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

16.12.2019

Hinweise zum SQL-Lernspiel von Frau Isabell Rösner

Vorbemerkung

Dieses Lernspiel ist im Rahmen einer Abschlussarbeit entstanden. Das Spiel ist ein klassisches Textadventure mit Kommandozeilen-Schnittstelle. Als Lernspiel für SQL ist es neuartig. Es gibt schon andere bekannte SQL-Lernspiele, z.B. "SQL Island":

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

Dies ist im Prinzip aber 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, wenn dafür auch die Verwendung von SQL momentan eher freiwillig ist. In zukünftigen Versionen sind Rätsel geplant, die sich nur mit SQL lösen lassen.

Wir hatten schon einmal im Rahmen dieses Forschungsprojektes versucht, ein Lernspiel nach diesem Prinzip zu entwickeln (mit Web-Schnittstelle und Mehrbenutzer-Fähigkeiten), aber die Entwicklung ist durch den Studienabschluss der beteiligten Studierenden steckengeblieben. Ich freue mich daher, dass Frau Rösner die Entwicklung eines einfachen Prototyps als Machbarkeitsstudie durchgezogen hat. Das hat schon zu einem Erkenntnisgewinn geführt, weil auch Probleme bzw. Details aufgetaucht sind, über die wir vorher nicht nachgedacht hatten.

Wir würden dieses Spiel jetzt gern mit Ihnen ausprobieren, um "Input" für die weitere Entwicklung zu bekommen.

Dieser Text ist nicht die Original-Anleitung zum Spiel. Die Anleitung der Spiel-Autorin ist kürzer und steht in der Datei "`help`".

Voraussetzungen

- Minimal braucht man die Datei "`game.jar`" und das Verzeichnis "`src/databases`" mit den Dateien
  - `game_db.properties`
  - `game_db.script`
  - `user_db.properties`
  - `user_db.script`
- 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`" Dateien enthalten SQL-Befehle, um die Datenbank beim Start anzulegen. Sie nehmen sich den Spielspass, wenn Sie diese Dateien lesen.
- Nötig ist Java8. Falls Sie noch kein Java auf Ihrem Rechner installiert haben, können Sie sich das z.B. von "AdoptOpenJDK"

  [(https://adoptopenjdk.net/]

  herunterladen. 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. Lesen Sie die Lizenzbedingungen aber besser selbst.

Start des Programms

- Das Programm wird gestartet mit

  ```
  java -jar game.jar
  ```

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

  ```
  src/databases
  ```

  gefunden werden.
- Das Programm fragt Sie dann, ob Sie ein neues Spiel beginnen wollen, oder ein existierendes Spiel fortsetzen. Sie antworten mit "`n`" für ein neues Spiel.
- Dann fragt das Spiel Sie, mit welchem Namen Sie angesprochen werden wollen. Geben Sie z.B. Ihren Vornamen oder einen Phantasienamen ein.
- Zuletzt sollen Sie aussuchen, welches Spiel Sie starten wollen. Momentan gibt es aber nur ein "`SampleGame`" mit der ID "`g1`". Geben Sie also "`g1`" ein.
- Der Befehl

  ```
  help
  ```

  listet die Spielbefehle auf. Zusätzlich sind SQL-Anfragen und Updates möglich.
- Mit dem Befehl

  ```
  quit
  ```

  beendet man das Spiel.

Räume

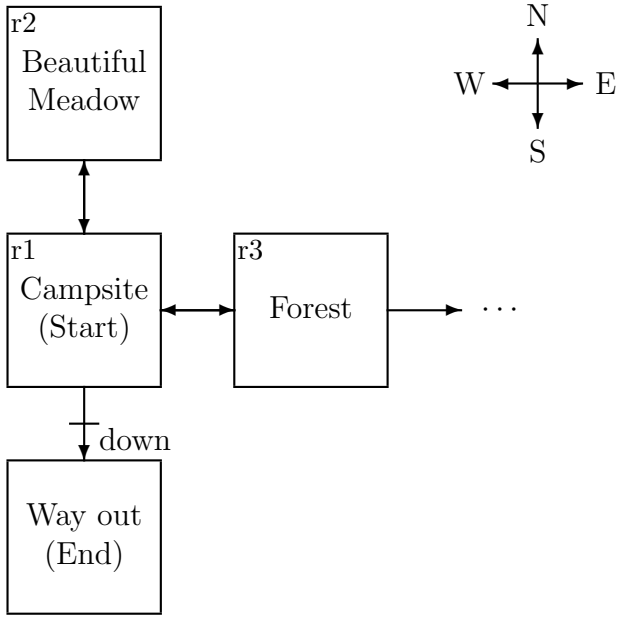
- Bei Textadventure-Spielen bewegt man "sich" bzw. die Spielfigur durch eine Spielwelt, die aus mit einander verbundenen Spielpositionen besteht. In Textadventure-Spielen heißen die Spielpositionen "Räume". Auch ein Campingplatz oder eine Lichtung im Wald wird als "Raum" bezeichnet.
- Ein Befehl zur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



- Es ist möglich, dass ein Weg versperrt ist, und man einen bestimmten Gegenstand (s.u.) benötigt, um passieren so können.
- Man kann eine zusätzliche Beschreibung des aktuellen Raums mit folgendem Befehl anzeigen:

  ```
  examine room
  ```

  Eine zusätzliche Beschreibung eines Weges in einer bestimmten Richtung (z.B. "`n`") bekommt man mit

  ```
  examine path n
  ```

Verwendung von SQL, Tabellen für Räume

- Weil es in dem Spiel ja eigentlich daraum 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 nur die bereits besuchten Räume. Es gibt natürlich auch eine Tabelle mit der vollständigen Information, 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. Leider ist die Ausgabe von Tabellen, insbesondere mit längeren Texten in den Spalten, nicht ganz einfach. Voreingestellt ist eine Ausgabe, in der die Werte der Spalten einfach durch Komma getrennt ausgegeben werden. Man kann mit

  ```
  tablestyle 2
  ```

  eine Ausgabeformatierung wählen, die einer Tabelle etwas ähnlicher ist. Die Einstellung

  ```
  tablestyle 3
  ```

braucht etwas mehr Platz, da für jeden Tabelleneintrag eine Zeile ausgegeben wird. Da nun jeweils der Spaltenname vorangestellt wird, ist die Struktur der Ergebnis-Tabelle hier aber besonders klar. Insgesamt wird es natürlich übersichtlicher, wenn man sich nicht einfach alle Spalten der Tabelle ausgeben läßt, sondern eine bewusste Auswahl trifft.

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

```
player(player_id, player_name, player_life_points,
       player_attack, player_money, player_in_room → room,
       player_show_statements, player_line_break,
       player_table_style,
       player_plays_game, player_last_login)
```

  Folgende Anfrage liefert also den gewünschten Raum:

```
select player_in_room from player
```

  Man kann die SQL-Anfrage nicht über mehrere Zeilen verteilen, da beim Drücken der "Enter"-Taste ausgeführt wird, was bis dahin eingegeben wurde.

- Auch Update-Befehle sind möglich:

```
update player set player_in_room = 'r2'
```

  Falls man vorher noch nie in dem Raum mit der `room_id` "`r2`" war, ist dieser Zug aber illegal, und das Programm nimmt die Änderung zurück.

- Aus Datenbank-Sicht ist das Spielprinzip also Folgendes:
  - Der Spieler kann auf seiner Datenbank bzw. seinen Tabellen im Prinzip beliebige Änderungen vornehmen.
  - Das Steuerprogramm vergleicht nach jedem Zug des Spielers den Zustand der Spieler-Datenbank mit den Daten in einer zentralen Datenbank (für den Spieler nicht zugänglich). Dadurch erkennt es die Updates, die der Spieler ausgeführt hat.
  - Illegale Updates, wie etwa die beliebige Teleportierung, werden zurückgenommen.
  - Legale Updates können dazu führen, dass das Steuerprogramm zusätzliche Updates am Zustand der Spieler-Datenbank vornimmt. Zum Beispiel ist es legal, sich per Update-Befehl in einen benachbarten Raum zu versetzen. Wenn der Raum das erste Mal betreten wird, wird die Beschreibung des neuen Raums in die Spieler-Datenbank kopiert.
  - Tabellen oder Spalten, die grundsätzlich nicht geändert werden können, sind ggf. schon durch die Zugriffsrechte der Datenbank geschützt. Das vereinfacht das Steuerprogramm etwas, da es diese Updates nicht zu behandeln braucht.

- 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.

- Ein Vorteil der Nutzung von SQL-Befehlen ist, dass sie den Spieler teilweise Arbeit abnehmen können. Momentan kann man sich zwischen den bereits besuchten Räumen beliebig teleportieren. Wenn es später auf den Wegen mehr Hindernisse gibt, soll das nicht mehr ganz so einfach möglich sein, aber das Programm würde dann versuchen, einen passierbaren Weg für den Spieler zu finden.
- Ein andere interessante SQL-Anfrage ist, 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.
- Folgender Befehl listet alle Tabellen auf, die für den Spieler zugänglich sind:

  ```
  tables
  ```

- Es gibt eine Tabelle

  ```
  history(command_id, command_given, command_as_action),
  ```

  in der die Eingaben abgelegt werden. Befehle wie "`examine room`" werden dabei aber ausgelassen. 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, action_description).
  ```

  Es gibt auch eine Tabelle `actions_deutsch`. Man könnte mit einer `LIKE`-Bedingung in SQL nach einem Befehl suchen.

Gegenstände

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

- Man kann Gegenstände untersuchen mit dem Befehl "`examine`". Leider muss man in der aktuell implementierten Version verschiedene Varianten des Befehls verwenden, je nachdem, worum es sich handelt. Z.B.

  ```
  examine inv tent
  ```

  weil das Zelt "`tent`" ein Objekt im Inventar des Raums ist (ein Container-Objekt). Es gibt auch "versteckte" Gegenstände, die man erst entdeckt, wenn man ein Objekt untersucht. Zum Beispiel wäre ja möglich, dass in dem Zelt noch weitere Objekte enthalten sind.

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

  ```
  take water bottle
  ```

- Einen Gegenstand, den man genommen hat, oder der frei im Raum herumliegt, kann man untersuchen mit

  ```
  examine object water bottle
  ```

  Wieder muss man "`examine object`" statt einfach "`examine`" schreiben — zum Teil liegt das an der Anforderung, dass jeder Spiel-Befehl sich in eine nicht zu komplexe SQL-Anweisung übersetzen lassen muss (s.u.).

- Der Befehl "`examine object`" funktioniert momentan nicht für Gegenstände in Container-Objekten (wie dem Zelt). Solche Gegenstaende muss man zuerst mit "`take`" nehmen.

- 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
  ```

- Man kann Gegenstände verwenden mit dem Befehl "`use`".

- 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.

Tabellen für Objekte

- Die Tabelle `object` enthält die unveränderlichen Daten der Objekte:

  ```
  object(object_id, object_name, object_description,
         object_value,
         object_eatable, object_effect_when_eaten,
         object_switchable,
         object_magical, object_effect_when_magical,
         object_attack)
  ```

  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.

- Die veränderlichen Daten der Objekte stehen in der Tabelle

  ```
  object_properties(object_id → object,
                    object_eaten, object_switched_on,
                    object_magical_effect_used,
                    object_in_room° → room,
                    object_from_npc, object_in_inventory)
  ```

  Objekte, die man mit sich herumträgt, sind durch

  ```
  object_in_inventory = 'true'
  ```

  gekennzeichnet.

- Die Trennung der beiden Tabellen erfolgte, da das Datenbank-Schema schon für die Speicherung mehrerer Spiel-Zustände ausgelegt ist. Den Eintrag in der Tabelle `object_properties` muss es dann ein Mal für jeden Spieler bzw. jedes laufende Spiel geben, während `object` nur die festen Daten enthält, die der Spiel-Autor verfasst hat.

- Im Prinzip können (fast) alle Aktionen auch mit SQL-Befehlen ausgeführt werden. Z.B. kann man das Objekt "`o2`" ("bag of chips") essen, indem man folgenden Update ausführt:

  ```
  update object_properties
  set    object_eaten = 'true'
  where  object_id = 'o2'
  ```

Es muss natürlich der ganze Befehl in eine Zeile geschrieben werden. Mit einer Unteranfrage kann man auch den Namen "`bag of chips`" des Gegenstands verwenden, nicht nur die ID. Eine entsprechende SQL-Anweisung wird auch angezeigt, wenn man den entsprechenden Spiel-Befehl eingibt:

```
eat bag of chips
```

- 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. Leider scheint es mit dem gleichzeitigen Nehmen mehrerer Gegenstände Probleme zu geben.
- Leider kann man momentan nicht alle Aktionen mit Gegenständen auch über SQL ausführen: Wenn die Untersuchung eines Gegenstands zum Entdecken versteckter Gegenstände führt, geschieht das nur über den Befehl "`examine`". Den Text, der ausgegeben wird, kann man natürlich mit SQL abfragen.
- Folgende Tabelle enthält unbewegliche Container-Objekte:

```
room_inventory(inventory_id, inventory_name, inventory_article,
               inventory_description, inventory_in_room,
               inventory_needs_object)
```

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`".
- 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.
- Man kann NPCs auch anschauen:

```
examine npc mark
```

  Eventuell bekommt man so zusätzliche Informationen.
- Wenn man Geld hat, kann man mit manchen NPCs auch handeln:

```
trade with mr cook
```

Es wird dann ein Angebot gemacht.

- Man kann NPCs auch ein Objekt geben:

  ```
  give bag of chips to john
  ```

  Es ist dann natürlich weg. Vielleicht zeigt sich die Person aber erkenntlich.

Gegner, Kämpfe

- Es gibt auch Gegner ("Monster"). Man kann Gegner anschauen:

  ```
  examine enemy ...
  ```

  Dabei muss man "..." natürlich durch den Namen des Gegners ersetzen. Falls es dabei Unklarheiten geben sollte, kann man den Namen in der Tabelle `enemy` nachschlagen (s.u.).

- Man kann versuchen, mit ihnen zu sprechen:

  ```
  talk to enemy ...
  ```

- Man kann sie angreifen:

  ```
  attack ...
  ```

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

  ```
  attack ... use ...
  ```

  Waffen verbessern die eigenen Angriffswerte.

  Leider funktioniert momentan der Angriff nur über den `attack`-Befehl, nicht über SQL-Anweisungen. Wenn man den Kampf gestartet hat, läuft er bis zum (bitteren) Ende ab, man kann nicht die Flucht ergreifen oder die Waffe wechseln. Man kann nur zuschauen, wie die Lebenspunkte von Gegener und Spieler sinken. Falls man verlieren sollte, wir man aber nur in der Startraum zurückversetzt. Man behält alle Gegenstände, die man vor dem Kampf hatte. In der derzeitigen Version greifen Gegner nicht von sich aus an.

Tabellen für andere Personen, Aufgaben, Gegner

- 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)
  ```

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

```
quest(quest_title, quest_text, quest_from_npc → npc,
      quest_solved, quest_reward_received)
```

- Gegener, denen man schon begegnet ist, sind in der Tabelle `enemy` aufgelistet:

```
enemy(enemy_name, enemy_text, enemy_description,
      enemy_life_points, enemy_attack, enemy_in_room → room,
      enemy_current_life_points, enemy_defeated,
      enemy_effect_when_defeated)
```

- Die eigenen Lebenspunkte und Angriffswerte kann man in der Tabelle `player` (s.o) nachschauen.

Der "Cheat" Modus

- Wenn man einen bestimmten Gegenstand hat (im Spiel ist das "Big Leaf"), kann man den Befehl

  ```
  cheat
  ```

  verwenden. Anschließend kann man eine SQL-Anfrage eingeben, die sich auf die eigentlichen Spiel-Tabellen mit allen Daten bezieht. Bisher wurden nur die Sichten gezeigt, die nur den Zugriff auf die schon bekannten Daten gestatten.

- Zumeist sind die eigentlichen Tabellen genau so benannt wie die Sichten, aber im Plural. Z.B. wäre jetzt möglich:

  ```
  select *
  from   rooms
  where  room_id not in (select room_id from room)
  ```

  Damit würde man Daten von Räumen abfragen, die man noch nicht kennt.

- Damit das Mogeln nicht ausufert, wird aber nur die erste Ergebnis-Zeile angezeigt. Hätte man also nur

  ```
  select * from rooms
  ```

  eingegeben, wäre möglich, dass man nur die Daten eines schon bekannten Raums bekommt, und damit den Mogel-Versuch vertan hat. Die Gesamtanzahl der Mogel-Versuche ist begrenzt.

- Alle Tabellen des Spiels stehen in der Datei `game_table_information`.