Kurzeinführung in Datenbanken am Beispiel des Spiels "Schemaverse"

Prof. Dr. Stefan Brass

Universität Halle, Institut für Informatik

Inhalt

Was sind Datenbanken? (1)

- Datenbanken dienen zur Verwaltung von Daten:
  - Man kann Daten in eine DB speichern (hineintun)
  - und sie wieder abfragen (nachschauen).

    Entlastung des Gedächtnisses durch Notizen.
- Die Daten einer Datenbank (DB) sind strukturiert (häufig in Form von Tabellen), nicht einfach Texte.
- Dadurch sehr mächtige Such-/Filterfunktionen.

  Z.B. Lernkartei Deutsch/Englisch: Suche nach "hat".
- Auch für sehr große Datenmengen.

  Yahoo hatte schon 2008 eine 2-Petabyte Datenbank:
  [http://www.computerworld.com/article/2535825/]
- Bietet auch Aggregationsfunktionen (z.B. Summe).

Was sind Datenbanken? (2)

- Tabellen werden auch in Tabellenkalkulationen wie Excel genutzt, aber relationale Datenbanken:
  - bestehen meist aus mehreren Tabellen.
  - funktionieren mit großen Datenmengen.
  - unterstützen auch komplexere Auswertungen der Daten mit relativ einfachen Anfragen in SQL (DB-Sprache).
  - erlauben gleichzeitige Zugriffe mehrerer Nutzer.
- Hier wird die Datenbank für ein Mehrbenutzer-Spiel verwendet: Alle Spieler greifen auf einen gemeinsamen Datenbestand zu (und ändern ihn).
- Ein Datenbanksystem arbeitet ähnlich wie ein Webserver: Über das Netz kommt eine Anfrage, er schickt die Antwort.

Was sind Datenbanken? (3)

- Während man in eine Excel-Tabelle einfach Daten einfüllt, muss man bei einer relationalen Datenbank erst die benötigten Tabellen in einem DB-Schema deklarieren.

  Es hat aber auch Vorteile, wenn die Struktur der Daten "bedachter" ist, und nicht wild wächst. Mehr Prüfungen, Steigerung der "Datenqualität".

- Die Datenbank enthält ein Abbild eines Ausschnitts der realen/gedachten Welt. Man muss die für die Anwendung nötigen Strukturen erkennen und in Tabellen fassen.

- Bekannte Datenbanksysteme: MySQL, SQL Server, Oracle.

  Historie: [https://hpi.de/naumann/projects/rdbms-genealogy.html]

- Hier wird PostgreSQL verwendet (Open Source, kostenlos).

  Vorreiter der objektrelationalen Datenbanken.

  [https://www.postgresql.org/docs/manuals/]

Inhalt

Grundlagen des Spiels

- Es gibt ein Universum mit Planeten.

  Das Universum ist 2D: Die Planeten haben (X,Y)-Koordinaten.

  Die Koordinaten sind jeweils ganze Zahlen, offenbar von -10 Mio. bis +10 Mio.

- Man hat einen Heimatplaneten, kann aber weitere “erobern”.
- Resourcen: Geld und Treibstoff (1:1 konvertierbar).
- Man kann Raumschiffe bauen, ihre Eigenschaften verbessern.
- Aktionen, die Raumschiffe ausführen können:
  - “Mining” (Gewinnung von Treibstoff auf einem Planeten).
  - Zu einem anderen Planeten fliegen.
  - Ein anderes Raumschiff angreifen.
  - Ein anderes Raumschiff (oder sich selbst) reparieren.

Wichtige Tabellen (1): Spieler

- Die Tabelle `my_player` enthält die Daten des Spielers:

  Virtuelle Tabelle (Sicht), deswegen für jeden Spieler anderer Inhalt. Die Daten stammen aus der Tabelle `player` (nur für Administrator zugreifbar).

my_player				
id	username	created	balance	fuel_reserve
9	brass	2017-07-21	10000	100000

  - `id`: Eindeutige Nummer des Spielers.
  - `username`: Benutzername des Spielers.
  - `created`: Zeitstempel wann der Spieler angelegt wurde.
  - `balance`: Geld des Spielers (Kontostand).
  - `fuel_reserve`: Treibstoff-Vorrat des Spielers.
  - Es gibt noch weitere Spalten (u.a. Passwort im Klartext).

Wichtige Tabellen (2): Planeten

- Die Tabelle `planets` enthält die Daten der Planeten (aktuell 500 Zeilen, hier nur "mein Planet"):

  Eingeschränkte Sicht auf nur für Administratoren zugreifbare Tabelle `planet`.

planets				
id	name	location_x	location_y	conqueror_id
139	Rio_138	-2696624	5991625	9

  - `id`: Eindeutige Nummer des Planeten.
  - `name`: Name des Planeten.
  - `mine_limit`: Max. Anzahl Schiffe beim Treibstoff-Abbau.
  - `location_x`, `location_y`: Koordinaten des Planeten.
  - `conqueror_id`: ID des Spielers, dem der Planet gehört.
  - `location`: (X,Y)-Koordinaten zusammen (Typ `point`).

Wichtige Tabellen (3): Raumschiffe

- `my_ships` enthält die Daten eigener Raumschiffe.

  Die Tabelle hat insgesamt 30 Spalten.

my_ships					
id	player_id	name	current_health	max_health	···
1	9	Adler1	100	100	···

  - `id`: Eindeutige Nummer des Raumschiffes.
  - `player_id`: Nr. des Spielers, dem das Schiff gehört.

    Überflüssig: Diese Sicht auf die für normale Spieler nicht zugreifbare Tabellen `ship` und `ship_control` enthält nur eigene Schiffe.
  - `name`: Name des Raumschiffs.
  - `current_health`: Lebenspunkte des Raumschiffs.

    Bei 0 Lebenspunkten kann das Schiff keine Aktion mehr ausführen, kann aber für eine gewisse Zeit noch repariert werden.
  - `max_health`: Volle Lebenspunktezahl (kein Schaden).

Wichtige Tabellen (4): Raumschiffe

- Weitere Spalten von `my_ships`:

my_ships					
···	current_fuel	max_fuel	max_speed	range	···
···	100	100	1000	30	···

  - `current_fuel`: Treibstoff-Menge im Tank.

    Treibstoff kann "gebeamt" werden: Solange `my_player.fuel_reserve` > 0 kann man alle seine Schiffe auftanken, egal wo sie sind.

  - `max_fuel`: Größe des Tanks.
  - `max_speed`: Maximal mögliche Geschwindigkeit.

    Die Änderung der Geschwindigkeit um 1 (oder Richtungsänderung um 1°) kostet eine Treibstoff-Einheit. Mit 100 Treibstoff-Einheiten kann man also bis zur Geschwindigkeit 50 beschleunigen und wieder abbremsen.

  - `range`: Maximale Entfernung, in der Aktionen möglich sind.

    Zum Beispiel Angriffe auf fremde Raumschiffe.

Wichtige Tabellen (5): Raumschiffe

- Noch mehr Spalten von `my_ships`:

my_ships					
···	attack	defense	engineering	prospecting	···
···	3	2	3	12	···

  - `attack`: Angriffsstärke (gegenüber anderen Schiffen).

    Formel für Schaden aus der Dokumentation (`defense_efficiency=50`):

    ```
    damage = my_ships.attack * (50/(enemy.defense+50))
    ```

    Ein Schiff mit `attack=5` zieht von einem mit `defense=2` in jeder Zeiteinheit 4 Lebenspunkte ab (also auf 0 nach ca. 4 Minuten).

  - `defense`: Verteidigungsstärke (Schutzschild).
  - `engineering`: Reparaturmöglichkeit.

    Die Reparatur-Aktion erhöht die Lebenspunkte um diesen Wert pro "`tic`".

  - `prospecting`: Fähigkeit zum Abbau von Treibstoff.

    Man kann 20 Punkte über diese vier Eigenschaften verteilen.

Wichtige Tabellen (6): Raumschiffe

- Und noch mehr Spalten von `my_ships`:

my_ships				
location_x	location_y	direction	speed	destination_x
-2696624	5991625	0	0	NULL

  - `location_x`/`location_y`/`location`: Position des Schiffes.
  - `direction`/`target_direction`: Flugrichtung.

    Angabe in Grad: 0: in x-Richtung (Osten), 90: in y-Richtung (Norden), 180: entgegen x-Richtung (Westen), 270: entgegen y-Richtung (Süden). Wenn der `target`-Wert abweicht, gab es nicht genügend Treibstoff.
  - `speed`/`target_speed`: Geschwindigkeit.

    Mit Geschwindigkeit 1000 würde man in einer Stunde (360 `tics`) erst 1.8% der All-Ausdehnung in x-Richtung durchfolgen haben.
  - `destination_x`/`destination_y`/`destination`: Ziel.

    Statt die Richtung zu berechnen, kann man die Ziel-Koordinaten setzen.

Wichtige Tabellen (7): Raumschiffe

- Letzte Folie mit Spalten von `my_ships`:

my_ships		
action	action_target_id	last_action_tic
ATTACK	2	4256

  - `action`: Aktion, die das Schiff ausführen soll: `MINE`, `ATTACK`, oder `REPAIR`.

    Die Aktion wird beim nächsten "`tic`" ausgeführt und dann bei jedem folgenden "`tic`" bis man den Wert in dieser Spalte ändert.

  - `action_target_id`: Nr. des Schiffes/Planeten, auf die sich die Aktion bezieht.

    Für `ATTACK` oder `REPAIR` Schiff-ID, für `MINE` Planet-ID.

  - `last_action_tic`: Letzte Zeiteinheit, bei der eine Aktion ausgeführt wurde.

    Es gibt außerdem: `last_living_tic`, `last_move_tic`

Wichtige Tabellen (8): Preisliste für Upgrades

price_list		
code	cost	description
SHIP	1000	HOLY CRAP. A NEW SHIP!
FLEET_RUNTIME	10000000	Add one minute of runtime to a fleet script
MAX_HEALTH	50	Increases a ships MAX_HEALTH by one
MAX_FUEL	1	Increases a ships MAX_FUEL by one
MAX_SPEED	1	Increases a ships MAX_SPEED by one
RANGE	25	Increases a ships RANGE by one
ATTACK	25	Increases a ships ATTACK by one
DEFENSE	25	Increases a ships DEFENSE by one
ENGINEERING	25	Increases a ships ENGINEERING by one
PROSPECTING	25	Increases a ships PROSPECTING by one

"fleet scripts" sind Programme (in PL/pgSQL), die die Schiffe automatisch steuern (z.B. auch in Abwesenheit des Spielers Eindringlinge automatisch angreifen).

Wichtige Tabellen (9): Fremde Raumschiffe

- `ships_in_range` listet Raumschiffe von anderen Spielern, die sich in Reichweite befinden (feindlich?).

ships_in_range				
id	ship_in_range_of	player_id	name	health
10	1	3	Destroyer	1.0

  - `id`: ID des fremden Schiffes.
  - `ship_in_range_of`: ID des eigenen Schiffes.
  - `player_id`: Besitzer des fremden Schiffes.
  - `name`: Name des fremden Schiffes.
  - `health`: Schadensstatus des fremden Schiffes.

    Berechnet als `current_health/max_health` des fremden Schiffes.
  - `location`: Position des fremden Schiffes (`point`).

Wichtige Tabellen (10): Andere Spieler

- Die Tabelle `online_players` listet alle Spieler, die momentan angemeldet sind:

online_players	
id	username
9	brass

  - `id`: Eindeutige Nummer des Spielers.
  - `username`: Benutzername des Spielers.

Wichtige Tabellen (11): Ereignisse/Spielverlauf

- Die Tabelle `my_events` enthält den Spielverlauf:

my_events					
id	action	player_id_1	ship_id_1	player_id_2	ship_id_2
2372	BUY_SHIP	9	1	NULL	NULL
4207	ATTACK	3	10	9	1

  - `id`: Global eindeutige Nummer des Events.

    Es sind viele, weil alle 10 Sekunden die Aktion "`TIC`" protokolliert wird.
  - `action`: Ereignis-Typ.
  - `player_id_1/2`: Nr. des ausführenden/betroffenen Spielers.
  - `ship_id_1/2`: Nr. des ausführenden/betroffenen Schiffes.
  - `location`: Ort des Ereignisses.
  - `public`: Ergnis für alle Spieler sichtbar.
  - `tic/toc`: Nr. des Zeitpunktes und tatsächliche Uhrzeit.

Inhalt

Zugriff mit phpPgAdmin (1)

- Zugriff auf die Datenbank ist möglich mit der Webschnittstelle "phpPgAdmin":
  https://appinventor.informatik.uni-halle.de/phppgadmin/index.php

  Auf dem Startbildschirm (Balken "Introduction" oben unter "phpPgAdmin") könnte man auch die Sprache Deutsch einstellen.
- Im linken Bereich auf Server "PostgreSQL" klicken.
- Benutzername und Passwort für die Datenbank eingeben.
- Nun wählt man die Datenbank "schemaverse".
- Darin wiederum das Schema "public".
- Darauf klicken: Listet u.a. "Tables" und "Views".

Zugriff mit phpPgAdmin (2)

- Die meisten Tabellen sind nur für den Besitzer der Datenbank (Nutzer `schemaverse`) zugänglich.

  Wenn man will, kann man sich aber das Schema anzeigen lassen (Spalten).

- Als normaler Spieler kann man auf die "Views" ("Sichten") zugreifen, das sind virtuelle Tabellen, die aus den Daten der eigenlichen Tabellen berechnet werden.

  Auf diese Art kann man z.B. nur die Daten der eigenen Schiffe sehen.

- Wenn man auf eine Sicht wie "`my_player`" klickt, wird zunächst das Schema (Tabellenspalten) angezeigt.

  Hier gibt es den Knopf "`ALTER`" zur Änderung z.B. des Spaltennamens.

  Wenn man das aber versucht, erscheint die Meldung "Keine Berechtigung".

- In der Fußzeile gibt es den Knopf "`Browse`", damit wird der ganze Tabelleninhalt angezeigt.

Zugriff mit phpPgAdmin (3)

- Einige Tabellen sind sehr groß.

  Z.B. `planets`, `my_events`. Dann ist `Browse` wenig hilfreich.

- Wenn man auf `Select` klickt, bekommt man die Möglichkeit, einfache Filter anzugeben.

  Z.B. kann man bei `planets` nur den/die Planeten selektieren, bei denen die Spalte `conqueror_id` die eigene Benutzernummer enthält.

- Viel mehr Möglichkeiten bietet die Datenbanksprache SQL, die in der Vorlesung "Datenbanken I" gelehrt wird.

  Um eine SQL-Anfrage einzugeben, klicke man auf den Link `SQL` in der Kopfzeile (rechts oben). Es öffnet sich dann ein neues Fenster für die Abfrage, die Antwort erscheint aber im ursprünglichen Fenster. (Wenn man vorher mit dem `Select`-Formular gearbeitet hat, kann man hier die generierte Anfrage sehen. Sie ist aber unnötig kompliziert.)

Zugriff mit phpPgAdmin (4)

- Meine Planeten:

```
SELECT *
FROM   planets
WHERE  conqueror_id = (SELECT id FROM my_player)
```

Groß/Kleinschreibung, Leerplatz und Zeilenumbrüche sind in SQL egal.

- Planeten mit Entfernung < 500000 von Planet Nr. 139:

```
SELECT x.id, x.name, x.location <-> y.location
FROM   planets x, planets y
WHERE  x.location <-> y.location < 500000
AND    y.id = 139
```

Der Operator <-> berechnet in PostgreSQL die euklidische Distanz zwischen zwei Werten vom Typ `point`.

Zugriff mit phpPgAdmin (5)

- Interessante Ereignisse, chronologisch sortiert:

```
SELECT *
FROM   my_events
WHERE  action <> 'TIC'
ORDER  BY id
```

In SQL schreibt man <> für $\neq$. Zeichenkettenkonstanten werden in `'...'` eingeschlossen. Umgekehrt chronologische Sortierung: `ORDER BY id DESC`.

- Wie viele Planeten gibt es überhaupt?

```
SELECT COUNT(*)
FROM   planets
```

Wenn Sie nur Planeten zählen wollen, die niemand gehören, fügen Sie `WHERE conqueror_id IS NULL` hinzu.

Sie können auch `MIN(location_x)`, `MAX(location_x)` verwenden, um die Ausdehnung des Universums in X-Richtung zu erfahren.

Ausblick: Spiel-Aktionen mit SQL

- Um ein neues Raumschiff zu bauen, fügt man eine Zeile in `my_ships` ein:

```
INSERT INTO my_ships(name, attack, defense,
                     engineering, prospecting)
VALUES ('MyShip', 5, 1, 4, 10)
```

- Es gibt für alle SQL-Aktionen Funktionen, z.B.

```
SHIP_COURSE_CONTROL(Ship ID, Target Speed,
                    Target Direction, Destination)
```

- So startet man Schiff 1 zum Mittelpunkt des Universums:

```
SELECT SHIP_COURSE_CONTROL(1, 1000, NULL,
                           POINT(0,0))
```

[https://schemaverse.com/tutorial/tutorial.php?page=Movement]

Inhalt

Web-Schnittstelle (1)

- Der Datenbank-Zugriff aus AppInventor geschieht über eine Web-Schnittstelle (am Institut entwickelt).
  - Diese besteht aus 11 kurzen PHP Programmen.

    PHP wird häufig für server-seitige Web-Programmierung verwendet.
  - Man kann diese Programme aufrufen, indem man auf die entsprechende Webadresse zugreift.

    Auch von AppInventor aus. Der Webserver führt die Programme dann aus, um eine vom Programm generierte Webseite zu liefern.
  - Die Programme enthalten ihrerseits SQL-Anweisungen, um mit der Datenbank zu kommunizieren.

    Also Daten abzufragen bzw. Änderungen durchzuführen.
  - Die Daten werden nicht wie normale Webseiten als HTML geliefert, sondern im Datenformat JSON.

    AppInventor hat Bausteine, um Daten in diesem Format zu verarbeiten.

Web-Schnittstelle (2)

- Die Eingabe für die Programme (Parameter der Abfrage oder Spielaktion) werden an die Webadresse angehängt:

  http://appinventor.informatik.uni-halle.de/schemaverse-json-backend/my_player.php?username=alex_21&password=simple

- Die Komponenten der Webadresse (URI) sind:
  - Protokoll: `http`
  - Rechner: `appinventor.informatik.uni-halle.de`
  - Verzeichnis: `schemaverse-json-backend`
  - Programm: `my_player.php`
  - Attribut-Wert-Paare, z.B. `username=alex_21`.

  Ein Web-Formular (Methode `GET`) erzeugt ebenfalls dieses Format.

Web-Schnittstelle (3)

- Sicherheitshinweis:
  - Einfache Lösung, die von AppInventor aus benutzbar ist.
  - Normal haben Passworte in Webadressen nichts zu suchen.

    Tauchen so z.B. in Bookmarks, History, Log-Dateien auf.
- Das Ergebnis wird JSON-codiert geliefert:

```
[{"id":"3","username":"alex_21",
  "balance":"10000","fuel_reserve":"100000"}]
```

  "JavaScript Object Notation". Die äußeren Klammern `[...]` zeigen an, dass es eine Liste ist (in diesem Fall aber mit nur einem Element).
  Die inneren Klammern `{...}` zeigen ein Objekt mit Attribut-Wert-Paaren an.
  Diese Schachtelung entspricht einer Tabelle (jedes Objekt einer Zeile).
- Man kann das mit jedem Web-Browser testen.

Web-Schnittstelle (4)

Wenn der Server die richtigen Antwort-Kopfzeilen sendet, wird die Antwort vom Browser auch strukturiert formatiert dargestellt:

Spiel-Aktionen (1)

- Webseite mit Liste der PHP-Programme (u.a. Links):
  http://appinventor.informatik.uni-halle.de/schemaverse-json-backend/schemaverse-summerbyte.html
- Diese enthält u.a. Webadressen für folgende Aktionen:
  - Bauen eines Raumschiffs (mit Parametern)
  - Raumschiff bewegen
  - Raumschiff auftanken
  - Eigenschaften eines Raumschiffes verbessern (Upgrade)
  - Auftrag für ein Raumschiff erteilen

    `MINE`, `ATTACK`, `REPAIR`, `NOACTION` (d.h. Stop).
  - Zwischen Geld und Treibstoff konvertieren

Spiel-Aktionen (2)

- Außerdem werden folgende Abfragen unterstützt:
  - Basisdaten des Spielers
  - Planeten des Spielers
  - Raumschiffe des Spielers
  - Planeten im Bereich eines Raumschiffes
  - Feindliche Raumschiffe im Bereich der eigenen Schiffe
- Man sucht sich jeweils heraus, wie das Programm heisst, und welche Parameter es benötigt.
  - Dann konstruiert man die entsprechende Webadresse.
  - Für diese Adresse erzeugt man einen Webzugriff.
  - Wenn die Antwort da ist, extrahiert man die Daten.

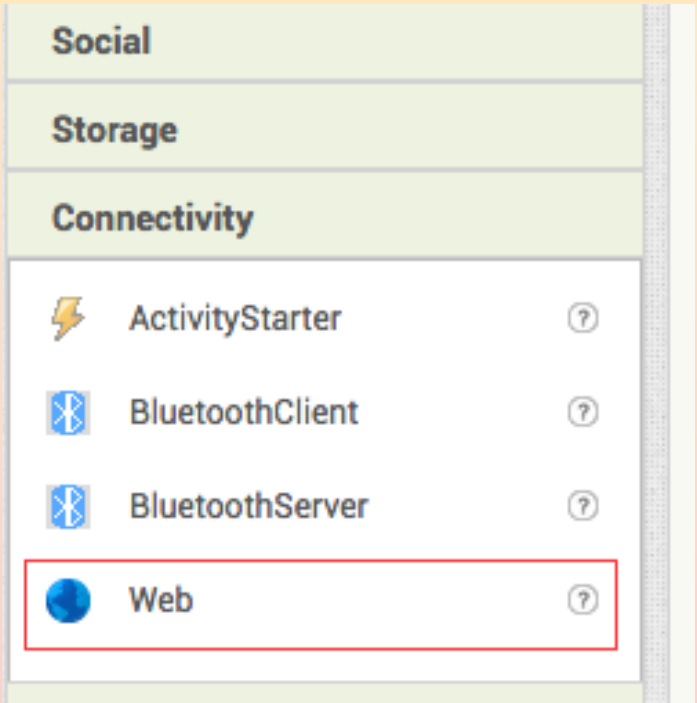
AppInventor: Web-Komponente (1)

Um in AppInventor eine Anfrage zum Server zu senden,

- wird eine unsichtbare Web-Komponente (aus der Kategorie Connectivity) benötigt,
- deren URL-Eigenschaft auf den gewünschten Einstiegspunkt und die zu übertragenden Daten festgelegt wird und
- abschließend mit Aufruf der GET-Methode abgesendet wird.

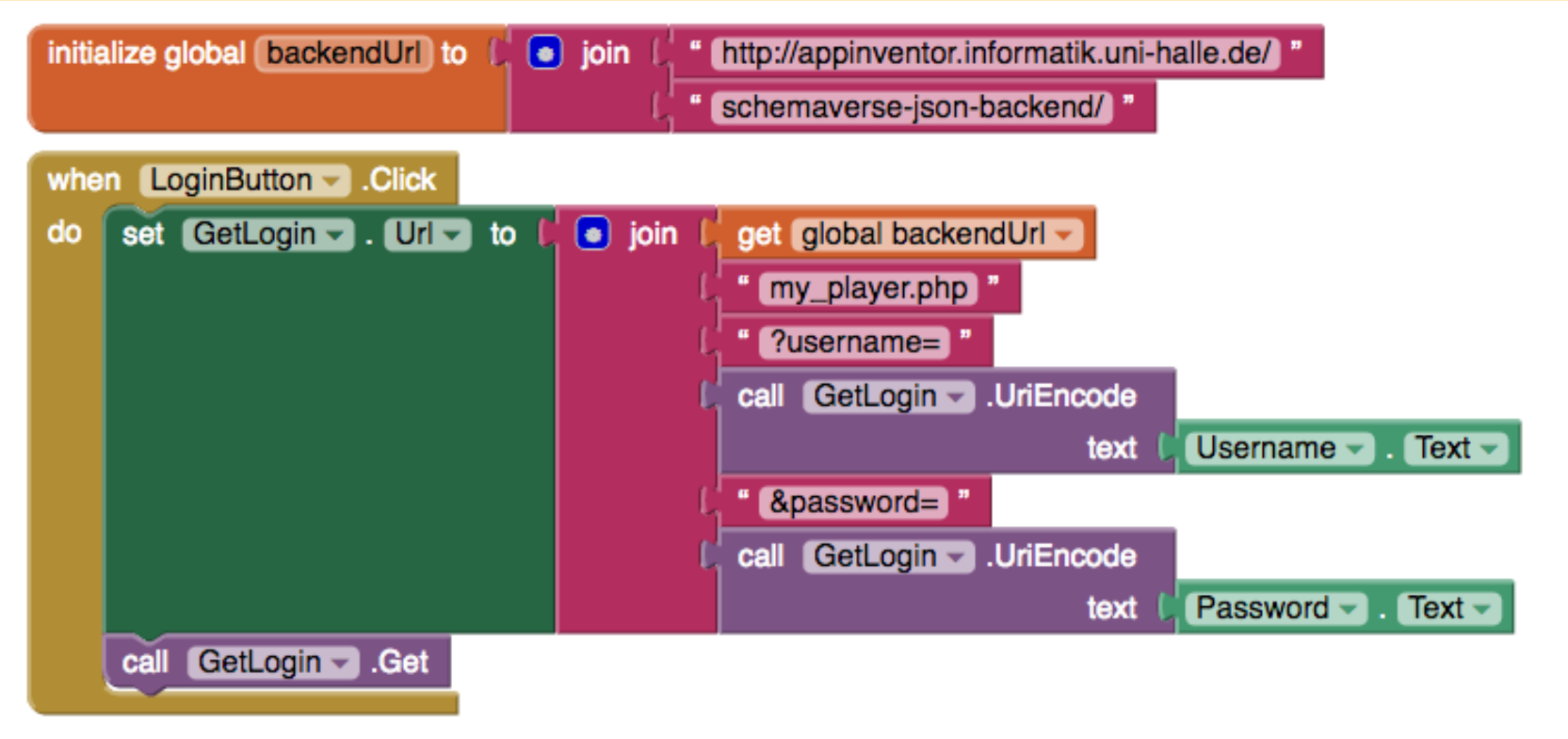
AppInventor: Web-Komponente (2)

Die Web-Komponente befindet sich in der Palette des Designers in der Kategorie `Connectivity` und kann von dort per Drag&Drop in das Vorschaubild des Viewers kopiert werden.



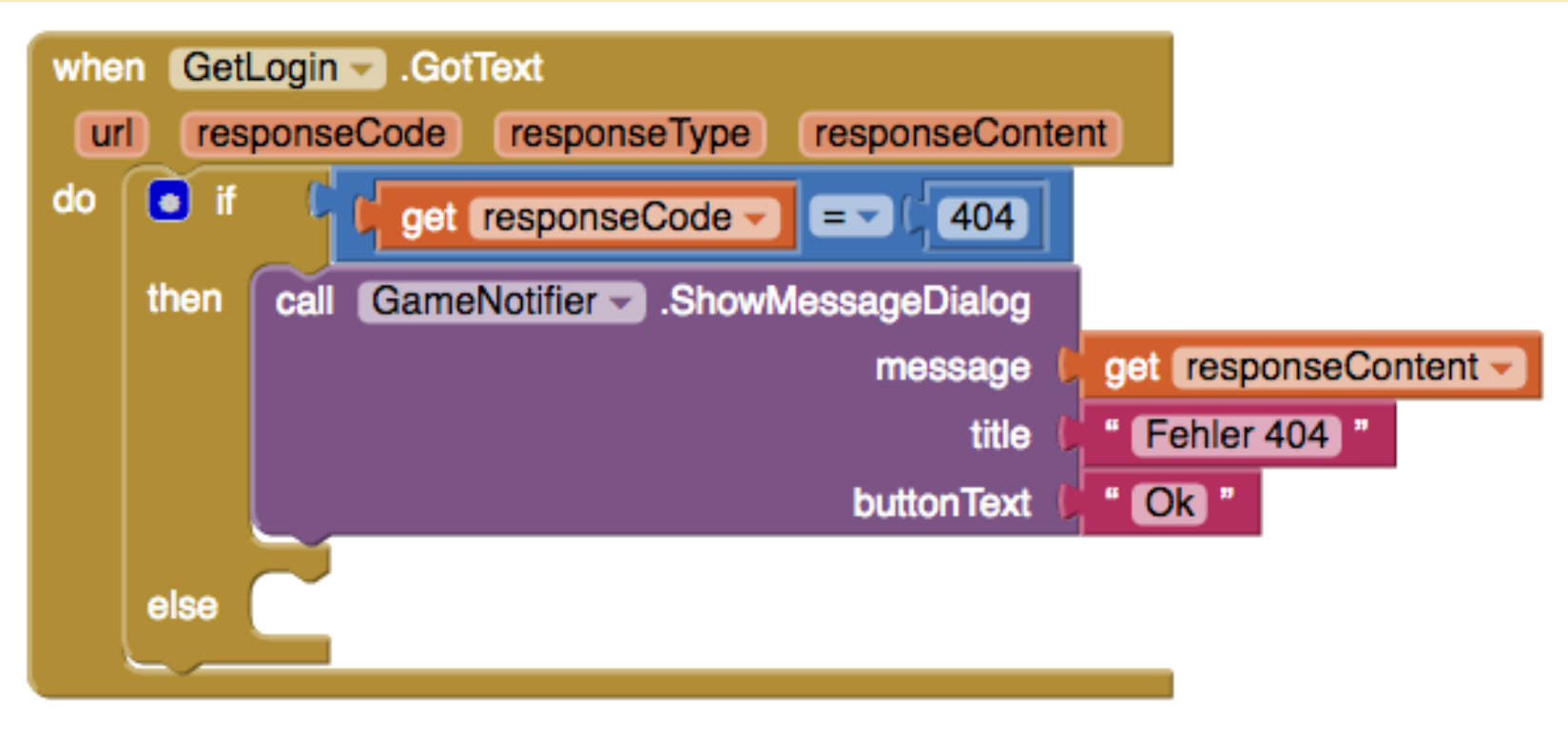
AppInventor: Web-Komponente (3)

Zuerst wird die URL durch Zusammenfügen mehrerer Zeichenketten erstellt (Konkatenation), danach die Anfrage mit dem Aufruf `WebKomponente.Get` abgesendet.



AppInventor: Web-Komponente (4)

Nachdem die Web-Komponente die Antwort des Servers geladen hat, löst diese das `GotText`-Ereignis aus. `responseContent` enthält den Antworttext.



Inhalt

Noch ein paar Tipps zum Spiel

- Man erobert einen Planeten, indem man dort mehr Treibstoff abbaut als alle anderen.

  Das gilt natürlich auch, wenn man der Einzige ist, der dort etwas abbaut.

- Wenn man keinen Planeten mehr hat, kann man keine Raumschiffe mehr bauen.

  Raumschiffe können nur auf einem Planeten erbaut werden, den man besitzt. Man sollte zuerst also in den Treibstoff-Abbau auf dem Heimatplaneten investieren, und dort eine ganze Reihe von Raumschiffen (am besten `mine_limit`) haben, die Treibstoff abbauen. Der Vorrat des Planeten ist allerdings nicht unbegrenzt (gelegentlich füllt das Spiel auch auf).

- Der Weltraum ist weit: Man muss ein Schiff aufrüsten, bevor man damit einen anderen Planeten erreichen kann.

- Um 0 Uhr beginnt eine neue Runde (alles wird gelöscht).

Datenbanken sind spannend weil ...

- sie für ganz viele verschiedene Anwendungen benötigt werden.

  Informatik ist auch wegen der Anwendungen interessant. Fast nichts geht mehr ohne Computer. Speziell Datenbanken sind sehr nahe an den Anwendungen. Meine Lieblingsanwendung ist die Feuerwerkerei.

- sie helfen, die reale Welt besser zu verstehen.

  Man lernt, die für eine Aufgabe nötigen Daten zu erkennen und zu strukturieren.

- dies auch eine Anwendung von mathematischer Logik ist.

  Im Datenbankbereich kann man durchaus vieles mathematisch präzise definieren und Sätze beweisen.

- man hier mit großen Parallelrechnern mit Hunderten von Platten arbeiten kann.

- dieser Bereich nicht von Microsoft dominiert wird.

Weitere Informationsquellen zum Spiel

- Entwickelt ca. 2009 von Joshua McDougall
- Vortrag des Autors bei der "PGCon - PostgreSQL Conference for Users and Developers" (2012):
  [https://av.tib.eu/media/19035]
  [https://www.youtube.com/watch?v=D-BqZ6ZnhUw]

  Englisch, ziemlich chaotisch, enthält aber interessante Informationen.
- Offizielle Webseite: [https://schemaverse.com/]

  Tutorial als PDF (nicht ganz aktuell):
  [https://schemaverse.com/Abstrct-TheSchemaverseGuide.pdf]
- Der Quellcode ist frei zugänglich:

  [https://github.com/Abstrct/Schemaverse]
  [https://github.com/Abstrct/Schemaverse/wiki/How-to-Install]

Dank/Team

Diese SummerByte-Aufgabe wurde vorbereitet von:

- PD Dr. Alexander Hinneburg

  Mitarbeiter der Datenbank-Gruppe, hält als "Privatdozent" (PD) eigene Vorlesungen, z.B. über "Data Mining" und "Informationsvisualisierung".
  Hat das Spiel getestet und die PHP-Skripte entwickelt. Hat gerade Urlaub.

- Marcel Jähn

  Fortgeschrittener Master-Student, Betreuer der SummerByte.
  Hat die Beispiel-App entwickelt und wird den Tag über helfen.

- Janis Dähne

  Fortgeschrittener Master-Student, Betreuer der SummerByte.
  Administrator der PostgreSQL-Datenbank.

- Felix Pahlow

  Betreuer der SummerByte, FSR-Mitglied, Autor der Folien 30, 33–36.