Das tttool-Buch

Release 1.11

Joachim Breitner

09.09.2023

Inhaltsverzeichnis:

Willkommen auf den Seiten des `tttool`-Buchs. Wenn du die Muße hast, so führe dir doch das Vorwort zu Gemüte, ansonsten kannst du auch gleich den Ersten Schritten folgen. Wenn dies nicht dein erster Kontakt mit dem `tttool` ist, so willst du vermutlich etwas in den beiden Referenz-Kapitel nachschlagen, oder durch die Tipps und Tricks stöbern.

KAPITEL 1

Vorwort

Seit den frühen 2010er Jahren erfreut der Tiptoi-Stift aus dem Hause Ravensburger Kinder in Deutschland und in aller Welt. Mit ihm entlockt das Kind den schön gestalteten Tiptoi-Büchern und -Spielen allerlei Töne, Geräusche, Musik und Geschichten. Offensichtlich steckt in dem Stift ein kleiner Computer, auf dem kleine Programme ablaufen. Da stellt sich dir sicherlich die Frage: Kannst du mit dem Stift noch mehr machen, als nur auf die original Ravensburger-Produkte zu reagieren?

Ja, kannst du! Denn – nach einigem Reverse-Engineering der Dateien, die man auf den Stift lädt, wenn man ein neues Produkt gekauft hat, sowie der feinen Punktmuster, die auf die Produkte gedruckt sind – bildete sich eine kleine Gemeinschaft von Tiptoi-Bastlern, die allerlei kleine und große eigene Tiptoi-Bücher und -Spiele entwickelt haben. Technisch dreht sich dabei alles um das `tttool`, ein Kommandozeilenprogramm mit dem man die Tiptoi-Dateien und -Codes erzeugt.

Dieses Buch ist sozusagen das Handbuch zum `tttool`, aber noch mehr: Es erklärt dir allgemeines zur Funktionsweise des Stiftes und wie du dein eigenes Tiptoi-Projekt verwirklichst.

1.1 Rechtliches

Eine häufig gestellte Frage ist: Darf ich das überhaupt. Diese Frage lässt sich nicht einfach beantworten, und keiner der Autoren ist Jurist.

Was auf jeden Fall gar nicht geht, ist mit dem `tttool` ein Tiptoi-kompatibles Produkt zu erstellen und so zu tun, als ob es von Ravenburger sei.

Was sicherlich auch Ärger gibt, ist wenn du dein eigenes Tiptoi-Produkt erstellst und kommerziell vertreibst, oder damit etwas machst das der Marke “Tiptoi” schadet – etwa ein eindeutig für Kinder ungeeignetes Thema.

Alles andere – Werke für den Eigengebrauch oder als Geschenk, Anleitungen online, eine offene Diskussion über die Funktionsweise des Stifts – hat irgendwer schon mal gemacht. Wir gehen davon aus dass Ravensburger davon weiß und uns stillschweigend – vielleicht gar wohlwollend – gewähren lässt.

[1] CC BY-SA 3.0, by Schwesterschlumpf, from Wikimedia Commons

Abb. 1: Tiptoi-Stift mit Spielbrett[1]

1.2 Grafische Tools

Das `tttool` ist ein Kommandozeilentool. Wer sich davon nicht abschrecken lässt, kann tolle Sachen damit machen. Wer es einfacher haben will, sollte sich folgende Projekte anschauen, die allesamt auf `tttool` aufbauen:

- Andreas Grimme hat mit ttaudio eine Windows-GUI erstellt, falls man einfach nur ein paar Audio-Dateien auf den Stift laden will.
- Till Korten hat mit ttmp32gme eine grafische Anwendung (Windows, OS X und Linux) erstellt, die ebenfalls Audio-Dateien auf den Stift lädt und sehr schöne Übersichten zum Antippen druckt.
- Auf der Plattform https://soundolino.ch/ kann man Tiptoi-kompatible Sticker mit für den Unterrichtseinsatz geeigneter Vertonung bestellen, selber drucken und auf der Webseite auch sehr einfach mit eigenen Tönen versehen.

1.3 Weiterführende Informationen

Das Ziel dieses Buches ist, dir alles nötige über die Bastelei mit dem Tiptoi-Stift zu erklären. Doch dieses Ziel wird vermutlich nie erreicht… solltest du also mehr Informationen brauchen, findest du sie hier:

- Du solltest unbedingt die tiptoi-Mailingliste abonnieren. Hier tauschen sich Tiptoi-Bastler aus und irgendwer kann dir sicherlich weiterhelfen. Auch freuen wir uns sehr zu erfahren, was du so mit dem `tttool` auf die Beine gestellt hast.
- Fehler im `tttool` oder Verbesserungsvorschläge, sowohl zum `tttool` als auch zu diesem Buch, darfst du gerne über den Github-Bugtracker melden.
- Wenn du dich für technische Details interessierst, die das `tttool` eigentlich vor dir versteckt, so solltest du ins Wiki der Github-Seite schauen.

KAPITEL 2

Erste Schritte

Dieses Kapitel erklärt dir Schritt für Schritt und an einfachen Beispielen, was du mit dem *tttool* anstellen kannst. Die Details werden dann vollständig in den nächsten Kapiteln erklärt.

2.1 Vorbereitung

Auf der Release-Seite findest du eine Zip-Datei, die du herunterlädst und entpackst. Du findest darin die Datei `tttool.exe` (für Windows) oder `tttool` (für Linux und OSX), die man direkt ausführen kann. Es ist keine weitere Installation nötig.

Allerdings ist zu beachten, dass es sich dabei um ein Kommandozeilenprogramm handelt. Doppelt klicken bringt also nichts, sondern du musst die Eingabeaufforderung starten, in das Verzeichnis mit der Datei `tttool.exe` wechseln und dann Befehle wie `./tttool info WWW_Bauernhof.gme` eintippen.

Bemerkung: Ein vorangestelltes `$` in folgenden Listings wird nicht mit eingegeben, sondern markiert die Zeilen, die einzugeben sind. Wenn dir das neu ist, dann sei dir das Kapitel Einführung in die Kommandozeile eines Python-Kurses empfohlen.

2.2 Töne in Tiptoi-Büchern ändern

Als erstes einfaches Projekt kannst du in einem deiner Tiptoi-Bücher ein paar Töne ändern. Überrasche dein Kind doch, indem du deine eigene Stimme ertönen lässt! Als Beispiel nehmen wir das Buch „Wieso? Weshalb? Warum? – Bauernhof“, aber du kannst diese Anleitung auch mit einem anderen Buch verfolgen.

Zu jedem Tiptoi-Produkt gehört eine „GME-Datei“, die sowohl die Töne als auch die Programmlogik enthält – in unserem Fall `WWW_Bauernhof.gme`. Du findest sie auf dem Tiptoi-Stift selbst, oder auf der Ravensburger Download-Seite. Kopiere Sie in den Ordner mit dem `tttool.exe`.

Führe nun die folgenden beiden Befehle aus:

```
$ ./tttool export WWW_Bauernhof.gme
$ ./tttool media WWW_Bauernhof.gme
```

Du solltest nun in diesem Verzeichnis eine Datei `WWW_Bauernhof.yaml` finden (die du vorerst ignorierst), sowie ein Verzeichnis `media/` mit vielen Audio-Dateien im `ogg`-Format finden. Hör einfach mal rein!

Damit bist du schon halb fertig. Suche die Audio-Datei, die du ersetzen willst und merke dir den Dateinamen – zum Beispiel `WWW_Bauernhof_3.ogg` für die Frage „Was ist das besondere an dem Kuhstall“. Du kannst die Datei ruhig löschen, denn du willst sie durch deine eigene Aufnahme ersetzen.

Zu erklären, wie du eine Audiodatei aufnimmst würde hier den Rahmen sprengen. Wichtig ist vor allem, dass du die Datei im OGG- oder MP3-Format aufnimmst. Speichere sie in dem `audio`-Verzeichnis als `WWW_Bauernhof_3.ogg`.

Nun musst du die GME-Datei wieder zusammenbauen:

```
$ ./tttool assemble WWW_Bauernhof.yaml Mein_Bauernhof.gme
```

Nun solltest du in dem Verzeichnis die Datei `Mein_Bauernhof.gme` finden. Kopiere diese Datei auf den Tiptoi-Stift. Es darf immer nur eine GME-Datei pro Produkt auf dem Stift sein, also musst du die originale `WWW_Bauernhof.gme` löschen – natürlich nur nachdem du eine Sicherheitskopie davon auf deinem Rechner erstellt hast.

Das war es schon! Wenn du nun den Stift einschaltest, den Bauernhof aktivierst und auf den Kuhstall tippst, solltest du deine eigene Stimme hören.

Bemerkung: Da wir das Dateiformat nicht vollständig verstanden haben, kann es sein, dass manche Elemente – insbesondere Spiele – nun nicht mehr funktionieren.

2.3 Eigene Tiptoi-Produkte herstellen

Als nächstes wirst du ein komplett eigenes Tiptoi-Projekt herstellen. Als kleines Beispiel nehmen wir hier ein Tic-Tac-Toe-Spiel – das braucht nur wenige aktive Bereiche und ist grafisch simpel, aber zeigt schon, wie du komplexere Abläufe programmierst.

Für ein Tiptoi-Produkt brauchst du zwei Komponenten:

- die GME-Datei mit der Programmierung und den Audio-Dateien, und
- das Buch (hier, das Blatt Papier) mit der Grafik und den Codes.

Bei beidem hilft das `tttool`, und in beiden ist der Ausgangspunkt die sogenannte YAML-Datei.

2.3.1 Der erste Ton

Du beginnst also mit einer einfachen YAML-Datei. Öffne den Texteditor deiner Wahl, schreibe folgendes und speichere es als `tic-tac-toi.yaml`:

```
product-id: 900
comment: Tic-Tac-Toe fuer den Tiptoi
scripts:
  feldOL: P(oben_links)
```

YAML ist ein generisches Datenformat, das du mit einem beliebigen Texteditor erstellen und bearbeiten kannst. Beachte dabei dass in YAML Einrückungen, also Leerzeichen am Anfang der Zeile, wichtig sind. Beachte auch, dass die Dateiendung `.yaml` lautet, und nicht `.yml`.

Die einzig wichtigen Felder sind `product-id` und `scripts`. Die `product-id` ist eine Nummer, die dein Projekt indentifiziert, und mit der am Ende der Stift die passende GME-Datei zum ausgewählten Buch findet. Das `comment`-Feld hat keine weitere Bedeutung. Wirklich spannend ist das `scripts`-Feld: Hier wird festgelegt, dass es ein Feld namens "`feldOl`" geben wird, und wenn der Benutzer mit dem Stift drauf geht, soll die Audiodatei namens `oben_links` abgespielt werden.

Diese Audiodatei muss nun irgendwo herkommen. Du kannst sie selber aufnehmen und als `oben_links.ogg` abspeichern, und am Ende wirst du das sicher machen wollen. Aber gerade während du dein Tiptoi-Produkt noch entwickelst, oder für schnelle Experimente, ist das sehr hinderlich.

Daher kann das `tttool` auch selbst die Ansagen erstellen, mittels Text-To-Speech[1]. Füge dazu folgende Zeilen der Datei hinzu:

```
language: de
speak:
  oben_links: "Du hast oben links hingetippt"
```

Damit ist die YAML-Datei schon brauchbar! Mit dem Befehl

```
$ ./tttool assemble tic-tac-toi.yaml
```

wird dir eine Datei namens `tic-tac-tiptoi.gme` erstellt, die du auf den Stift kopierst.

Nun brauchst du noch den zugehörigen Ausdruck mit den Punktmustern. Auch hier musst du dich noch nicht gleich an die fertige Gestaltung machen: Mit dem Befehl

```
$ ./tttool oid-table tic-tac-toi.yaml
```

erstellt dir das `tttool` die Datei `tic-tac-tiptoi.pdf`, die alle Punktmuster für dein Werk in einer nüchternen, aber praktischen Tabelle enthält. Du siehst dort neben dem Feld für `feldOL` auch eines mit der Beschriftung `START`, dem Anschaltzeichen für dein Produkt, sowie `REPLAY` und `STOP`, die man erstmal ignorieren darf.

Wenn du diese Datei nun ausdruckst, mit dem Stift auf das Anschaltzeichen gehst, und danach auf das andere Feld, solltest du eine Roboterstimme hören, die „Du hast oben links hingetippt." sagt.

Bemerkung: Das Drucken ist die hakeligste Sache an der ganzen Bastelei. Mit manchen Druckern klappt es auf Anhieb, mit anderen muss man lange mit den Druckeinstellungen herumspielen, oder die Punktmuster deutlich fetter auftragen, bei anderen ist gar nichts zu machen. Vielleicht hilft dir die Seite zum Thema Drucken auf dem Tiptoi-Wiki weiter, wenn es nicht gleich klappt.

2.3.2 Spiellogik programmieren

Nun füllst du das Spielfeld mit Leben. Dazu musst du dir überlegen, was sich das Programm „merken" muss.

Für das Feld oben links muss es sich zum Beispiel merken, ob es leer ist, von Spieler 1 (Kreuz) belegt oder von Spieler 2 (Kreis) belegt ist. Dazu verwendest du das Register `$OL`, was entsprechend die Werte 0, 1 oder 2 speichert. Wir sprechen hier von Registern, aber man kann genau so gut Variablen oder Speicherplätze sagen.

[1] Für Windows bringt das `tttool` die nötigen Programme mit, auf anderen Systemen wird dich das `tttool` gegebenenfalls bitten, weitere Pakete zu installieren.

Wenn ein Spieler nun `feldOL` antippt, und es ist schon belegt, so möchtest du ihn wissen lassen, was hier schon ist. Ist es allerdings frei, so musst du `$OL` entsprechend ändern. Dazu musst du natürlich wissen, wer dran ist! Das speicherst du im Register `$turn`, mit den Werten 1 und 2.

Insgesamt hast du also drei Fälle, die du wie folgt aufschreibst:

```
scripts:
  feldOL:
  - $OL == 0? $OL := $turn J(here_now)
  - $OL == 1? P(here_cross) J(next)
  - $OL == 2? P(here_circle) J(next)
```

Wenn der Benutzer nun das Feld antippt, prüft der Stift die drei Zeilen der Reihe nach, und führt die erste aus, wo alle Bedingungen passen. Bedingungen erkennnst du an dem Fragezeigen: Hier prüfst du, welcher Wert in `$OL` gespeichert ist. Den `P(...)`-Befehl kennst du schon, der gibt eine Audiodatei aus (und um die eigentliche Audiodateien kümmerst du dich später). Mit `$OL := $turn` wird die Nummer des aktuellen Spielers (laut `$turn`) in das Feld geschrieben (`$OL`).

In allen drei Fällen willst du allerdings noch mehr machen: Wenn ein neues Feld belegt wurde, willst du den Stift das sagen lassen. Und wenn nicht, willst du zumindest sagen, wer nun dran ist. Da das bei allen Feldern der gleiche Code sein wird, programmierst du ihn im Folgenden nur einmal, und springst den Code mit dem `J`-Befehl (für „Jump“) an.

Um zum Beispiel zu verkünden, wer eigentlich gerade dran ist, füge das folgende Skript hinzu, das du oben mit `J(next)` anspringst:

```
scripts:
  ...
  next:
  - $turn == 1? P(player1)
  - $turn == 2? P(player2)
```

Wie bereits erwähnt willst du dem Spieler auch sagen, wenn ein neues Kreuz oder ein neuer Kreis gesetzt wird. Das machst du im Skript `here_now`:

```
scripts:
  ...
  here_now:
  - $turn == 1? $set += 1 P(here_now_cross) J(win_check)
  - $turn == 2? $set += 1 P(here_now_circle) J(win_check)
```

Wie auch im vorherigen Skript prüfst du zuerst, wer gerade dran ist, um dann die entsprechende Meldung mittels `P(...)` auszugeben. Du zählst an der Stelle im Register `$set` mit, wie viele Felder insgesamt besetzt sind – das wird in Kürze nützlich sein. In beiden Fällen machst du anschließend mit `win_check` weiter, dem kompliziertesten Skript bisher, in dem du prüfst ob der aktuelle Spieler vielleicht gewonnen hat:

```
scripts:
  ...
  win_check:
  - $OL == $turn? $ML == $turn? $UL == $turn? J(win)
  - $OM == $turn? $MM == $turn? $UM == $turn? J(win)
  - $OR == $turn? $MR == $turn? $UR == $turn? J(win)
  - $OL == $turn? $OM == $turn? $OR == $turn? J(win)
  - $ML == $turn? $MM == $turn? $MR == $turn? J(win)
  - $UL == $turn? $UM == $turn? $UR == $turn? J(win)
  - $OL == $turn? $MM == $turn? $UR == $turn? J(win)
  - $OR == $turn? $MM == $turn? $UL == $turn? J(win)

  - $set == 9? P(draw) J(reset)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  - $turn == 1? $turn := 2 J(next)
  - $turn == 2? $turn := 1 J(next)
```

Zuerst gehst du alle acht Gewinn-Kombinationen, also die drei Spalten, die drei Zeilen und die zwei Diagonale, durch und prüfst, ob alle drei Felder dem aktuellem Spieler gehören. Wenn ja, dann hat er gewonnen! (Und der Stift macht mit dem Skript `win` weiter.)

Wenn der aktuelle Spieler nicht gewonnen hat, schaust du, ob trotzdem das Feld voll ist. Das erkennst du daran, dass das Register `$set`, das mitzählt, wie viele Felder belegt sind, den Wert 9 hat. Wenn ja, dann lässt du verlautbaren, dass das Spiel unentschieden endete, und beginnst von vorne.

Und sollte auch das nicht passiert sein, so änderst du aktuellen Spieler (und sagst wer jetzt dran ist, siehe oben).

Jetzt bist du fast fertig. Wenn der aktuelle Spieler gewinnt, dann willst du das verkünden, und das Spiel neu starten, mit dem Verlierer als neuen Startspieler:

```
scripts:
  ...
  win:
  - $turn == 1? P(player1wins) $turn := 2 J(reset)
  - $turn == 2? P(player2wins) $turn := 1 J(reset)
```

Und wenn du das Spiel neu startest, musst du alle Felder leeren:

```
scripts:
  ...
  reset: $set:=0 $OL:=0 $OM:=0 $OR:=0 $ML:=0 $MM:=0 $MR:=0 J(reset2)
  reset2: $UL:=0 $UM:=0 $UR:=0 J(next)
```

Du machst das mit zwei Skripten, weil der Tiptoi-Stift es nicht immer mag, wenn mehr als 8 Befehle in einem Skript sind. Das `tttool` würde dich allerdings warnen, falls du das mal vergisst.

Natürlich musst du für alle 9 Felder ein Program wie `feldOL` schreiben, aber die sehen genau so aus wie jenes, nur statt `OL` steht dann da `OM`, `OR`, `ML`, und so weiter. Das ist ein wenig repetitiv, aber da kommst du nicht ohne Weiteres drum rum.

Nun bist du fast fertig mit der Programmierung. Es fehlt nur noch ein Detail: Du musst sicherstellen, dass ganz am Anfang alle Register einen vernünftigen Wert haben. Wenn du nichts machst, sind die Register anfangs alle auf 0, was für die Felder und `$set` durchaus passt. Aber der aktuelle Spieler, `$turn`, muss ja stets 1 oder 2 sein. Deshaben gibst du ein Init-Skript an.

```
init: $turn := 1
```

Die Init-Zeile darf nur Zuweisungen enthalten, aber du kannst trotzdem beim Anschalten auch Audio-Dateien abspielen. Dazu schreibst du sie in die `welcome`-Zeile:

```
init: $turn := 1
welcome: welcome, player1
```

Sowohl `init` als auch `welcome` gehören übrigens in die erste Spalte, und nicht etwa unterhalb von `skripts:` eingerückt.

Wenn du jetzt noch den `speak`-Abschnitt vervollständigst, so ist die YAML-Datei endlich fertig. Hier nochmal in voller Länge und am Stück:

```
product-id: 900
comment: Tic Tac Toe for the Tiptoi
init: $turn := 1
welcome: welcome, player1
language: de

scripts:
  feldOL:
  - $OL == 0? $OL := $turn J(here_now)
  - $OL == 1? P(here_cross) J(next)
  - $OL == 2? P(here_circle) J(next)
  feldOM:
  - $OM == 0? $OM := $turn J(here_now)
  - $OM == 1? P(here_cross) J(next)
  - $OM == 2? P(here_circle) J(next)
  feldOR:
  - $OR == 0? $OR := $turn J(here_now)
  - $OR == 1? P(here_cross) J(next)
  - $OR == 2? P(here_circle) J(next)
  feldML:
  - $ML == 0? $ML := $turn J(here_now)
  - $ML == 1? P(here_cross) J(next)
  - $ML == 2? P(here_circle) J(next)
  feldMM:
  - $MM == 0? $MM := $turn J(here_now)
  - $MM == 1? P(here_cross) J(next)
  - $MM == 2? P(here_circle) J(next)
  feldMR:
  - $MR == 0? $MR := $turn J(here_now)
  - $MR == 1? P(here_cross) J(next)
  - $MR == 2? P(here_circle) J(next)
  feldUL:
  - $UL == 0? $UL := $turn J(here_now)
  - $UL == 1? P(here_cross) J(next)
  - $UL == 2? P(here_circle) J(next)
  feldUM:
  - $UM == 0? $UM := $turn J(here_now)
  - $UM == 1? P(here_cross) J(next)
  - $UM == 2? P(here_circle) J(next)
  feldUR:
  - $UR == 0? $UR := $turn J(here_now)
  - $UR == 1? P(here_cross) J(next)
  - $UR == 2? P(here_circle) J(next)

  here_now:
  - $turn == 1? $set += 1 P(here_now_cross) J(win_check)
  - $turn == 2? $set += 1 P(here_now_circle) J(win_check)

  win_check:
  - $OL == $turn? $ML == $turn? $UL == $turn? J(win)
  - $OM == $turn? $MM == $turn? $UM == $turn? J(win)
  - $OR == $turn? $MR == $turn? $UR == $turn? J(win)
  - $OL == $turn? $OM == $turn? $OR == $turn? J(win)
  - $ML == $turn? $MM == $turn? $MR == $turn? J(win)
  - $UL == $turn? $UM == $turn? $UR == $turn? J(win)
  - $OL == $turn? $MM == $turn? $UR == $turn? J(win)
  - $OR == $turn? $MM == $turn? $UL == $turn? J(win)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  - $set == 9? P(draw) J(reset)
  - $turn == 1? $turn := 2 J(next)
  - $turn == 2? $turn := 1 J(next)

  win:
  - $turn == 1? P(player1wins) $turn := 2 J(reset)
  - $turn == 2? P(player2wins) $turn := 1 J(reset)

  reset: $set:=0 $OL:=0 $OM:=0 $OR:=0 $ML:=0 $MM:=0 $MR:=0 J(reset2)
  reset2: $UL:=0 $UM:=0 $UR:=0 J(next)

  next:
  - $turn == 1? P(player1)
  - $turn == 2? P(player2)

speak:
  welcome: "Willkommen bei Tic-Tac-Tiptoi."
  player1: "Kreuz ist dran."
  player2: "Kreis ist dran."
  here_cross: "Hier ist schon ein Kreuz"
  here_circle: "Hier ist schon ein Kreis"
  here_now_cross: "Hier ist jetzt ein Kreuz"
  here_now_circle: "Hier ist jetzt ein Kreis"
  player1wins: "Kreuz gewinnt."
  player2wins: "Kreis gewinnt."
  draw: "Unentschieden"
```

2.3.3 Die Programmierung testen

Natürlich schreibst du so ein Programm nicht von oben bis unten runter, sondern in kleinen Stücken, die du dann zwischendurch testest.

Eine Möglichkeit ist natürlich, mittels

```
$ ./tttool oid-table tic-tac-toi.yaml
$ ./tttool assemble tic-tac-toi.yaml
```

eine neue OID-Tabelle und eine neu GME-Datei zu erzeugen, diese zu drucken bzw. auf den Stift zu kopieren, und dann mit der echten Hardware zu testen.

Doch auch wenn man die Tabelle nur dann ausdrucken muss, wenn man neue Felder hinzugefügt hat, ist das relativ nervig. Daher bietet das `tttool` ein Simulations-Modus, wo du eintippst, welches Feld man antippt, und es spielt dann die entsprechenden Audio-Dateien ab (Lautsprecher anschalten!):

```
$ ./tttool play tic-tac-toi.yaml
Initial state (not showing zero registers): $0=0 $10=1
Playing audio sample 9
Playing audio sample 5
Next OID touched? feldOL
Executing:  $3==0? $3:=$10 J(11263)
Executing:  $10==1? $9+=1 P(4) J(11267)
Playing audio sample 4
Executing:  $10==1? $10:=2 J(11264)
Executing:  $10==2? P(7)
Playing audio sample 7
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
State now: $0=0 $3=1 $9=1 $10=2
Next OID touched?
```

So kannst du relativ schnell testen ob der neue Code funktioniert.

2.3.4 Das Spielbrett gestalten

Damit ist die Programmierung des Tic-Tac-Toe-Spiels abgeschlossen, und du kannst dich an die grafische Gestaltung machen.

Eine sehr einfache Möglichkeit ist es, die Tabelle mit den Mustern, die du schon mit `tttool oid-table` erstellt hast, zu zerschneiden und die Muster auf ein konventionell gebasteltes Spielbrett zu kleben.

Aber du kannst natürlich auch am Rechner das Spielbrett gestalten, zum Beispiel mit einem Bildverarbeitungsprogramm wie dem kostenlosen Gimp. Wenn du ein anderes Programm verwendest, musst du die Anleitung ensprechend anpassen.

Im Gimp legst du ein neues Bild an, und achtest hierbei, dass es in der richtigen Auflösung erstellt wird: Unter „Erweiterte Einstellungen“ setzt du die X- und Y-Auflösung auf 1200dpi:

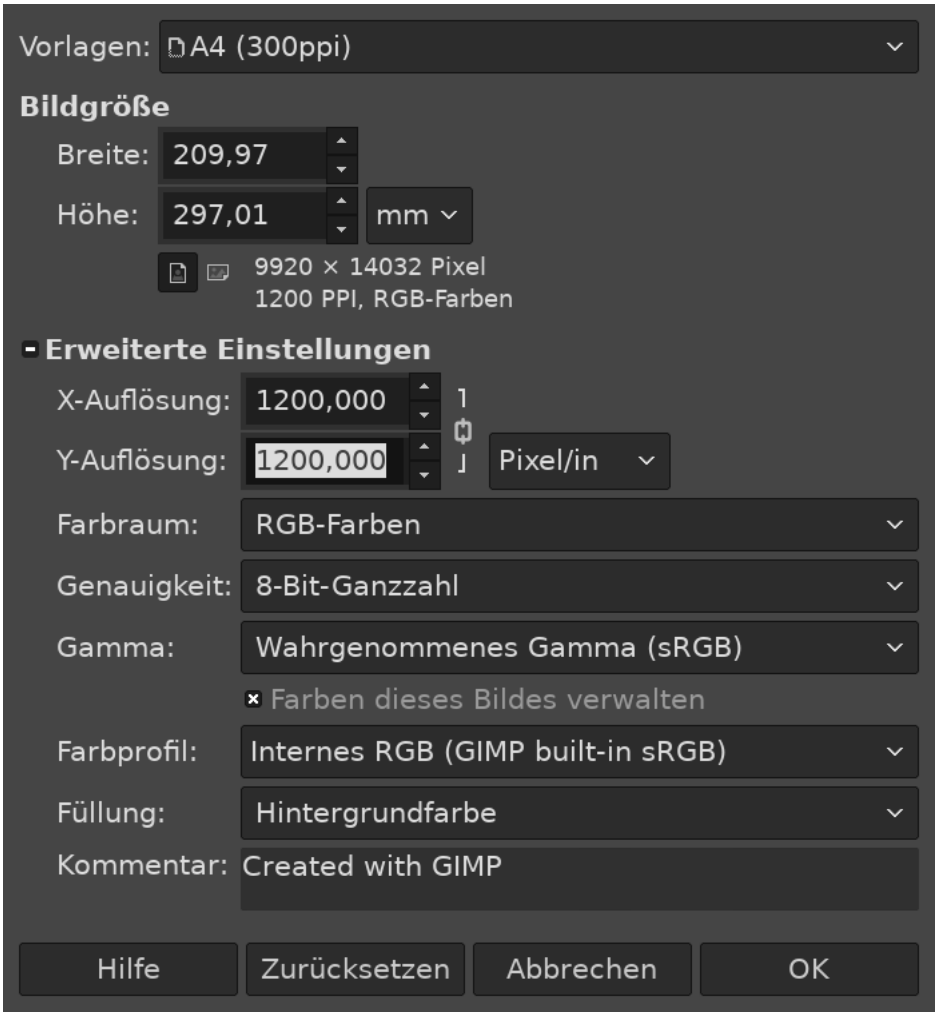

Abb. 1: Ein neues Projekt mit Gimp anlegen

Nun malst du nach Herzenslust das Spielbrett auf das Bild. Vermeide allerdings die Bereiche, die der Spieler nachher antippen soll, all zu dunkel zu gestalten. Bevor du weißt was dein Drucker so kann, lasse sie einfach erstmal weiß.

Wenn du zufrieden bist, gilt es, die Punktmuster über das Bild zu legen. Dazu erzeugst du erstmal eine PNG-Datei pro Muster:

```
$ ./tttool oid-codes tic-tac-toi.yaml
```

Du solltest jetzt im aktuellen Verzeichnis Dateien `oid-900-feldML.png`, `oid-900-feldMM.png` und so weiter finden. Die Muster für die Felder und das START-Muster lädst du über „Datei → Als Ebenen öffnen . . . “ in Gimp. Sie tauchen jetzt links in der Ebenenansicht auf.

Wenn die Auflösung des Dokuments stimmt, sollten sie 3×3cm groß sein. Schiebe sie über die entsprechenden Stellen, und wenn sie zu groß sind, dann schneide sie zu, oder arbeite mit Ebenenmasken.

Wichtig ist, dass du die Muster nie skalierst, sonst können sie nicht mehr erkannt werden! Wenn du größere Muster brauchst, dann kopiere sie mehrfach, oder erzeuge gleich größere Muster, etwa mittels

```
$ ./tttool --code-dim 100x100 oid-codes tic-tac-toi.yaml
```

für 10×10cm.

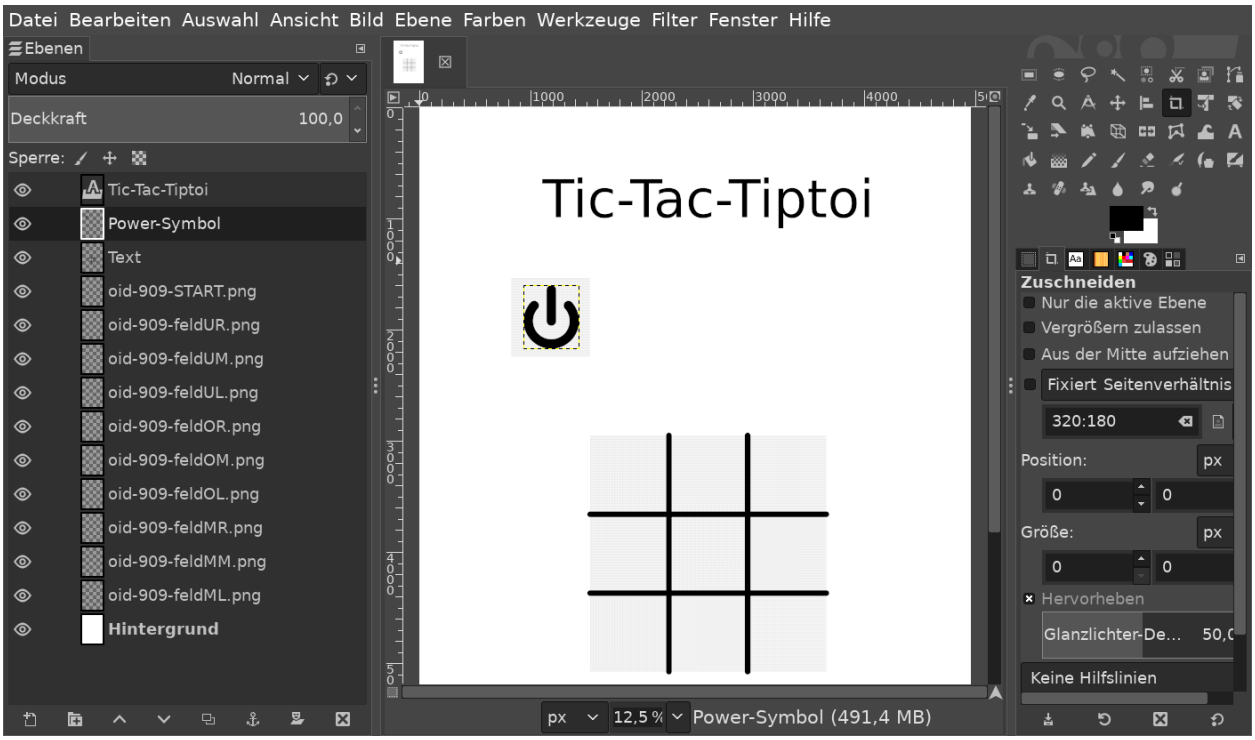

Abb. 2: Ein etwas lieblos gestaltetes Tic-Tac-Tiptoi

Nun druckst du das Bild direkt aus Gimp heraus aus, und achtest bei den Druckereinstellungen, dass es nicht skaliert wird (also nicht etwa „auf den Druckbereich anpassen“ auswählen“). Wenn die Druckergötter gnädig gestimmt sind, hälst du nun dein erstes selbst-gebautes Tiptoi-Spiel in der Hand!

2.3.5 Und jetzt?

Tic-Tac-Toe funktionert, und du hast Lust auf mehr? Dann lass dich doch auf der Galerie inspirieren, und lese in den nächsten Seiten, um einen Überblick darüber zu gewinne, wie der Tiptoi-Stift funktioniert, was das `tttool` so alles

kann, wie die YAML-Datei aufgebaut ist und was andere so alles an Tipps und Tricks für dich zusammen getragen haben.

Wenn du mal nicht weiterkommst, dann darfst du dich gerne an die Tiptoi-Mailingliste wenden. Und wenn du etwas schönes Gebastelt hast, freuen wir uns immer wenn du uns auf der Mailingliste davon erzählst.

Viel Vergnügen!

KAPITEL 3

Installation

Um eigene Tiptoi-Bücher zu gestalten brauchst du das `tttool`.

Egal ob du Windows, Linux oder OSX benutzt: Lade von https://github.com/entropia/tip-toi-reveng/releases die neueste Version als Zip-Datei (`tttool-version.zip`) herunter, und extrahiere sie. Du kannst das `tttool` direkt verwenden, es ist keine weitere Installation nötig.

Solltest du die selber kompilieren wollen, dann werfe einen Blick auf die Anweisungen im Quell-Repository.

KAPITEL 4

Konzepte

Du hast das `tttool` erfolgreich *installiert* und vielleicht schon ein paar *erste Schritte* gemacht? Dann ist es an der Zeit, einen vollständigen Überblick über die Arbeit mit dem Tiptoi-Stift zu bekommen.

4.1 Wie funktioniert der Stift?

In der Spitze des Tiptoi-Stifts steckt eine kleine Kamera. Wenn du mit dem Stift auf eine Seite eines Tiptoi-Buches tippst, so sucht die Kamera nach einem bestimmten Muster bestehend aus schwarzen Punkten. Dieses Muster ist sehr fein und mit bloßem Auge betrachtet fällt es kaum auf, aber wenn du genau hinschaust, kannst du es sehen.

Das Punktmuster selbst kodiert lediglich eine Zahl (zwischen 0 und 15000, um genau zu sein), den sogenannten *OID-Code*. Die eigentliche Logik des Stifts – d.h. was er wann sagt, also auch die eigentlichen Audio-Dateien – ist in der GME-Datei gespeichert.

Daraus ergibt sich, dass man den Stift umprogrammieren kann, indem man diese GME-Dateien ändert.

4.2 Was sind Anschaltfelder und Produkt-IDs?

Nun finden sich auf deinem Tiptoi-Stift sicherlich mehrere GME-Dateien. Woher weiß der Stift, in welcher er nachschauen muss? Dazu gibt es die *Produkt-IDs*!

Wenn du einfach ein Buch öffnest und mit dem Stift irgendwo hintippst, dann liest der Stift den entsprechenden OID-Code, weiß aber nicht was er damit anfangen soll und wird dich daher auffordern, das Anschaltfeld des Produktes anzutippen.

Jedes Produkt hat so ein Anschaltfeld, und es zeichnet sich dadurch aus, dass es einen OID-Code im Bereich 1 bis 1000 kodiert. Das ist gleichzeitig die Produkt-ID des Produkts, und jedes Tiptoi-Produkt hat eine eigene Produkt-ID.

Eine GME-Datei enthält auch eine Produkt-ID. Der Stift schaut nun in alle GME-Dateien, die du auf ihn geladen hast, und sucht die GME-Datei mit der entsprechenden Produkt-ID. Wenn er eine solche findet, lädt er sie. Wenn du nun ins Buch tippst, kann der Stift in dieser GME-Datei nachschauen, was er zu tun hat.

Bemerkung: Aktiviere doch ein Buch (z.B. den Bauernhof) und tippe dann auf Elemente in einem anderen Buch. Mit etwas Glück reagiert der Stift mit Bauernhof-Tönen. Wenn das passiert, dann wurde der gleiche OID-Code in verschiedenen Produkten verwendet. Das ist kein Fehler: Die Produkt-ID löst die Uneindeutigkeit auf.

4.3 Was steckt in einer GME-Datei?

Neben der Produkt-ID, wie gerade eben erklärt, sowie den Audio-Dateien, die der Stift abspielen kann (in der Regel als Ogg-Vorbis, Mono, 22050Hz, aber der Stift versteht auch andere Audioformate wie WAV und MP3) enthält er die Logik, was er wann abzuspielen hat.

In erster Näherung ist das eine einfache Tabelle, die zu jedem OID-Code die Audio-Datei angibt, die abzuspielen ist.

Aber da ist natürlich noch mehr, denn der Stift macht ja nicht immer das gleiche, wenn man auf ein Feld tippt. Tatsächlich enthält diese Tabelle zu jedem OID-Code ein kleines Computer-Programm, dass nach dem Tippen abläuft. Dieses Programm (oder *Skript*) kann

- Audio-Dateien abspielen,
- mit Zahlenwerten rechnen,
- diese Zahlenwerte in sogennanten Registern ablegen und abrufen und
- abhängig von diesen Werten unterschiedliche Programschritte abarbeiten.

Die Werte mit denen gerechnet wird sind dabei 16-bit natürliche Zahlen (also 0 bis 65535). Auch die Register (man könnte sie auch Variablen oder Speicherzellen nennen) speichern jeweils genau eine solche Zahl.

Desweiteren kann eine GME-Datei noch folgendes enthalten:

- Die Sprache der GME-Datei (Deutsch, Englisch usw.). Sollte die GME-Datei eine Sprache nennen, so wird sie nur abgespielt, wenn der Stift auf die gleiche Sprache eingestellt ist.
- Spiele. Die Logik mancher komplizierteren Abläufe („Finde alle Mäuse!“) sind fest im Stift eingebaut, und die GME-Datei benennt nur die relevanten Felder und Audio-Dateien. Diese Spiele sind von uns zum Teil noch nicht vollständig verstanden.
- Binäre Programme. Dies sind Maschinenprogramme, die direkt auf dem Prozessor des Stiftes ausgeführt werden. Auch diese sind von uns noch nicht vollständig verstanden.

Praktisch alle Elemente einer GME-Datei werden intern über schnöde Nummern statt über sprechende Namen angesteuert. Insbesondere kann man aus einer GME-Datei nicht mehr die Original-Dateinamen der (oft über hundert) Audio-Dateien rekonstruieren.

4.4 Wozu das `tttool`?

Um nun dein eigenes Tiptoi-Produkt zu erzeugen musst du eine solche GME-Datei erstellen. Nun ist das GME-Format ein unhandliches Binärformat, was du ohne Hilfsmittel nicht bearbeiten kannst. Genau dafür gibt es das `tttool`, welches das GME-Format lesen und schreiben kann.

Für die meisten Tiptoi-Bastler sind drei Hauptfunktionen wichtig:

1. Das `tttool` kann eine GME-Datei in seine Bestandteile – die Audio-Dateien und die Beschreibung der Logik – zerlegen. Die Logik wird dabei in einem (halbwegs) benutzerfreundlichen, textbasierten Format (der sogenannten *YAML-Datei*) abgelegt, das du direkt mit einem Texteditor bearbeiten kannst.

2. Natürlich beherrscht das `tttool` auch die andere Richtung, und kann aus einer YAML-Datei und den Audio-Dateien eine GME-Datei erstellen. Um die Entwicklung zu vereinfachen kann das `tttool` dabei fehlende Audio-Dateien mittels *Text-to-Speech* (Sprachsynthese) erzeugen.

   Warnung: Weil nicht alle Details des GME-Formats verstanden sind, können bei der Umwandlung von der GME-Datei zur YAML-Datei und zurück Teile verloren gehen, insbesondere Spiele.

3. Die OID-Codes, die zum Druck eines eigenen Tiptoi-Produktes nötig sind, können per `tttool` im PNG- oder PDF-Format erstellt werden.

Darüber hinaus verfügt das `tttool` über eine Reihe von Möglichkeiten zur Analyse von GME-Dateien, die vor allem zum Verstehen des GME-Formats nützlich sind.

Im Detail werden die einzelnen Funktionen des Tools im Kapitel "*Die tttool-Befehle*" erklärt.

KAPITEL 5

Die `tttool`-Befehle

Das `tttool`-Programm kann GME-Dateien zusammenbauen, zerlegen und die zugehörigen OID-Codes erstellen. Dazu unterstützt es eine Reihe von Unterbefehlen (`tttool assemble`, `tttool export` etc.), die wir im folgenden im Detail erklären.

Dieses Handbuch erklärt nur die Befehle, die für „normale Anwender“ relevant sind. Das Werkzeug unterstützt auch bei der Analyse von GME-Dateien (z.B. `tttool explain`), diese Features sind allerdings nur für Entwickler relevant.

5.1 GME-Datei zusammenbauen

Format:

tttool assemble *eingabe.yaml*
tttool assemble *eingabe.yaml ausgabe.gme*

Beispiel:

```
tttool assemble MeinTaschenrechner.yaml
```

Mit diesem Befehl baust du eine GME-Datei: Wenn *eingabe.yaml* eine gültige YAML-Datei, wie im Kapitel „*YAML-Referenz*“ beschrieben ist, so schreibt das `tttool` die darin enthaltene Logik, zusammen mit den Audio-Dateien, in die Datei *ausgabe.gme*.

Wenn du keinen zweiten Dateinamen angibst, so schreibt das `tttool` die Ausgabe nach *eingabe.gme*.

Wenn die Datei *eingabe.yaml* benannte OID-Codes, wie sie in *der YAML-Referenz* beschrieben sind, enthält, so vergibt `tttool` selbstständig Code-Nummern. Damit sich diese bei weiteren Aufrufen von `tttool assemble` (or `tttool oid-codes`, siehe unten) nicht ändern, merkt sich das `tttool` die Zuordnung, indem es die Datei *eingabe.codes.yaml* erzeugt. Diese solltest du nicht löschen.

5.2 OID-Codes erzeugen

Format:

tttool *Muster-Einstellungen* **oid-code** *code*
tttool *Muster-Einstellungen* **oid-code** *von-bis*
tttool *Muster-Einstellungen* **oid-codes** *eingabe.yaml*
tttool *Muster-Einstellungen* **oid-table** *eingabe.yaml*
tttool *Muster-Einstellungen* **oid-table** *eingabe.yaml ausgabe.pdf*
tttool *Muster-Einstellungen* **oid-table** *eingabe.yaml ausgabe.svg*

Mögliche Muster-Einstellungen:

--image-format PNG
--image-format PDF
--image-format SVG
--image-format SVG+PNG
--code-dim *Größe*
--code-dim *Breite***x***Höhe*
--dpi *DPI*
--pixel-size *Zahl*

Beispiel:

```
tttool oid-code 998
tttool --image-format PDF oid-code 0,50-100
tttool oid-codes MeinTaschenrechner.yaml
tttool --code-dim 20x20 oid-table MeinTaschenrechner.yaml
```

Das `tttool` kann OID-Muster in verschiedenen Formaten erzeugen – das brauchst du dann, wenn du deine eigenen Tiptoi-Produkte gestalten willst. Es versteht dazu mehrere Befehle, je nach dem woher es wissen soll, zu welche Codes es die Muster erzeugen soll, und mehrere Optionen, die steuern, wie die Muster auszusehen haben.

5.2.1 OID-Codes auswählen

Wenn du einfach nur ein bestimmtes Muster erzeugen willst, so verwendest du den `oid-code`-Befehl, und gibst das Muster an. Wenn du zum Beispiel

```
tttool oid-code 998
```

ausführst, erstellt dir `tttool` eine Datei `oid-998.png` (oder `oid-998.svg`, wenn du SVG als Format ausgewählt hast).

Du kannst auch mehrere Codes und Code-Bereiche auf einmal auswählen:

```
tttool oid-code 0,1,100-110
```

Aber oft willst du einfach alle Codes eines GME-Projektes erzeugen. Dazu kannst verwendest du `tttool oid-codes`::

```
$ tttool oid-codes example.yaml
Writing oid-42-START.png.. (Code 42, raw code 272)
Writing oid-42-REPLAY.png.. (Code 13445, raw code 9250)
Writing oid-42-STOP.png.. (Code 13446, raw code 9251)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
Writing oid-42-8065.png.. (Code 8065, raw code 3700)
Writing oid-42-8066.png.. (Code 8066, raw code 3701)
Writing oid-42-8067.png.. (Code 8067, raw code 3702)
```

Die erste Zahl im Dateinamen ist die Produkt-ID des Projekts, was dir helfen soll, die Dateien besser zuzuordnen. Wenn die YAML-Datei selbst explizit mit OID-Codes arbeitet, stehen diese auch im Dateinamen. Wenn du aber, wie in *der YAML-Referenz* erläutert, mit Code-Namen arbeitest, stehen die nachher auch im Dateinamen:

```
$ tttool oid-codes example2.yaml
Writing oid-42-START.png.. (Code 42, raw code 272)
Writing oid-42-REPLAY.png.. (Code 13445, raw code 9250)
Writing oid-42-STOP.png.. (Code 13446, raw code 9251)
Writing oid-42-conditional.png.. (Code 13447, raw code 9252)
Writing oid-42-hello.png.. (Code 13448, raw code 9253)
Writing oid-42-registers.png.. (Code 13449, raw code 9254)
```

Zuletzt kannst du auch alle Codes eines Projektes in eine einzelne PDF- oder SVG-Datei schreiben. Die Datei enthält dann eine schlichte, übersichtliche Tabelle mit Feldern für alle Codes, was sehr geschickt während der Entwicklung deines Projektes sein kann – so kannst du deine Programm-Logik schon planen und testen, bevor du dich and die grafische Gestaltung machst. Du erstellst die Tabelle einfach mit:

```
$ tttool oid-table example2.yaml
```

und findest danach eine `example2.pdf` im aktuellen Verzeichnis.

5.2.2 Datei-Formate

Das `tttool` unterstützt folgende Formate für die Muster

- PNG (mittels `--image-format PNG`) ist ein pixelbasiertes Bildformat. Es eignet sich gut wenn du dein Projekt mit einem Bildverarbeitungsprogramm wie GIMP oder Photoshop erzeugst. Achte darauf dass du das Bild nach dem Import in dein Programm nicht skalierst oder drehst, sondern allenfalls zuschneidest. PNG ist das Standardformat für `tttool oid-code` und `tttool oid-codes`, und wird von `tttool oid-table` nicht unterstützt.
- SVG (mittels `--image-format SVG`) ist ein Vektor-Format, und eigentlich sich gut für die Weiterverarbeitung in Zeichenprogrammen wie Inkscape oder Illustrator. So kann man zum Beispiel mit `tttool --image-format SVG oid-table` eine SVG-Datei mit allen Mustern erzeugen, und diese dann weiterverarbeiten. SVG wird von allen Befehlen unterstützt.
- SVG mit PNG (mittels `--image-format SVG+PNG`) ist eine Variante, bei der zwar als SVG-Dateien erzeugen werden, aber in der SVG-Datei das Muster selbst als PNG-Datei angelegt ist. Dies kann, je nach verwendetem Programm und Drucker, eventuell zu besser erkennbaren Mustern führen.
- PDF (mittels `--image-format PDF`) wird nur von `tttool oid-table` unterstützt und ist dort auch die Standardeinstellung, und eignet sich gut zum Drucken der Tabelle, jedoch nur bedingt für die Weiterverarbeitung.

5.2.3 Muster-Einstellungen

Mit folgenden Optionen kannst du das nachjustieren, wie das Muster erstellt wird – je nach Drucker funktionieren andere Einstellungen besser.

- Mit der Option `--code-dim` legst du fest, wie groß das Muster erzeugt werden soll. Du kannst entweder eine Zahl angeben, dann bekommst du ein Quadrat mit der angegebenen Seitenlänge in Millimeter, also

`--code-dim 30` für ein 3×3cm Quadrat (dies ist die Standard-Einstellung). Oder du gibst mit zwei Zahlen die Breite und Höhe an, etwa `--code-dim 210x297` für ein Muster in A4-Größe.

- Die Option `--dpi` gibt die gewünschte Auflösung des Musters an, in der im Druck üblichen Einheit Punkt-Pro-Zoll (dots per inch). Der Standardwert ist 1200 DPI, unter Umständen genügen auch 600 DPI.
- Die Option `--pixel-size` gibt an aus wievielen Pixel (im Quadrat) ein Punkt des Musters gebaut werden soll. Der Standardwert ist 2. Wenn du diese Zahl erhöhst bekommst du ein kräftigeres, schwärzeres Muster, das zwar stärker auffällt, aber vielleicht besser erkannt wird.

5.3 GME-Dateien extrahieren

Format:

tttool export *eingabe.gme*
tttool export *eingabe.gme ausgabe.yaml*
tttool media *eingabe.gme*
tttool media *eingabe.gme* **-d** *verzeichnis*

Beispiel:

```
tttool export WWW_Bauernhof.gme
tttool media WWW_Bauernhof.gme
```

Du kannst eine GME-Datei entpacken, und sowohl die Audio-Dateien als auch die die Logik in Form einer YAML-Datei extrahieren. Dies geschieht mit zwei Befehlen:

Der Befehl `tttool export WWW_Bauernhof.gme` schreibt die Logik in der GME-Datei in die Datei `WWW_Bauernhof.yaml`, bzw. in die angegebene Ausgabedatei.

Der Befehl `tttool media WWW_Bauernhof.gme` schreibt alle Audio-Dateien in der GME-Datei als separate Dateien, meist im OGG-Vorbis-Format mit Dateiendung `.ogg` in das Unterverzeichnis `media`. Du kannst auch ein anderes Verzeichnis mittels `-d` angeben, aber beachte dann die `media-path`-Einstellung in der YAML-Datei anzupassen, denn die vom `tttool export`-Befehl erstellte YAML-Datei verweist standardmäßig auf `media`.

5.4 Sprache einer GME-Datei ändern

Format:

tttool set-language *sprache datei.gme*
tttool set-language --empty *datei.gme*

Beispiel:

```
tttool set-language FRENCH WWW_Bauernhof.gme
```

Um eine sprachspezifische GME-Datei in einem Stift zu benutzen, der auf eine andere Sprache eingestellt ist, kann man die Sprache in der GME-Datei mit diesem Befehl ändern. Anders als via `export` und `assemble` bleibt so alle Funktionalität erhalten.

Typische Sprachangaben sind `GERMAN`, `FRENCH`, `RUSSIA`.

Achtung: Der Befehl überschreibt die angegebene GME-Datei.

5.5 Product-Id einer GME-Datei ändern

Format:

tttool set-product-id *id datei.gme*

Beispiel:

```
tttool set-product-id 991 WWW_Bauernhof.gme
```

Es kann manchmal nützlich sein, die Produkt-ID einer GME-Datei zu ändern, etwa um mehrere Versionen auf einen Stift zu laden.

Achtung: Der Befehl überschreibt die angegebene GME-Datei.

KAPITEL 6

YAML-Referenz

In diesem Kapitel erfährst du alle Details zur YAML-Datei, mit der du den Stift programmierst.

6.1 Das YAML-Format

Das YAML-Format im Allgemeinen wurde nicht für dieses Projekt erfunden, sondern ist gebräuchlich, um strukturierte Daten in einer menschenlesbaren Textdatei abzulegen. Eine knappe Übersicht findest du auf der Wikipedia-Seite zu YAML. Beachte, dass in YAML Einrückungen, also Leerzeichen am Anfang der Zeile, relevant sind und die Struktur der Datei beschreiben!

Die YAML-Datei bearbeitest du mit dem Texteditor deiner Wahl (Notepad, vim etc.).

6.2 YAML-Datei-Felder

Eine typische, minimale YAML-Datei für die Tiptoi-Programmierung sieht so aus:

```
product-id: 950
scripts:
  8066:
  - P(erstes_feld)
  8067:
  - P(zweites_feld)
```

Zwingend nötig ist dabei streng genommen nur das Feld `product-id`, aber in der Regel brauchst du auch `scripts` um irgendetwas Sinnvolles zu machen.

Es gibt noch eine Reihe weiterer, optionaler Felder, die im Folgenden erklärt werden. Eine etwas kompliziertere Datei könnte dann zum Beispiel so aussehen:

```
product-id: 950
comment: Ein kurzer Kommentar
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
welcome: willkommen
media-path: Audio/%s
gme-lang: GERMAN
init: $modus:=10
scripts:
   haus:
   - P(hello)
   tuer:
   - P(goodbye)
   monster:
   - P(warning)
language: en
speak:
   hello: "Hello, friend!"
   goodbye: "Goodbye"
   warning: "Watch out."
scriptcodes:
   haus: 4718
   tuer: 4716
   warning: 4717
```

6.2.1 product-id

Format: Ein OID-Code im Bereich 0 bis 999

Beispiel:

```
product-id: 950
```

Zweck: Die Produkt-ID dieses Projekts. Das Anschaltfeld des Produktes sollte mit dem hier angegebenen OID-Code bedruckt sein.

Es sollte zu jeder Produkt-ID nur eine GME-Datei auf den Stift geladen werden. Ravensburger zählt seine Produkte fortlaufend ab 1 hoch und Sprachen werden mit 999 abwärts nummeriert. Wir raten dir für dein eigenes Projekt daher eine Zahl zwischen 900 und 950 zu wählen.

6.2.2 comment

Format: Ein Textstring

Beispiel:

```
comment: "Mein Tiptoi-Produkt"
```

Zweck: Der Kommentar wird in der GME-Datei gespeichert, aber sonst ignoriert.

6.2.3 welcome

Format: Einen oder mehrere Audio-Dateinamen, durch Kommata getrennt

Beispiel:

```
welcome: hello
```

Zweck: Beim Aktivieren des Produktes werden die angegebenen Audio-Dateien abgespielt.

6.2.4 media-path

Format: Ein Dateipfad, mit `%s` als Platzhalter

Beispiel:

```
media-path: Audio/%s
```

Zweck: Gibt an, wo sich die Audiodateien befinden. Der Platzhalter `%s` wird dabei durch den in der YAML-Datei verwendeten Dateinamen ersetzt. Das Programm sucht nach allen geeigneten Dateiendungen (`.wav`, `.ogg`, `.flac`, `.mp3`).

Beispiel: Für den im `welcome: hello` angegebenen Begrüßungssound würde das `tttool` also die Datei `Audio/hello.ogg` einbinden.

Beispiel 2: Wenn du `media-path: Audio/Schatzsuche_%s` verwendest, würde bei einem `welcome: hello` die Datei `Audio/Schatzsuche_hello.ogg` eingebunden werden.

6.2.5 gme-lang

Format: Eine Sprache (`GERMAN`, `ENGLISH`, `FRENCH`...)

Beispiel:

```
gme-lang: GERMAN
```

Zweck: Das Sprach-Feld der GME-Datei. Bei eigenen Produkten gibt es in der Regel keinen Grund, dieses Feld anzugeben.

6.2.6 init

In diesem Feld werden Register (bzw. Variablen oder Speicherplätze) initialisiert. Beispielsweise werden hier Spielmodi oder Zähler auf 0 gesetzt, damit diese später mit einem vorgegebenen Wert starten können. Beispiel: Wird hier „$modus:=0 $i:=0“ geschrieben, so werden nach Aktivierung des Projekts die beiden Variablen modus und i mit 0 gestartet.

6.2.7 scripts

Format: Eine Zuordnung von OID-Codes (oder Code-Namen) zu einer Liste von Skriptzeilen.

Beispiel:

```
scripts:
  8067:
   - P(hi)
  haus:
   - $mode==3? P(welcome)
   - P(goodbye)
```

Zweck: Enthält die Logik dieses Tiptoi-Produktes und gibt für einen OID-Code an was geschehen soll, wenn du diesen Code antippst.

Statt eines konkreten OID-Codes kann auch ein Code-Name angegeben werden, siehe Abschnitt „*scriptcodes*“.

Die Skripte werden in Detail im Abschnitt „*YAML-Programmierung*“ erklärt.

6.2.8 `language`

Format: Ein Sprach-Kürzel (`de`, `en`, `fr`...)

Beispiel:

```
language: de
```

Zweck: Gibt die Sprache für die Sprachsynthese (siehe Feld `speak`) an.

6.2.9 `speak`

Format: Eine Zuordnung von Dateinamen zu Text

Beispiel:

```
speak:
   hello: "Hello, friend!"
   goodbye: "Goodbye"
   warning: "Watch out."
```

Zweck: Gibt an, welche Audiodateien das `tttool` per Text-to-Speech generieren soll, sofern es die entsprechenden Audiodateien nicht findet. Dabei wird die in `language` angegebene Sprache verwendet.

Das tttool verfügt über ein integriertes Text-to-Speech tool, welches dir erlaubt, Texte automatisch vorgelesen zu bekommen. So kannst du deine Tiptoi-Entwicklung testen, bevor du alles Nötige aufgenommen hast.

Solltest du Text-to-Speech in verschiedenen Sprachen benötigen, kannst du mehrere Abschnitte mit eigener Sprache angeben:

```
speak:
 - language: en
   hello: "Hello, friend!"
   goodbye: "Goodbye"
 - language: de
   warning: "Achtung!"
```

6.2.10 `scriptcodes`

Format: Eine Zuordnung von Codenamen zu OID-Code

Beispiel:

```
scriptcodes:
   haus: 4718
   tuer: 4716
   warning: 4717
```

Zweck: Erlaubt dir, im Abschnitt `scripts` und in `J`-Befehlen mit sprechenden Namen statt OID-Codes zu arbeiten. Bei der Erstellung der GME-Datei wird in dieser Zuordnung nachgeschlagen, welcher OID-Code verwendet werden soll.

Du kannst sprechende Namen auch ohne `scriptcodes` verwenden, in diesem Fall wählt das `tttool` die Codes selbst. Damit stets die gleichen Codes verwendet werden (und bereits gedruckte Codes weiterhin funktionieren), speichert es die Auswahl in einer Datei mit Endung `.codes.yaml`, die nur den `scriptcodes`-Eintrag enthält. Es steht dir frei, diese Zuordnung in die eigentliche YAML-Datei zu übernehmen.

Warnung: Das `tttool` arbeitet *entweder* mit Namen *oder* mit Nummern. Du kannst die beiden Varianten nicht mischen.

6.2.11 `replay`

Format: Ein OID-Code

Beispiel:

```
replay: 12159
```

Zweck: Beim Tippen auf diesen Code wird das zuletzt ausgegebene Audio nochmal ausgegeben.

Auch wenn man diesen dies nicht explizit in der .yaml-Datei notiert, weist das `tttool` einen solchen Code zu, und erzeugt ein `REPLAY`-Muster aus.

6.2.12 `stop`

Format: Ein OID-Code

Beispiel:

```
stop: 12158
```

Zweck: Beim Tippen auf diesen Code wird die Ausgabe gestoppt

Auch wenn man diesen dies nicht explizit in der .yaml-Datei notiert, weist das `tttool` einen solchen Code zu, und erzeugt ein `STOP`-Muster aus.

6.3 YAML-Programmierung

Die Logik einer Tiptoi-Programmierung steckt vor allem in den im `scripts`-Feld angegebenen Skripten. Es gibt zu jedem OID-Code ein Skript. Dieses besteht aus einer oder gegebenenfalls mehreren Zeilen, die wiederum aus Befehlen bestehen.

Das einfachste Beispiel ist also

```
scripts:
  2000: P(hallo)
```

Hier wird, wenn du den OID-Code 2000 antippst, der Befehl `P(hallo)` ausgeführt. (Die Befehle selbst werden in Kürze erklärt.)

Eine Skriptzeile kann mehrere Befehle enthalten, etwa

```
scripts:
  2000: P(hallo) P(freund) J(2001)
```

Hier werden drei Befehle nacheinander ausgeführt.

> **Warnung:** Soweit bekannt kann es zu Problemen kommen, wenn **mehr als 8** Befehle in einer Zeile stehen. Darüber hinaus interagieren manche Befehle seltsam; mehr dazu im Abschnitt „*J – Sprung*“.

Im Allgemeinen können zu einem Skript mehrere Zeilen angegeben werden:

```
scripts:
  2000:
   - $offen==1? P(willkommen)
   - $offen==0? P(finde_den_schluessel)
```

Tippst du nun Code 2000 an, wird die erste Zeile ausgeführt, deren Bedingungen alle erfüllt sind (mehr zum Programmieren mit Bedingungen im *Abschnitt zu Bedingungsbefehle*).

Statt die OID-Codes numerisch anzugeben, kannst du auch sprechende Namen verwenden, siehe Abschnitt „*scriptcodes*“.

6.3.1 Register

Viele Befehle manipulieren *Register*. Diese repräsentieren Speicherzellen, in denen im Programmverlauf Werte abgelegt und abgerufen werden können. Man könnte sie auch Variablen nennen.

Der Name eines Registers beginnt immer mit einem *$*, gefolgt von Buchstaben, Zahlen oder Unterstrichen (_). Direkt nach dem *$* muss ein Buchstabe kommen.

Alle Arithmetik auf dem Tiptoistift arbeitet mit ganzen Zahlen im Bereich 0 bis 65535. Alle Register haben zu Beginn den Wert 0, sofern du es nicht im `init`-Feld anders verlangst (siehe Abschnitt „*init*“).

Wenn du eine GME-Datei exportierst (siehe Abschnitt „*GME-Dateien extrahieren*“), so kennt das `tttool` die Namen der Register nicht. In diesem Fall werden Nummern verwendet (`$0`, `$1`...). Es gibt in der Regel keinen Grund, dies in deinen eigenen Tiptoi-Produkten so zu machen.

6.4 Befehlsreferenz

Im Folgenden werden die Befehle im Einzelnen erklärt: Wie du sie in der YAML-Datei schreibst, was sie bewirken, und was sonst so dabei zu beachten ist.

In der Format-Beschreibung werden folgende Platzhalter verwendet:

- *audio-datei*: Der Name einer Audio-Datei. Aus dem Namen wird, wie im Abschnitt „*media-path*“ beschrieben, der Dateiname der Audiodatei abgeleitet.
- *oid-code*: Die Nummer eines OID-Codes (und damit einer Skiptzeile), wenn `scriptcodes` *nicht* verwendet wird.
- *code-name*: Der Name eines OID-Codes (und damit einer Skiptzeile), wenn `scriptcodes` verwendet wird.
- *register*: Der Name eines Registers, mit *$*. Beispiel: *$mode*.
- *argument*: Entweder der Name eines Registers oder eine Zahl. Beispiele: *$mode*, *0*, *1024*.

  Der *Wert* eines Argumentes ist im ersten Fall der aktuell in dem Register gespeicherte Wert; im zweiten Fall einfach die Zahl selbst.

6.4.1 `P` – Audio abspielen

Format:

P(*audio-datei*)
P(*audio-datei*,*audio-datei*,...)

Beispiel:

```
haus:
- P(willkommen) P(zu_hause,daheim)
```

Effekt: In der ersten Form spielt der Befehl die angegebene Audio-Datei ab.

In der zweiten Form spielt der Befehl zufällig eine der angegebenen Audio-Dateien ab.

Hinweis: Wenn der Dateiname Zeichen enthält außer *a-z*, *0-9* und _, dann muss der Dateiname in Anführungszeichen stehen.

Beispiel:

```
haus:
- P("Willkommen zu Hause")
```

6.4.2 `J` – Sprung

Format:

J(*oid-code*)
J(*code-name*)

Beispiel:

```
endlos:
- P(endlose) P(kein_anschluss)
```

Effekt: Der Stift führt, _nach dem aktuellen **Skript**_, das Skript mit Code *oid-code* bzw. *code-name* (wenn *scriptcodes* verwendet wird) aus.

Es sollte nur ein `J`-Befehl pro Zeile existieren (bei mehreren wird der letzte ausgeführt).

Nach dem `J`-Befehl sollte mindests ein `P`-Befehl stehen, der ein nicht zu kurzes Audio-Schnippsel (mindestens 5ms) abspielt, sonst warten Tiptoi-Stifte (ab der zweiten Generation) beim Sprung etwa zwei Sekunden lang.

Eine stille, 5ms lange Audiodatei liegt dem `tttool` als `example/5ms.ogg` bei.

6.4.3 `:=` – Register setzen

Format:

register1 **:=** *argument*

Beispiel:

```
- $zuletzt := $aktuell  $aktuell := 5
```

Effekt: Der Wert des Registers *register* wird auf den Wert des Arguments *argument* gesetzt.

6.4.4 `+=, -=, *=, /=, %=` – Arithmetik

Format:

register **+=** *argument*
register **-=** *argument*
register ***=** *argument*
register **/=** *argument*
register **%=** *argument*

Beispiel:

```
taste_fuenf:
- $anzeige*=10 $anzeige+=5
gegner_getroffen
- $wert := 10 $wert *= $bonus $score += $wert
```

Effekt: Es wird die entsprechende Rechenoperation auf die aktuell in *register* gespeicherte Zahl und den Wert des Arguments *argument* angewandt, und das Ergebnis in *register1* abgelegt.

Es wird dabei nur mit ganzen Zahlen gerechnet. Insbesondere rundet die Division (**/=**) das Ergebnis stets ab. Wenn also Register *$x* den Wert 8 enthält und *$x/=3* ausgeführt wird, so enthält es den Wert 2.

Der Befehl **%=** berechnet entsprechend den Divisionsrest. Wenn also Register *$x* den Wert 8 enthält und *$x%=3* ausgeführt wird, so enthält es den Wert 2.

6.4.5 `Neg()` – Register negieren

Format:

Neg(*register*)

Beispiel:

```
- Neg($r)
```

Effekt: Der Wert des Registers *register* wird negiert: Aus 5 wird -5 und umgekehrt.

6.4.6 `&=, |=, ^=` – bitweise Operatoren

Format:

register **&=** *argument*
register **|=** *argument*
register **^=** *argument*

Zweck: Es wird die entsprechende bitweise Operation auf die aktuell in *register* gespeicherte Zahl und den Wert des Arguments *argument* angewandt, und das Ergebnis in *register* abgelegt.

Dabei ist **&=** das bitweise Und, **|=** das bitweise Oder, **^=** das bitweise exklusive Oder (XOR). Wenn dir das nichts sagt, brauchst du es vermutlich nicht.

6.4.7 `T` – Timer

Format:

T(*register***,***modulus* **)**

Beispiel:

```
wuerfel:
- T($wurf,6)
```

Effekt: Der Wert des Tiptoi-Zählers zu Beginn des Skriptes wird (modulo dem *modulus*) im Register *register* abgelegt.

Der Tiptoi-Stift verfügt über einen Zähler, der während der Benutzung hochgezählt wird. Er wird schneller hochgezählt, wenn mit dem Stift interagiert wird, er ist also nicht zur Zeitmessung geeignet. Man kann damit aber (einfache) Zufallszahlen bekommen. Mehr dazu im Abschnitt *Zufallszahlen*.

6.4.8 ==, >=, <=, >, <, != – Bedingungen

Format:

argument1 **==** *argument2*
argument1 **>=** *argument2*
argument1 **<=** *argument2*
argument1 **>** *argument2*
argument1 **<** *argument2*
argument1 **!=** *argument2*

Beispiel:

```
haus:
- $mode == 1? P(willkommen)
- $mode == 2? $gefunden < 3? P(finde_mehr_steine)
- $mode == 2? $gefunden == 3? P(raetsel_geloest)
```

Effekt: Bedingungsbefehle müssen stets am Anfang der Zeile stehen. Es wird der Wert des ersten Arguments entsprechend dem Vergleichsoperator mit dem zweiten Argument verglichen. Wenn alle Bedingungsbefehle einer Zeile zutreffen, dann wird die Zeile ausgeführt, sonst wird die nächste Zeile des Skriptes geprüft.

Die Operatoren sind:

Befehl	Bedeutung
==	gleich
>=	größer oder gleich
<=	kleiner oder gleich
>	echt größer
<	echt kleiner
!=	ungleich

6.4.9 Weitere Befehle

(Befehle die der normale Tiptoi-Bastler nicht braucht, aber die das `tttool` ausspuckt)

- `P*()`
- `PA*()`
- `PA*()`

- `PA()`
- `G()`
- `C`
- `?() ()`

KAPITEL 7

Tipps und Tricks

Hier kannst du zu konkreten Problemstellungen nachlesen, wie du sie lösen kannst. Dieses Kapitel lebt auch von deinen Beiträgen!

7.1 Zufallszahlen

Für Spiele und Rätsel mit dem Tiptoi-Stift ist es häufig nötig, Aktionen zufallsgesteuert auszuführen. Auch wenn bisher die Fähigkeiten des Stifts in dieser Hinsicht noch nicht ganz verstanden sind, gibt es ein paar Techniken, die du hier einsetzen kannst.

7.1.1 Zufälliges Abspielen von Audio-Dateien

Wenn es nur darum geht, zufällig eine von mehreren Audio-Dateien abzuspielen, genügt der `P()`-Befehl mit mehreren Argumenten:

```
- P(bing, plopp, peng)
```

Es ist nicht bekannt, wie der Zufallsgenerator hier funktioniert und wie gleichmäßig die Verteilung ist. Manche Anwender haben beobachtet, dass die erste Datei häufiger abgespielt wird.

(TODO: es wäre interessant zu dokumentieren, in welchen Ravensburger-Produkten das vorkommt und wie es dort angewendet wird.)

7.1.2 Timer

Der Stift verfügt über eine Art Timer, den du per Befehl `T()` abfragen kannst:

```
# Timer-Wert in $register speichern
T($register, 65535)
```

Syntax: T(*register*, *modulo*)

- *register*: Ziel der Berechnung
- *modulo*: Der Timer-Wert wird per Modulo-Operation (Teilungsrest) auf den Bereich 0–*modulo* begrenzt.

Die so erhaltenen Werte kannst du unter Umständen bereits als Zufallszahlen einsetzen, jedoch gibt es ein paar Probleme:

- Aufeinanderfolgende Abrufe des Timers liefern monoton ansteigende Werte.
- Während der Stift inaktiv ist, läuft der Timer langsamer.
- Ein zweimaliger Aufruf innerhalb der selben Anweisung liefert exakt den gleichen Wert.

Reicht der Timer also nicht aus, kannst du dir mit Pseudo-Zufallszahlen behelfen:

7.1.3 Algorithmen zur Erzeugung von Pseudo-Zufallszahlen

Als Pseudo-Zufallszahlen bezeichnet man Reihen von Zahlen, welche aus einer deterministischen Berechnung hervorgehen, und daher natürlich nicht wirklich zufällig sind, aber wie zufällig erscheinen. Indem du den Timer als Startwert (*seed*) verwendest, erhältst du eine in der Praxis nicht vorhersagbare Zahlenfolge.

Eine einfache Implementation sieht beispielsweise folgendermaßen aus:

```
random:
  - T($r,65535) $rnd+=$r $rnd*=25173 $rnd+=13849
```

Nach Aufruf von `random` befindet sich im Register `$rnd` eine Zufallszahl zwischen 0 und 65535. Der Wertebereich lässt sich durch Modulo beschränken:

```
random:
  - T($r,65535) $rnd+=$r $rnd*=25173 $rnd+=13849 $wuerfel:=$rnd $wuerfel%=6
↪$wuerfel+=1
```

In diesem Beispiel erhält `$wuerfel` einen zufälligen Wert zwischen 1 und 6.

(TODO: Angaben darüber ergänzen, wie gut dieser Algorithmus funktioniert) (TODO: weitere PRNG-Algorithmen)

7.2 Unterbrechen von Audio verhindern

Wenn man auf einen Code tippt, der einen `P()`-Befehl enthält, während der Tiptoi-Stift gerade eine Audio-Datei abspielt, dann wird normalerweise sofort die neue Audio-Datei abgespielt und das Abspiel der aktuellen Audio-Datei dadurch unterbrochen. Falls dies nicht erwünscht ist, so kannst du das Abspielen einer Audio-Datei folgendermaßen gegen solches Unterbrechen „schützen“:

```
code1:
  - $p==1?
  - $p:=1 J(_p0) P(audio1)
code2:
  - $p==1?
  - $p:=1 J(_p0) P(audio2)
...
_p0:
  - $p:=0
```

- Das Register `$p` hat den Wert 0, wenn gerade nichts abgespielt wird, und 1, während etwas abgespielt wird.
- Jedes Skript muss mit der Zeile `- $p==1?` beginnen. Falls `$p` den Wert 1 hat wird das Skript dann nicht weiter ausgeführt und ein laufendes Abspiel daher nicht unterbrochen.

- Ein Abspiel wird durch die Konstruktion `$p:=1 J(_p0) P(audio)` „geschützt“. Dies bewirkt, dass der Wert im Register `$p` auf 1 gesetzt wird und die angegebene Audio-Datei abgespielt wird. Zudem wird **nach Ende des Abspiels** zum Skript `_p0` gesprungen. Dies nutzt das (noch nicht verstandene) Zusammenspiel von `J()`-Befehl und `P()`-Befehl, wobei der `J()`-Befehl **vor** dem `P()`-Befehl stehen muss!
- Das Skript `_p0` setzt das Register `$p` wieder auf den Wert 0 zurück.

Beachte:

- Zwei aufeinander folgende `P()`-Befehle können durch die Konstruktion `J(_p0) P(sag_dies) P(dann_das)` leider **nicht** „geschützt“ werden. Tippst du während des Abspiels der ersten Audio-Datei (`sag_dies`) auf einen OID-Code, so wird die zweite Audio-Datei (`dann_das`) nicht mehr abgespielt. Zum Skript `_p0` wird aber dennoch gesprungen, so dass das Register `$p` auf den Wert 0 zurückgesetzt wird. Varianten der `P()`-Befehle wie etwa `PA(sag_dies, dann_das)` ändern daran nichts. Allerdings kann man sich behelfen, indem man eine neue Audio-Datei `sag_dies_dann_das` erstellt und nur einen Befehl `P(sag_dies_dann_das)` zum Abspielen nutzt.
- Während der Tiptoi-Stift eine Audio-Datei abspielt können durchaus OID-Codes angetippt werden und auch einfache arithmetische Befehle ausgeführt werden (z.B. den Wert in einem Register erhöhen), ohne das Abspiel zu unterbrechen. Es dürfen nur keine `P()`-Befehle oder `J()`-Befehle ausgeführt werden.
- Du kannst auch Skripte haben, die selbst „geschützte“ Abspiele unterbrechen (etwa ein „Stop“-Skript):

  ```
  stop:
    - $p:=0 P(nichts)
  ```

  Du musst dann aber darauf achten, das Register `$p` wieder auf 0 zurück zu setzen.

7.3 Hintergrundmuster

Es kann optisch schöner sein, wenn nicht nur die aktiven Bereiche eines Tiptoi-Werkes mit OID-Codes versehen sind, sondern alle. Dazu kannst du ein neutrales Muster verwenden, dass vom Tiptoi-Stift einfach ignoriert wird, und das auch ein laufendes Skript nicht unterbricht. Dieses Muster erzeugst du mit:

```
$ ./tttool oid-code --raw 65535
```

Stichwortverzeichnis