch ein Einbeziehen des Spielkonzeptes. Zum Makern gehört nun mal mehr als das Beherrschen der Technik. Eine gute Story, ein interessantes Konzept, sowie ausgeglichene Spielschwierigkeit sind ebenso bedeutend, weshalb wir in den späteren Kursen auch hierauf eingehen wollen.
Bis dahin könnt ihr euch ja schon mal darin üben, die bereits erlernten Fertigkeiten an anderen Scripten anzuwenden. Versucht doch einmal, statt den *Conditional Branch* im Heiltrankversteckevent mit verschiedenen Seiten zu arbeiten. Oder statt den Heiltrank-Items sofort Variablen zu verwenden.
Sollten Fragen zum Tutorial bestehen, so steht euch das Technik-Forum von http://rpga.info/forum jederzeit offen.
Statt die verschiedenen Befehle der Eventbefehlsliste listenartig abzuarbeiten, haben wir uns entschieden, diese in Form von Scripten zu erklären. Das mag sicherlich anschaulicher und motivierender wirken, ist jedoch auch weniger übersichtlicher.
Deshalb gibt es nun das Register, in dem die erwähnten Eventfunktionen und deren Bedeutung aufgelistet sind.
Damit danke ich den Leser für sein Interesse und hoffe, dass er das neu Erlernte auch erfolgreich anwenden kann. Somit verabschiede ich mich bis zum nächsten Teil des RPG-Maker XP Kurses.

Register

Befehl	*Seite*	*Spalte*	*Zeile*	*Bedeutung*
Show Text	1	links	1	Zeigt eine Textnachricht an
Show Choices	1	links	2	Erstellt verschiedene Auswahlmöglichkeiten
Wait	1	links	6	Hält den Code für eine bestimmte Framezahl an
Conditional Branch	1	links	8	Baut eine Bedingung ein
Control Switches	1	rechts	1	Schaltet einen globalen Schalter auf ON oder OFF
Control Variables	1	rechts	2	Weist einer Variable einen neuen Wert zu
Control SelfSwitch	1	rechts	3	Schaltet einen lokalen Schalter auf ON oder OFF
Change Gold	1	rechts	5	Erhöht/Verringert Geldbetrag des Spielers
Change Item	1	rechts	6	Fügt Items dem Inventar hinzu/entfernt Items aus dem Inventar
Transfer Player	2	links	1	Teleportiert Spieler auf eine bestimmte Position einer Map
Change Screen Color Tone	2	links	13	Verändert den Farbton des Bildschirms
Play BGM	2	rechts	7	Spielt eine Hintergrundsmusik ab
Fade Out BGM	2	rechts	8	Lässt die Hintergrundsmusik ausklingen
Play BGS	2	rechts	9	Spielt Hintergrundsgeräusche ab
Fade Out BGS	2	rechts	10	Lässt die Hintergrundsgeräusche ausklingen
Memorize BGM/BGS	2	rechts	11	Merkt sich die momentane Hintergrundsmusik und Hintergrundsgeräusche
Restore BGM/BGS	2	rechts	12	Spielt die gemerkte Hintergrundsmusik und Hintergrundgeräusche ab.
Play ME	2	rechts	13	Spielt einen kurzen, einmaligen Musikeffekt ab
Play SE	2	rechts	14	Spielt einen kurzen, einmaligen Geräuscheffekt ab
Stop SE	2	rechts	15	Bricht das Abspielen aller Geräuscheffekte ab
Shop Processing	3	links	2	Ruft das Kauf/Verkaufs-Fenster auf
Recover All	3	links	7	Heilt einen Helden oder die ganze Truppe vollständig