Prof. Dr. Stefan Brass
Institut für Informatik
MLU Halle-Wittenberg

14.02.2025

Hinweise zum SQL-Lernspiel von Tom Schindler

Vorbemerkung

Dieses Lernspiel ist im Rahmen einer Bachelorarbeit entstanden. Am Spielprinzip — Textadventure mit SQL-Schnittstelle — wird in der Datenbank-Gruppe schon länger gearbeitet, und es gab auch einige frühere Abschlussarbeiten. Die Bachelorarbeit von Herrn Schindler baut insbesondere auf der Masterarbeit von Frau Isabell Rösner auf, von der große Teile der Geschichte stammen.

Wir würden dieses Spiel jetzt gern mit Ihnen ausprobieren, um „Input“ für die weitere Entwicklung zu bekommen. Wir haben das auch von mit dem früheren Spiel gemacht, und einige wichtige Verbesserungsvorschläge sind in diese Neuentwicklung eingeflossen. Wir arbeiten auch schon an einer dritten Version mit Webschnittstelle, die von Ihrem Feedback zur aktuellen Version profitieren könnte.

Es gibt schon andere bekannte SQL-Lernspiele, z.B. „SQL Island“:

> [https://sql-island.informatik.uni-kl.de/].

Das ist aber im Prinzip nur eine rein sequentielle Folge von Übungsaufgaben, die in eine Geschichte verpackt sind.

Ich halte das hier zugrundeliegende Spielprinzip für spannender: Es bietet mehr Freiheiten und einen ganz individuellen Spielablauf. Allerdings ist die Verwendung von SQL nur teilweise erzwungen. Man kann es mehr oder weniger auch als klassisches Textadventure spielen, aber es werden die wesentlichen SQL-Befehle angezeigt. Gelegentlich muss man aber auch selbst einen SQL-Befehl verwenden. In zukünftigen Versionen sind Rätsel geplant, die sich nur mit SQL lösen lassen.

Spielprinzip

Bei einem Adventurespiel steuert man eine Spielfigur durch eine Spielwelt, die aus mit einander verbundenen „Räumen“ besteht. „Räume“ sind einfach Orte/Positionen im Spiel: Auch eine Lichtung im Wald kann ein „Raum“ sein. In diesen Räumen gibt es Gegenstände (Objekte), die man einsammeln kann, und die später eventuell nützlich sind, um Hindernisse zu überwinden. Man kann sich auch mit der Spielfigur identifizieren und mit den Steuerbefehlen ausdrücken, was man in der virtuellen Spielwelt tun würde.

In der Spielwelt gibt es auch andere Spielfiguren („Non-Player-Characters“ oder NPCs), die dem Spieler Aufgaben geben („Quests“), und bei Erfüllung der Aufgabe meist einen nützlichen Gegenstand als Belohnung übergeben. Man kann auch (sehr eingeschränkt)

mit ihnen kommunizieren. Je nach Spiel kann es auch Monster geben, mit denen man kämpfen muss.

Ein Adventurespiel hat viele Daten, die am besten in einer Datenbank gehalten werden. Z.B. gibt es eine Tabelle mit allen Räumen des Spiels. Bei klassischen Adventurespielen hat der Spieler keinen direkten Zugriff auf die Datenbank, und kommuniziert mit der Spiel-Software über Befehle wie „`go north`“. Da es sich in unserem Fall aber um ein SQL-Lernspiel handelt, kann man auch mit SQL-Befehlen auf die Datenbank zugreifen. Allerdings hat man keinen Administrator-Zugriff (außer, man ist im Besitz eines magischen Objektes zum Mogeln). Deswegen kann man z.B. nicht gleich alle Räume im Spiel anzeigen, sondern hat nur Zugriff auf eine Sicht, die lediglich bereits besuchte Räume enthält. Die Überwindung von Hindernissen und Entdeckung neuer Bereiche der Spielwelt macht ja gerade einen Teil des Spielspaßes aus.

Für einige Tabellen hat man auch UPDATE-Rechte. Z.B. hat die Tabelle `PLAYER` eine Spalte `PLAYER_IN_ROOM`, welche eine ID des aktuellen Raumes enthält. Man kann sich also im Spiel bewegen, indem man `PLAYER_IN_ROOM` verändert. Nun soll man sich aber nicht beliebig teleportieren können — das würde ja jegliche Hindernisse aushebeln und den Spielspaß runinieren. Deswegen gibt es einen Trigger, der bei solchen Updates ausgeführt wird, und prüft, ob der Raum nach den Regeln des Spiels erreichbar war (im wesentlichen also ein Nachbarraum des aktuellen Raums ist). Falls nicht, bewirkt der Trigger die Rücknahme des Updates. Man kann also nur Updates durchführen, die die Spiellogik akzeptiert.

Es wäre aber auf die Dauer etwas mühsam, wenn man sich im Spiel nur mit `UPDATE`-Befehlen bewegen könnte. Der Befehl hätte immer wieder die gleiche Struktur, so dass man nichts mehr lernt, aber er wäre doch etwas länglich:

```
UPDATE PLAYER SET PLAYER_IN_ROOM = 'r2'
```

Wenn man die Raum-ID `r2` erst mit einer Anfrage ermitteln muss, wäre es noch umständlicher. Deswegen versteht das Spiel auch klassische Adventure-Befehle, wie z.B.

```
go north
```

Es wird dann der ausgeführte SQL-Befehl angezeigt, so dass man immerhin durch das Lesen dieser Anweisung noch SQL lernt.

Damit man aber zumindest gelegentlich einen SQL-Befehl eingeben muss, gibt es im Spiel von Herrn Schindler die „Action Points“: Jeder klassische Adventure-Befehl kostet einen „Action Point“. Jedes SQL-Kommando bringt dagegen zwei solche Punkte. Auf lange Sicht muss also jede dritte Eingabe eine SQL-Anfrage oder ein SQL-Update sein. Man startet allerdings mit 10 „Action Points“, so dass man die Gelegenheit hat, sich erstmal einige SQL-Entsprechungen von klassischen Adventure-Befehlen anzeigen zu lassen. Damit man nicht allzu einfach an die Punkte kommt, liefert eine Wiederholung von einem der letzten zehn SQL-Befehle keine neuen „Action Points“.

Es gibt auch kleine Erleichterungen durch die Verwendung von SQL: Während man mit den `go`-Befehlen nur in einen direkten Nachbarraum kommt, kann man sich mit SQL zwischen allen schon bekannten Räumen teleportieren (da man sich an den Weg erinnert).

Vor allem aber erlauben SQL-Anfragen, präzise Daten der Gegenstände zu sehen, die aus den Beschreibungen nicht ganz so offensichtlich hervorgehen. Z.B. möchte man in den Endkampf ja mit der bestmöglichen Waffe gehen.

Voraussetzungen

- Das Spiel ist unter folgender Webadresse verfügbar:

  [https://users.informatik.uni-halle.de/~brass/db24/sql_lernspiel.zip]

- Entpacken Sie das ZIP-Archiv. Es enthält folgende Dateien:
  - `sql_lernspiel/sql_lernspiel.jar`
  - `sql_lernspiel/database/database.script`
  - `sql_lernspiel/database/database.properties`
  - `sql_lernspiel/help.txt`
- Es wird das Datenbanksystem HSQLDB

  [http://hsqldb.org/]

  benutzt, das als Bibliothek in die jar-Datei eingebunden ist. Sie brauchen es also nicht getrennt zu installieren.
- **Warnung:** Die „`*.script`“ Datei enthält SQL-Befehle, um die Datenbank beim Start anzulegen. Sie nehmen sich den Spielspaß, wenn Sie diese Datei lesen.
- Nötig ist Java 16 (oder neuer). Falls Sie noch kein Java auf Ihrem Rechner installiert haben, können Sie sich das OpenJDK z.B. von folgender Webseite herunterladen:

  [https://adoptium.net/]

  Die offizielle Java-Implementierung von Oracle gibt es hier:

  [https://www.java.com/de/download/].

  Von dieser Java-Version verlangen einige Nutzungen eine kostenpflichtige Lizenz. Wenn Sie das Lernspiel auf Ihrem eigenen PC betreiben, sollte es sich aber um „Personal Use“ handeln, der frei ist. Dennoch sollten Sie die Lizenzbedingungen besser selbst lesen.
- Das Spiel ist in Englisch.

Start des Programms

- Das Programm wird gestartet mit

  ```
  java -jar sql_lernspiel.jar
  ```

Es ist wichtig, dass die Datenbank-Dateien vom aktuellen Verzeichnis aus in

```
database
```

gefunden werden. Sie müssen das Spiel also im Verzeichnis `sql_lernspiel` starten.

- Wenn Sie das Programm gestartet haben, erscheint ein großes Fenster mit den Knöpfen „New Game“, „Continue“ und „Exit“. Sie drücken natürlich „New Game“.
- Dann werden Sie nach einem „Player Name“ gefragt. Hier können Sie irgendeinen Namen eingeben. Die Datenbank ist eigentlich für mehrere Spieler gemacht, aber die Sichten zeigen Ihnen nur Ihre eigenen Daten an.
- Sie können auch ein Spiel auswählen, aber im Moment gibt es nur das Spiel „g1“ (was auch schon ausgewählt ist).
- Dann drücken Sie den Knopf „Start New Game“ (Wenn das Programm hier hängt, wurde die Datenbank nicht gefunden.).
- Nun erscheint die eigentliche Spiel-Fenster.

Spiel-Fenster

- Die rechte Hälfte des Spiel-Fensters enthält die Ausgaben des Spiels. Auch Ihre Eingaben werden dort gelistet. Es ist also eine Art „Protokoll“ des Spiels. Sie können sich den bisherigen Spielverlauf anzeigen lassen, indem Sie nach oben scrollen.

  Wenn Sie das Spiel frisch gestartet haben, steht im Ausgabefeld rechts eine Begrüßung und eine kurze Anleitung. Dann folgt eine Einführung in die Geschichte. Anschließend wird der Startraum beschrieben.

  – Der Präfix `INV` steht vor unbeweglichen Dingen im aktuellen Raum (Requisiten wie z.B. ein Baum). Es ist gewissermaßen das „Inventar“ des Raums.

    Diese Abkürzung ist etwas unglücklich, weil man mit dem Adventure-Befehl „`inv`“ (für „inventory“) die Objekte auflisten kann, die man bei sich trägt (besitzt). Man darf das nicht verwechseln. Die Objekte, die man besitzt, werden nicht automatisch angezeigt.
  – Der Präfix `OBJECT` steht vor Gegenständen, die im Raum herumliegen und tragbar sind. Man könnte sie also mitnehmen. Im Startraum gibt es das leider nicht.
  – Der Präfix `NPC` steht vor Informationen über Personen im aktuellen Raum.
  – Der Präfix `PATH` steht vor Informationen über Ausgänge aus dem aktuellen Raum, also Verbindungen in andere Räume. Nicht alle diese Pfade sind auch passierbar. Z.B. führt aus dem Startraum nach Süden ein Pfad aus dem Spiel heraus, aber man braucht einen Schlüssel, um diesen Weg gehen zu können (und damit das Spiel zu beenden). Leider gibt es in diesem Fall keine Erklärung, warum man nicht nach Süden gehen kann. Das steht auf der TODO-Liste für die nächste Version.

- Links unten ist das Eingabefeld. Sie können dort SQL-Kommandos und Adventure-Befehle eingeben. Die Eingabe wird erst ausgeführt, wenn Sie auf den „Execute“-Knopf darunter klicken. Daher können Sie auch mehrzeilige SQL-Befehle problemlos eingeben. Bei Adventure-Befehlen wäre es schön gewesen, wenn „Return“/„Enter“ den Befehl sofort ausgeführt hätte, aber der Autor hat diese Unterscheidung nicht vorgenommen. Er hat allerdings das Kürzel „⟨Ctrl⟩+Enter“ für „Execute“ eingeführt. Mit „⟨Ctrl⟩+↑“ kommen Sie zum vorigen Kommando (man kann das auch mehrfach drücken, um die Historie zu durchlaufen). Mit „⟨Ctrl⟩+↓“ kommen Sie entsprechend aus der Historie wieder zurück.
- Links oben ist ein Mehrzweck-Ausgabebereich. Sie können mit den Tabs wählen, was Sie sehen wollen. Z.B. könnte der „Help“-Tab nützlich sein, dort gibt es auch eine Liste der Tabellen, auf die Sie Zugriff haben.
- Unter „Table“ werden die letzten zwei Ausgabe-Tabellen angezeigt. Sie können die Aufteilung der Spalten ändern, indem Sie den Trennstrich mit der Maus verschieben. Bei großen Tabellen werden auch Scrollbars angezeigt (vertikal und horizontal). Die Tabellendaten werden weniger schön, aber möglicherweise praktischer auch im Ausgabebereich rechts angezeigt: Bei langen Texten in den Tabellen können Sie den Text dort sofort vollständig lesen, dafür gibt es keine Spaltenaufteilung, sondern nur einen Trennstrich „`|`“. Sie können mit dem Befehl „`tablestyle 2`“ noch eine alternative Darstellung wählen, die etwas übersichtlicher ist, aber mehr Platz kostet (ein Datenwert pro Ausgabezeile). Mit `tablestyle 1` kommen Sie zurück zur Default-Darstellung. Beides kostet einen „Action Point“. Sie können alternativ auch die Spalte `PLAYER_TABLE_STYLE` in der Tabelle `PLAYER` aktualisieren (damit würden Sie sogar zwei „Action Points“ verdienen).
- „Map“ zeigt ihnen eine Karte mit den bisher bekannten Räumen an. In den schwarzen Rechteck steht jeweils die Raum-ID (z.B. „`r1`“ für den Start-Raum). Der rote Punkt markiert den aktuellen Raum (in dem die von Ihnen gesteuerte Spielfigur gerade ist). Sie können dort auch sehen, wo es noch Verbindungen zu Räumen gibt, die Sie bisher noch nicht erkundet haben. Das wäre natürlich auch eine interessante SQL-Anfrage.
- „History“ enthält eine Liste Ihrer bisherigen Eingaben.
- „Command“ bietet ein Menu mit Adventure-Befehlen. Diese Befehle gibt es aber auch oben in der Menuleiste.

Räume

- Bei Textadventure-Spielen bewegt man „sich“ bzw. die Spielfigur durch eine Spielwelt, die aus mit einander verbundenen Spielpositionen besteht. In Textadventure-Spielen heißen die Spielpositionen „Räume“. Auch ein Campingplatz oder eine Lichtung im Wald wird als „Raum“ bezeichnet.
- Ein Befehl zur Bewegung der Spielfigur ist z.B.

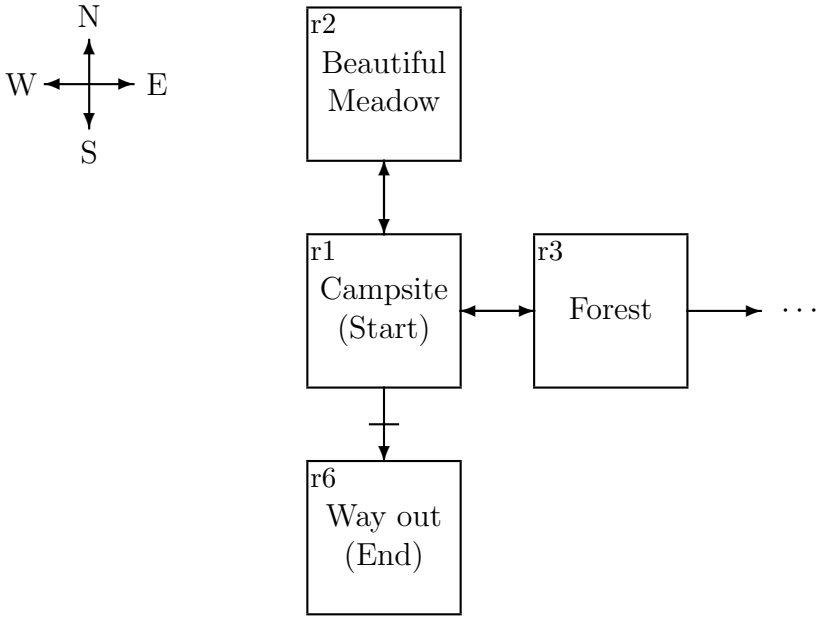
```
go north
```

Man kann das auch abkürzen zu

```
go n
```

Leider unterstützt die aktuelle Version des Spiels noch nicht die sonst auch übliche Abkürzung zu „`n`“.

- Mögliche Richtungen für die Bewegung sind: `n` („north“), `e` („east“), `s` („south“), `w` („west“), `d` („down“), `u` („up“), `ne` („northeast“), `nw` („northwest“), `se` („southeast“), `sw` („southwest“). Im Spiel werden aber nur die vier Himmelsrichtungen `n`, `e`, `s` und `w` verwendet.
- Jeder Raum hat Ausgänge nur in bestimmten Richtungen. Man kann sich nur in den Richtungen bewegen, die in der Raumbeschreibung unter „`PATH`“ aufgelistet sind. In einem stimmungsvollen Spiel gibt es Gründe, warum man sich in die Richtung nicht bewegen kann, z.B. eine hohe Mauer, ein reißender Fluss, der Wald ist zu dicht, u.s.w.
- Erfahrene Adventure-Spieler zeichnen sich eine Karte des Spiels, d.h. einen Graphen mit Knoten für die Räume, und Kanten für die Verbindungen zwischen den Räumen. Die Kanten können mit Himmelsrichtungen markiert werden. Meist versucht man aber, die Räume wie auf einer normalen Landkarte anzuordnen.



- Dieses Spiel-Programm zeichnet automatisch eine Karte (im Tab „Map“ links oben). Sie enthält allerdings nur die Raum-IDs, nicht die Bezeichnungen der Räume.
- Es ist möglich, dass ein Weg versperrt ist, und man einen bestimmten Gegenstand (s.u.) benötigt, um passieren so können.
- Eine zusätzliche Beschreibung des aktuellen Raums kann mit dem Befehl `examine` angezeigt werden, z.B.

```
examine campsite
```

Das geht aber nur, wenn man gerade im Raum mit dem Namen „`campsite`“ ist.

- Auch Ausgänge haben eine zusätzliche Beschreibung, die man mit „`examine`“ und der Himmelsrichtung abfragen kann, z.B.

```
examine s
```

Verwendung von SQL, Tabellen für Räume

- Weil es in dem Spiel ja eigentlich darum geht, die Datenbank-Sprache SQL zu üben, sind die Informationen des Spiels in Datenbank-Tabellen gespeichert. Der Spieler kann auch SQL-Anfragen und Updates eingeben. Tatsächlich sind die Adventure-Befehle wie „`go north`“ nur Abkürzungen für SQL-Befehle, die dem Spieler auch angezeigt werden. So kann man auch, wenn man gar kein SQL nutzt, zumindest passiv SQL-Befehle sehen und hoffentlich auch lesen. Man hat aber Vorteile, wenn man zumindest gelegentlich auch aktiv einen SQL-Befehl nutzt.
- Die Tabelle (oder eigentlich Sicht) mit den Räumen ist

```
ROOM(ROOM_ID, ROOM_NAME, ROOM_TEXT, ROOM_DESCRIPTION,
     ROOM_IS_DARK)
```

Da das Entdecken neuer Räume Teil des Spielspaßes ist, enthält die Tabelle (Sicht) nur die bereits besuchten Räume. Es gibt natürlich auch eine Tabelle mit den Daten aller Räume, aber diese ist für den Spieler normalerweise nicht zugänglich.

- Man kann sich die ganze Tabelle anzeigen lassen mit der folgenden einfachen SQL-Anfrage:

```
SELECT * FROM ROOM
```

Das Programm versteht an dem Schlüsselwort „`SELECT`“ ganz am Anfang der Eingabe, dass es sich um eine SQL-Anfrage handelt, und führt diese dann aus. Die Tabelle wird sowohl im normalen Ausgabefenster (rechts) ausgegeben, als auch (als Tabelle formatiert) im „Table“ Tab links.

Die Groß-/Kleinschreibung von SQL-Befehlen spielt wie üblich keine Rolle (außer in Zeichenketten-Konstanten).

- Der aktuelle Raum, in dem sich der Spieler befindet, steht in der Tabelle

```
PLAYER(PLAYER_ID, PLAYER_NAME, PLAYER_LIFE_POINTS,
       PLAYER_ATTACK, PLAYER_ATTACK_SPEED, PLAYER_MONEY,
       PLAYER_ACTIONS_LEFT, PLAYER_IN_ROOM → ROOM,
       PLAYER_SHOW_STATEMENTS, PLAYER_TABLE_STYLE,
       PLAYER_THEME_STYLE, PLAYER_FONT_SIZE,
       PLAYER_PLAYS_GAME, PLAYER_LAST_LOGIN)
```

Folgende Anfrage liefert also den aktuellen Raum:

```
SELECT PLAYER_IN_ROOM FROM PLAYER
```

- Auch Update-Befehle sind möglich:

  ```
  UPDATE PLAYER SET PLAYER_IN_ROOM = 'r2'
  ```

  Der Raum muss dabei ein Nachbarraum des aktuellen Raums sein, oder schon bekannt (in der `ROOM`-Tabelle gelistet). Ansonsten gibt es eine Fehlermeldung und die Änderung wird zurückgenommen.

- Aus Datenbank-Sicht ist das Spielprinzip also Folgendes:
  - Der Spieler führt Updates an der Datenbank durch (entweder direkt mit SQL oder indirekt über einen Adventure-Befehl).
  - Diese Updates lösen Trigger aus (Programmcode wird ausgeführt).
  - Illegale Updates, wie etwa die beliebige Teleportierung, sind nicht möglich (der Programmcode löst eine Exception aus, und dadurch kann die Transaktion nicht mit `COMMIT` beendet werden, so dass das DBMS den Zustand vor der Transaktion wieder herstellt).
  - Legale Updates können dazu führen, dass der Trigger zusätzliche Updates am Zustand der Datenbank vornimmt. Z.B. wird beim ersten Betreten eines neuen Raums ein Eintrag in einer Tabelle `ROOMS_VISITED` vorgenommen (für den Spieler nicht zugreifbar). Diese Tabelle wird in der Sicht `ROOM` verwendet, um nur die Daten von schon besuchten Räumen anzuzeigen. Obwohl man also direkt nur ein Update in der Tabelle `PLAYER` vorgenommen hat, erscheint anschließend eine neue Zeile in der Tabelle (Sicht) `ROOM`.
  - Tabellen oder Spalten, die grundsätzlich nicht geändert werden können, sind ggf. schon durch die Zugriffsrechte der Datenbank geschützt. Das vereinfacht den Programmcode, da keine Trigger für unmögliche Updates geschrieben werden müssen.
  - Im Gegensatz zur früheren Version ist die Spiellogik jetzt vollständig in der Datenbank implementiert. Das Spielprogramm ist nur ein Client zur Datenbank, der eine für diesen Zweck besonders angepasste Benutzerschnittstelle bietet. Man könnte das Spiel aber auch direkt mit einer SQL-Schnittstelle zur Datenbank spielen. Das wäre nur etwas umständlicher.
  - Zugriffsrechte, Trigger und View Updates werden in der Vorlesung „Datenbank-Programmierung“ im Sommersemester besprochen.

- Die Verbindungen der Räume stehen in der Tabelle

  ```
  PATH(PATH_FROM_ROOM → ROOM, PATH_FROM_ROOM_NAME,
       PATH_TO_ROOM, PATH_DIRECTION,
       PATH_TYPE, PATH_DESCRIPTION, PATH_NEEDS_OBJECT)
  ```

Tatsächlich handelt es sich um eine Sicht (virtuelle Tabelle), so dass die redundante Spalte „PATH_FROM_ROOM_NAME“ kein Problem ist. Sie vereinfacht die Anfragen etwas. Die Spalte „PATH_TO_ROOM“ sollte eigentlich ein Fremdschlüssel sein, der auf die Raum-Tabelle ROOM verweist. Falls der Ziel-Raum aber bisher noch nie betreten wurde, fehlt sein Eintrag in ROOM. In der zentralen Datenbank des Spiels mit den vollständigen Informationen ist es natürlich ein Fremdschlüssel. Da PATH nur eine Sicht (virtuelle Tabelle) ist, kann man ohnehin keine Fremdschlüssel dafür deklarieren. Man könnte aber natürlich feststellen, dass die entsprechende Bedingung eine logische Folgerung aus der Sicht-Definition und den Fremdschlüsseln auf den Basistabellen ist. Allerdings das gilt hier nur für PATH_FROM_ROOM.

- Ein interessante SQL-Anfrage wäre zum Beispiel, nach bisher nicht besuchten Räumen zu suchen. Wenn man in dem Spiel nicht weiter kommt, stellt sich ja die Frage, ob man vielleicht irgendwo einen Ausgang übersehen hat. Dazu muss man nach Einträgen in der Spalte PATH_TO_ROOM suchen, die (noch) nicht in der Tabelle ROOM als ROOM_ID auftreten.

Historie, Verben, Hilfetexte

- Es gibt eine Tabelle

  ```
  HISTORY(COMMAND_ID, COMMAND_GIVEN, COMMAND_AS_ACTION),
  ```

  in der die Eingaben abgelegt werden. Man kann sich die Spiel-Historie folgendermaßen ansehen:

  ```
  SELECT * FROM HISTORY ORDER BY COMMAND_ID
  ```

- Die Daten für den Befehl `help` stehen in der Tabelle

  ```
  ACTIONS_ENGLISH(ACTION_ID, ACTION, ACTION_SQL_COMMAND,
                  ACTION_DESCRIPTION, ACTION_TYPE).
  ```

  Es gibt auch eine Tabelle ACTIONS_DEUTSCH. Man könnte mit einer LIKE-Bedingung in SQL nach einem Befehl suchen. Das Spiel versteht auch den Befehl

  geh n

  Da die Ausgabe-Texte aber immer auf Englisch sind, hilft das momentan nicht viel. Aber ein Spielautor könnte natürlich die Spiel-Daten auch in Deutsch verfassen. Im Prinzip kann man ein neues Spiel entwickeln, indem man die Inhalte der Datenbank austauscht.

- Auch ACTIONS_DEUTSCH und ACTIONS_ENGLISH sind nur Sichten. Intern gibt es Tabellen ACTIONS und ACTIONS_SEQUENCES mit noch weiteren Informationen, die aber leider für den Spieler nicht zugänglich sind. Wundern Sie sich also nicht, wenn noch andere SQL-Befehle in der Ausgabe erscheinen als die, die in ACTIONS_ENGLISH angezeigt werden.

Gegenstände

- Es gibt im Spiel Gegenstände (Objekte), die teils wichtig sind, um bestimmte Aktionen durchführen zu können.

- Nicht alle Gegenstände sind sofort sichtbar. Man kann Raum-Inventar (wie auch normale Gegenst'ande) mit dem Befehl „`examine`“ untersuchen, z.B.

  ```
  examine tent
  ```

  Das geht natürlich nur, wenn das Objekt, also das Zelt, im gleichen Raum wie der Spieler ist (es ist im Startraum „campsite“). Die Untersuchung hat zwei Effekte: Einerseits wird eine genauere Beschreibung des Objektes ausgedruckt. Andererseits findet man eventuell Objekte, die im untersuchten Objekt enthalten sind (also „versteckt“ waren). Im aktuellen Spiel kann nur Raum-Inventar ein „Container“-Objekt sein, das andere Objekte enthält. In der Realität (und in anderen Spielen) sind beliebige Schachtelungen möglich.

- Die Sicht `ROOM_INVENTORY` listet alle unbeweglichen Gegenstände auf, die in besuchten Räumen enthalten sind:

  ```
  ROOM_INVENTORY(INVENTORY_ID, INVENTORY_NAME,
                 INVENTORY_ARTICLE, INVENTORY_DESCRIPTION,
                 INVENTORY_IN_ROOM → ROOM,
                 INVENTORY_NEEDS_OBJECT).
  ```

- Natürlich möchte man Gegenstände auch mitnehmen, um sie später bei passender Gelegenheit verwenden zu können. Das geht nicht für das Zelt, weil es zu groß und schwer ist. Die Basis-Daten aller beweglichen Gegenstände im Spiel (Objekte) stehen in der Tabelle `OBJECT`:

  ```
  OBJECT(OBJECT_ID, OBJECT_NAME, OBJECT_DESCRIPTION,
         OBJECT_EATABLE, OBJECT_EFFECT_WHEN_EATEN,
         OBJECT_SWITCHABLE,
         OBJECT_MAGICAL, OBJECT_EFFECT_WHEN_MAGICAL,
         OBJECT_ATTACK)
  ```

  Natürlich werden wieder nur Objekte angezeigt, die man im Prinzip schon kennt oder kennen könnte. Z.B. sind auch Objekte dabei, die der Besitzer des Campingplatzes zum Verkauf anbietet.

- Weil die Datenbank eigentlich für mehrere Spieler gemacht ist, und jeder seine eigene Kopie der Objekte hat, sind die änderbaren Objekt-Eigenschaften in eine eigene Tabelle ausgelagert:

```
OBJECT_PROPERTIES(OBJECT_ID → OBJECT, OBJECT_VALUE,
                  OBJECT_EATEN, OBJECT_SWITCHED_ON,
                  OBJECT_MAGICAL_EFFECT_USED,
                  OBJECT_IN_CONTAINER° → ROOM_INVENTORY,
                  OBJECT_IN_ROOM° → ROOM,
                  OBJECT_FROM_NPC° → NPC,
                  OBJECT_IN_INVENTORY, OBJECT_CHEATS_LEFT)
```

Für `OBJECT_PROPERTIES` hat man auch `UPDATE`-Rechte, während man für `OBJECT` nur Leserechte hat. Diese Aufteilung bedeutet aber, dass man häufig einen Join von `OBJECT` und `OBJECT_PROPERTIES` braucht. Das ist im Prinzip gut, um SQL zu lernen. Auf der anderen Seite greift man dann vielleicht doch eher zu einem Adventure-Befehl, weil die entsprechende SQL-Anweisung relativ komplex ist.

Aber die Tabellen enthalten Daten, die aus der Objektbeschreibung nicht so klar hervorgehen. Z.B. bedeutet „`lpup 5`“ in der Spalte `OBJECT_EFFECT_WHEN_EATEN`, dass sich die Lebenspunkte um 5 erhöhen, wenn man das Objekt isst. Das kann man tun, indem man die Spalte `OBJECT_EATEN` in der Tabelle `OBJECT_PROPERTIES` auf „`true`“ setzt. Dazu muss man das Objekt natürlich erst besitzen.

Es ist vermutlich etwas zu großzügig, dass die Effekte beim Essen bzw. bei magischer Verwendung in der Tabelle angezeigt werden. Wer SQL kann, ist hier im Vorteil, da er z.B. giftige Objekte vorher erkennen kann (wenn man die Codes der Effekte richtig rät). Andererseits reduziert das die Spannung etwas.

- Momentan ist der Effekt von eingeschalteten Objekten, dass sie Licht geben, was für dunkle Räume wichtig ist. Ein typischer schaltbarer Gegenstand ist also eine Taschenlampe. Später sollte es wohl auch eine Spezifikation des Effekts von eingeschalteten Objekten geben. Mit „`use`“ kann man passende Gegenstände einschalten, aber es tut natürlich auch ein Update der Spalte `OBJECT_SWITCHED_ON` von `OBJECT_PROPERTIES`.

- Man kann einen Gegenstand, z.B. eine Wasserflasche „`water bottle`“, nehmen mit folgendem Befehl:

  ```
  take water bottle
  ```

  Im wesentlichen wird dabei die Spalte `OBJECT_IN_INVENTORY` auf `true` gesetzt (in der Tabelle `OBJECT_PROPERTIES`). Gleichzeitig müssen aber die Spalten `OBJECT_IN_ROOM` und `OBJECT_IN_CONTAINER` auf `NULL` gesetzt werden, da ein Objekt immer nur in einem Ort sein kann.

- Man kann einen Gegenstand wieder ablegen mit einem Befehl der Art

  ```
  drop water bottle
  ```

- Alle Gegenstände, die man aktuell mit sich herumträgt, zeigt folgender Befehl an:

  ```
  inv
  ```

Dies druckt den `OBJECT_NAME` aus der Tabelle `OBJECT` von allen Objekten aus, die in der Tabelle `OBJECT_PROPERTIES` mit `OBJECT_IN_INVENTORY=true` gekennzeichnet sind.

- Manche Gegenstände sind essbar, man kann z.B. folgenden Befehl eingeben:

  ```
  eat bag of chips
  ```

  Der Gegenstand ist dann allerdings verbraucht. Falls er später im Spiel noch wichtig gewesen wäre, hat man keine Chance mehr, das Spiel erfolgreich zu beenden. (Allerdings hätte im konkreten Spiel der Besitzer des Campingplatzes noch eine zweite Tüte Chips zu verkaufen.) Auch die „`water bottle`“ muss man mit „`eat`“ trinken. Ein „`drink`“-Befehl steht auch noch auf der TODO-Liste. Mit SQL geht es so:

  ```
  UPDATE OBJECT_PROPERTIES
  SET    OBJECT_EATEN = 'true'
  WHERE  OBJECT_ID = 'o1'
  ```

- Ein Vorteil der direkten Verwendung von SQL ist, dass man eine Anweisung gleich auf mehrere Objekte anwenden kann, z.B. alle essbaren Objekte essen, die man in seinem „Inventory“ hat.
- Man kann magische Gegenstände verwenden mit dem Befehl „`use`“. In SQL kann man entsprechend `OBJECT_MAGICAL_EFFECT_USED` auf `true` setzen.

Andere Personen im Spiel, Aufgaben

- Es gibt im Spiel „Non-Player Characters“, kurz „NPCs“. Gleich im Startraum sind z.B. der Wanderer „`Mark`“ und der Besitzer des Campingplatzes „`Mr Cook`“.
- Personen, die man schon getroffen hat, stehen in der Tabelle `NPC`:

  ```
  NPC(NPC_ID, NPC_NAME, NPC_TYPE, NPC_TEXT, NPC_DESCRIPTION,
      NPC_IN_ROOM → ROOM, NPC_IS_MERCHANT)
  ```

- Man kann mit diesen Spielpersonen sprechen:

  ```
  talk to mark
  ```

  Manchmal bekommt man so eine nützliche Information, manchmal wird auch eine Aufgabe („Quest“) gestartet. Wenn man eine Aufgabe erfüllt hat, spricht man wieder mit der Person. Man bekommt dann eine Belohnung, meist einen wichtigen Gegenstand.

- Ansonsten sind die Personen sehr einfach programmiert, sie sagen immer dasselbe und bewegen sich auch nicht.

  Etwas mehr „künstliche Intelligenz“ bei den NPCs wäre aber schon für sich eine Abschlussarbeit. Tatsächlich gibt es in Autorenwerkzeugen für „Interactive Fiction“ wie Quest schon deutlich mehr Möglichkeiten als im aktuellen Spiel:

[https://textadventures.co.uk/quest]

- Aufgaben, die man bekommen hat, stehen in der Tabelle `QUEST`:

  ```
  QUEST(QUEST_ID, QUEST_TITLE, QUEST_TEXT,
        QUEST_FROM_NPC → NPC, QUEST_RECEIVED,
        QUEST_SOLVED, QUEST_REWARD_RECEIVED)
  ```

- Wenn man die Aufgabe gelöst hat, sollte man nochmal mit dem NPC sprechen, der die Aufgabe gestellt hat. Alternativ kann man die Spalte `QUEST_REWARD_RECEIVED` der Tabelle `QUEST_UPDATE` auf `true` setzen.

- Man kann NPCs auch anschauen:

  ```
  examine mark
  ```

  Dies druckt einfach die `NPC_DESCRIPTION` aus der Tabelle `NPC`. Eventuell bekommt man so zusätzliche Informationen.

- Mit `viewoffers` kann man die Angebote von Händler-NPCs im gleichen Raum anschauen. Wenn man ausreichend Geld hat, kann man Gegenstände kaufen:

  ```
  trade bag of chips
  ```

  Dies bewirkt, dass die Position des Gegenstands verändert wird. Vorher enthält `OBJECT_FROM_NPC` in der Tabelle `OBJECT_PROPERTIES` die ID des NPCs, das wird auf `NULL` gesetzt, und gleichzeitig `OBJECT_IN_INVENTORY` auf `true`. Außerdem zieht der entsprechende Trigger dem Spiel noch das Geld für den Gegenstand ab (Spalte `PLAYER_MONEY` in der Tabelle `PLAYER`). Falls der Spieler nicht genug Geld hat, wird die Änderung rückgängig gemacht.

- Man kann probieren, vor dem Kauf eine Einfügung in die Tabelle `TRADE_OFFER` zu machen:

  ```
  TRADE_OFFER(NPC_ID → NPC, OBJECT_ID → OBJECT, OFFER_VALUE)
  ```

  Natürlich wird man einen `OFFER_VALUE` wählen, der unterhalb des `OBJECT_VALUE` in der Tabelle `OBJECT_PROPERTIES` liegt. Wenn man Glück hat, wird das Angebot akzeptiert, und der `OBJECT_VALUE` geändert. Anschließend kann man den Artikel wie oben beschrieben kaufen, jetzt aber zum ggf. reduzierten Preis.

- Man kann NPCs auch ein Objekt geben:

  ```
  give john, bag of chips
  ```

  Es ist dann natürlich weg. Vielleicht zeigt sich die Person aber erkenntlich. In SQL verändert man wieder die Position des Objekts (umgekehrt wie beim Kauf).

Gegner, Kämpfe

- Es gibt auch Gegner („Monster“). Gegener, denen man schon begegnet ist, sind in der Tabelle `ENEMY` aufgelistet:

  ```
  ENEMY(ENEMY_ID, ENEMY_NAME, ENEMY_TEXT, ENEMY_DESCRIPTION,
        ENEMY_LIFE_POINTS, ENEMY_ATTACK, ENEMY_ATTACK_SPEED,
        ENEMY_IN_ROOM → ROOM, ENEMY_CURRENT_LIFE_POINTS,
        ENEMY_STORED_PLAYER_MONEY,
        ENEMY_DEFEATED, ENEMY_EFFECT_WHEN_DEFEATED)
  ```

- Die eigenen Lebenspunkte und Angriffswerte kann man in der Tabelle `PLAYER` (s.o) nachschauen. Man kann auch Monster mit `examine` anschauen und versuchen, mit `talk to` mit ihnen zu sprechen.

- Man kann Gegner angreifen:

  ```
  attack ...
  ```

  Wenn man einen Gegenstand besitzt, der als Waffe verwendbar ist, muss man das beim Angriff sagen:

  ```
  attack ... with ...
  ```

  Waffen verbessern die eigenen Angriffswerte.

- Der Angriff wird ausgeführt, indem eine Zeile in die Tabelle `ENEMY_HISTORY` eingefügt wird:

  ```
  ENEMY_HISTORY(ENEMY_ID → ENEMY, OBJECT_ID → OBJECT)
  ```

  Dabei ist `OBJECT_ID` die verwendete Waffe. Auch ein Nullwert ist möglich, wenn man mit bloßen Händen angreift. Diese Tabelle/Sicht hat keinen Schlüssel, bzw. der Schlüssel ist in der Sicht herausprojiziert. Doppelte Zeilen sind also möglich, falls man mehrfach mit dem gleichen Gegenstand angreift.

  Das Kampfsystem ist momentan noch recht einfach (auch ohne zufällige Entscheidungen).

  Natürlich werden die Monster einen Angriff nicht einfach hinnehmen, aber sie werden bisher nicht so aggressiv, dass sie immer weiter angreifen. Man kann den Kampf also jederzeit beenden, was wohl nicht besonders realistisch ist. In der Vorgängerversion lief der Kampf dagegen immer bis zum bitteren Ende, ohne dass man noch eingreifen konnte. Das wurde hier verbessert.

Der „Cheat“ Modus

- Wenn man einen bestimmten Gegenstand hat (im Spiel ist das „Big Leaf“), kann man den `cheat`-Befehl verwenden, um auf die Original-Tabellen des Spiels mit den vollständigen Daten zuzugreifen (statt der eingeschränkten Sichten). Die Original-Tabellen heißen meist wie die Sichten, aber im Plural. Z.B. bekommt man so eine Liste aller Räume des Spiels:

  ```
  cheat SELECT * FROM ROOMS
  ```

  Noch etwas zielgerichteter kann man auch nur Räume abfragen, die man noch nicht kennt:

  ```
  SELECT *
  FROM   ROOMS
  WHERE  ROOM_ID NOT IN (SELECT ROOM_ID FROM ROOM)
  ```

  Die Gesamtanzahl der Mogel-Versuche ist begrenzt (allerdings nicht stark). Man kann keine Updates ausführen, nur Anfragen.

- Tabellen für den „Cheat“-Modus sind: `GAMES`, `ROOMS`, `PATHS`, `NPCS`, `QUESTS`, `ENEMIES`, `OBJECTS`, `CONTAINERS`, `QUEST_REWARD`, `OBJECT_EFFECTS`, `TRASHS`.

Zusammenfassung: Tabellen/Sichten des Spiels

- ROOM(ROOM_ID, ROOM_NAME, ROOM_TEXT, ROOM_DESCRIPTION,
  ROOM_IS_DARK)
- PATH(PATH_FROM_ROOM → ROOM, PATH_FROM_ROOM_NAME,
  PATH_TO_ROOM, PATH_DIRECTION,
  PATH_TYPE, PATH_DESCRIPTION, PATH_NEEDS_OBJECT)
- PLAYER(PLAYER_ID, PLAYER_NAME, PLAYER_LIFE_POINTS,
  PLAYER_ATTACK, PLAYER_ATTACK_SPEED, PLAYER_MONEY,
  PLAYER_ACTIONS_LEFT, PLAYER_IN_ROOM → ROOM,
  PLAYER_SHOW_STATEMENTS, PLAYER_TABLE_STYLE,
  PLAYER_THEME_STYLE, PLAYER_FONT_SIZE,
  PLAYER_PLAYS_GAME, PLAYER_LAST_LOGIN)
- ROOM_INVENTORY(INVENTORY_ID, INVENTORY_NAME,
  INVENTORY_ARTICLE, INVENTORY_DESCRIPTION,
  INVENTORY_IN_ROOM → ROOM,
  INVENTORY_NEEDS_OBJECT).
- ROOM_INV_EXAMINE(INVENTORY_ID → ROOM_INVENTORY,
  INVENTORY_IS_EXAMINED)

- OBJECT(<u>OBJECT_ID</u>, OBJECT_NAME, OBJECT_DESCRIPTION,
  OBJECT_EATABLE, OBJECT_EFFECT_WHEN_EATEN,
  OBJECT_SWITCHABLE,
  OBJECT_MAGICAL, OBJECT_EFFECT_WHEN_MAGICAL,
  OBJECT_ATTACK)
- OBJECT_PROPERTIES(<u>OBJECT_ID</u> → OBJECT, OBJECT_VALUE,
  OBJECT_EATEN, OBJECT_SWITCHED_ON,
  OBJECT_MAGICAL_EFFECT_USED,
  OBJECT_IN_CONTAINER° → ROOM_INVENTORY,
  OBJECT_IN_ROOM° → ROOM,
  OBJECT_FROM_NPC° → NPC,
  OBJECT_IN_INVENTORY, OBJECT_CHEATS_LEFT)
- NPC(<u>NPC_ID</u>, NPC_NAME, NPC_TYPE, NPC_TEXT, NPC_DESCRIPTION,
  NPC_IN_ROOM → ROOM, NPC_IS_MERCHANT)
- QUEST(<u>QUEST_ID</u>, QUEST_TITLE, QUEST_TEXT,
  QUEST_FROM_NPC → NPC, QUEST_RECEIVED,
  QUEST_SOLVED, QUEST_REWARD_RECEIVED)
- QUEST_UPDATE(<u>QUEST_ID</u> → QUEST, QUEST_RECEIVED,
  QUEST_SOLVED, QUEST_REWARD_RECEIVED)
- TRADE_OFFER(NPC_ID → NPC, OBJECT_ID → OBJECT, OFFER_VALUE)
- ENEMY(<u>ENEMY_ID</u>, ENEMY_NAME, ENEMY_TEXT, ENEMY_DESCRIPTION,
  ENEMY_LIFE_POINTS, ENEMY_ATTACK, ENEMY_ATTACK_SPEED,
  ENEMY_IN_ROOM → ROOM, ENEMY_CURRENT_LIFE_POINTS,
  ENEMY_STORED_PLAYER_MONEY,
  ENEMY_DEFEATED, ENEMY_EFFECT_WHEN_DEFEATED)
- ENEMY_HISTORY(ENEMY_ID → ENEMY, OBJECT_ID → OBJECT)
- TRASH(<u>TRASH_ID</u>, TRASH, TRASH_IN_ROOM → ROOM)
- ACTIONS_ENGLISH(<u>ACTION_ID</u>, ACTION, ACTION_SQL_COMMAND,
  ACTION_DESCRIPTION, ACTION_TYPE).
- HISTORY(COMMAND_ID, COMMAND_GIVEN, COMMAND_AS_ACTION)
- OUTPUT(OUTPUT_ID, OUTPUT, OUTPUT_TIME, OUTPUT_IS_LAST)