

Abgabe Dieses Blatt muss bis Sonntag, 12.2.2006 abgegeben werden und in der Woche vom 13.2. bis 17.2. dem Tutor vorgeführt werden. Die Bearbeitung muss zur Zufriedenheit des Tutors erfolgen, es werden keine Punkte vergeben. Schreibe bis zum **Freitag, 3.2.** die Geschichte Deines Adventures (Abschnitt 2) auf und schicke Sie dem Tutor, der dann abschätzt, ob er mit diesem Umfang zufrieden wäre. Gib dabei detailliert die Interaktionen zwischen Items (s.u.) an, die Du implementieren willst.

Dokumentation zu DOM Die Schnittstelle zwischen JavaScript und dem Browser ist – in weiten Teilen – durch das Document Object Model (DOM) gegeben. Das im Browser angezeigte HTML-Dokument ist dabei als Objekt in der globalen Variable `document` verfügbar. Entsprechend der Baumstruktur des HTML-Dokuments stellt `document` eine Baumstruktur von Objekten dar, um auf das HTML-Dokument lesend und schreibend zuzugreifen. Änderungen werden dabei automatisch am Bildschirm sichtbar.

Die in `document` (und den übrigen DOM-Objekten) vorhandenen Properties findest Du in den folgenden Standards beschrieben:

- <http://www.w3.org/TR/DOM-Level-2-Core>
- <http://www.w3.org/TR/DOM-Level-2-HTML>
- <http://www.w3.org/TR/html4>

Wichtigstes Hilfsmittel für uns sind die Methoden

- `document.getElementById(i)` liefert ein `Element`-Objekt für den HTML-Unterbaum, der das Attribut `id=i` trägt. Mit diesem Hilfsmittel kann man direkt auf einen bestimmten Teil des Dokumentes zugreifen.
- `document.createTextNode(string)` erstellt einen Knoten, der den Text `string` darstellt. Dieser Knoten ist jedoch erst sichtbar, wenn er mit `appendChild` (s.u.) in das Dokument eingefügt wird.
- `document.createElement(tag)` erzeugt ein HTML-Knoten `<tag>...</tag>` (analog zu `createTextNode`)
- Für ein Node Objekt `n` fügt `n.appendChild(c)` `c` als Kind von `n` hinzu
- Für ein Node Objekt `n` entfernt `n.removeChild(c)` den Kindknoten `c` von `n`.

Beispielanwendungen findest Du auch im vorgegebenen Code.

1 Vorgegebener Code

Auf der Homepage der Vorlesung findest Du die Seite `adventure.html`, die bereits die Grundfunktionalität für ein Adventure in JavaScript bereitstellt. Es lohnt sich auch, den vorgegebenen Code im Detail auf weiter-verwendbare Code-Fragmente durchzusehen, insbesondere für den Umgang mit dem DOM.

1.1 Item

Jedes “Ding”, das dem Spieler im Adventure begegnet, ist ein `Item`. Alle Items haben als Properties eine textuelle Beschreibung `descr` und eine Repräsentation als DOM-Element `pic`. Das DOM-Element wird an passender Stelle im Dokument des Browsers eingehängt, um den aktuellen Standort des Items anzuzeigen. Items bieten dabei die grundsätzliche Infrastruktur, das einzelne Objekt kann weitere Properties besitzen, die es näher beschreiben.

Das globale Array `all_items` enthält alle Items, die erzeugt wurden. Ein Item speichert in seiner Property `num` den Index, unter dem es in `all_items` zu finden ist. Damit können Items leicht über ihre Nummer erreicht werden, was beispielsweise bei der Erzeugung von `onClick` Attributen sehr nützlich ist. (Siehe vorgegebenen Code.) Beachte, dass sowohl Personen als auch Sachen Items sind!

1.2 ItemContainer

`ItemContainer` verwalten Mengen von Items, sie entsprechen dabei Orten in der Spielwelt. Daher gilt, dass zu jedem Zeitpunkt ein Item in höchstens einem Container enthalten sein kann, und dieser Container ist in der Property `container` des Items festgehalten. Die Methode `add` eines Containers löscht daher das Item zunächst aus dem bisherigen Container und nimmt es dann auf. Items haben damit die Aufgabe, “Aufenthaltsorte” für Items bereitzustellen und dabei die in der Wirklichkeit zutreffende Invariante einzuhalten, dass sich ein Items zu jedem Zeitpunkt nur an einem Ort befinden kann.

Einige Orte der Spielwelt sind als Unterbäume des HTML-Dokumentes im Browser sichtbar. Der `ItemContainer`, der einen solchen Ort repräsentiert, verwaltet auch den entsprechenden Unterbaum des HTML-Dokumentes, indem er in diesen Unterbaum für jedes seiner Items gerade dessen `pic` Element (siehe Abschnitt 1.1) einfügt. Wenn ein `ItemContainer` ein neues Item bekommen hat, ruft er seine eigene `display` Methode auf und übergibt ihr das neue Item. Die `Display`-Methode soll das neue Item an der Stelle sichtbar machen, die für den `ItemContainer` vorgesehen ist. Wird ein Item mit der `remove` Methode gelöscht, so ruft der Container seine Methode `undisplay` auf. Beide Methoden können auch fehlen, womit man “unsichtbare” Container erzeugen kann. Andererseits kann man beliebige Funktionen einsetzen, wodurch bestimmte Container an bestimmten Stellen im Dokument erscheinen.

1.3 Räume

Die globale Variable `rooms` enthält eine Darstellung des Spielfeldes. Das Spielfeld ist eine Matrix mit Breite und Höhe, die in `rooms.width` und `rooms.height` angegeben sind. An jeder Stelle der Matrix kann sich entweder Mauerwerk oder ein Raum befinden. Räume sind `ItemContainer`, deren `display` und `undisplay` Methoden das `pic` ihrer Items gerade in die Übersicht des Spielfeldes eintragen.

Räume sind mit ihren Nachbarn durch die Felder `north`, `south`, `east`, `west` verbunden; diese Felder sind `null`, wenn in der entsprechenden Richtung eine Wand ist. Außerdem speichern Räume ihre eigenen Koordinaten in den Properties `x` und `y`.

1.4 Der Held

Das vorgegebene `adventure.html` enthält bereits einige Beispiel-Gegenstände und einen "Helden". Der Held kann über GUI-Elemente im Spielfeld gesteuert werden und er sieht bereits, welche anderen Items sich im gleichen Raum wie er befinden.

Der Held besitzt auch schon einen Abenteuerrucksack, in den er Dinge (leider auch Personen, aber das kannst Du ja ändern!) mitnehmen kann. Er kann diese Dinge in anderen Räumen wieder ablegen. Dieser Rucksack ist ein `ItemContainer` mit speziellen `Display`-Methoden, die seinen aktuellen Inhalt neben der Steuerung anzeigen.

1.5 Der aktuelle Raum

Der Held gibt dem Spieler die Möglichkeit, den Raum zu betrachten, in dem er steht. Dazu ruft er nach jeder Bewegung seine `updateView` Methode auf, die die Beschreibungen (`Property descr`) der Items neben der Inhaltsübersicht des Rucksacks anzeigt. Als interessante Programmieretechnik setzt die Methode das `onClick` Attribut in den einzelnen Zeilen der Anzeige, womit die Anzeige interaktiv wird: Wenn der Spieler auf eine Zeile klickt, wird die Methode `workWith` des Helden aufgerufen. Sie bekommt die Nummer des Items, mit dem der Held arbeiten soll, übergeben.

1.6 Die automatische Schleife

Alle halbe Sekunde (vielleicht kannst Du hier noch eine Einstell-Möglichkeit im GUI vorsehen?) bekommen alle Items in allen Räumen (aber nicht in anderen Containern!) die Möglichkeit, etwas autonom zu unternehmen: Wenn sie eine `play` Methode haben, wird diese aufgerufen und sie können beliebige Dinge anstellen. (Als Beispielanwendungen informiert uns "John" jeweils über seinen Gemütszustand und "Jack" wandert durch die Räume.)

1.7 Erzeugung des Spielfelds

Das Spielfeld in `adventure.html` wurde vom OCaml-Programm `genplace` erzeugt; dessen Ausgabe ist nach `<!-- PLACE BEGIN -->` zu finden und kann jederzeit gegen neue erzeugte Spielfelder ausgetauscht werden.

Die Arbeitsweise von `genplace` ist wie folgt: Optionen `-w` und `-h` geben die Breite und Höhe an, zu Beginn sind an allen Koordinaten Wände. Dann werden ausgehend vom Mittelpunkt mehrerer "runs" gestartet (Anzahl mit Option `-r` einstellbar), in denen ein "Bauarbeiter" jeweils eine feste Anzahl von Schritten (Option `-s`) über das Spielfeld läuft und dabei Wände einreißt. Mit der Option `-t` kann die Wahrscheinlichkeit (in Prozent) angegeben werden, mit der er bei jedem Schritt seine Richtung wechselt. Die Option `-o` gibt die Ausgabedatei an, in die `genplace` eine vollständige HTML-Datei schreibt. Du kannst gern das Aussehen des Spielfeldes anpassen, indem Du den ausgegebenen HTML-Code in der Funktion `print_place` modifizierst (wie wäre echtes Mauerwerk statt der Hintergrundfarbe schwarz?).

2 Die Aufgabe

Erfinde zunächst die Geschichte, die man in Deinem Adventure nachspielen kann. Der Held soll mit den Items seiner Umgebung interagieren können und dabei eine ihm gestellte Aufgabe lösen. Elemente im vorgegebenen Code könnten die Grundlage für einen Krimi sein, aber Du bist völlig frei, Dir eine eigene Handlung auszudenken.

Schreibe nun die Interaktionen zwischen Items auf, die in Deiner Geschichte notwendig werden, damit der Held am Ende die Abenteuer bestehen kann. Es sollten auch einige Interaktionen vorgesehen sein, die den Helden nicht weiterbringen, aber dem Spieler das Gefühl vermitteln, sich in einer "echten" Welt zu bewegen.

Anregungen für mögliche Interaktionen sind:

- Personen reden miteinander, der Held sieht sie reden, wenn er im selben Raum ist.
- Der Held kann Personen ansprechen und ihnen Fragen stellen. Jede Person (objekt-basierte Programmierung!) antwortet auf andere Fragen.
- Der Held kann ein Item *A* aus seinem Abenteurrucksack auf ein Item *B* im Raum "anwenden". Programmiere diese Interaktion so, dass die beiden Items untereinander ausmachen, ob und wie *B* auf die Anwendung von *A* reagiert. Gib dem Spieler eine Rückmeldung.
- Der Held kann Items anketten. Ein Item "Kette" beraubt dazu das angekettete Item um die Möglichkeit, in einen anderen Raum zu gehen.

Programmierstil: Der vorgegebene Code nutzt die Freiheiten von JavaScript als objekt-basierte Sprache rigoros aus. Schreibe Dein Adventure in diesem Stil weiter, um einmal den Programmierstil von klassen-basierten Sprachen hinter Dir zu lassen. Scheue Dich auch nicht, zu den Objekten im vorgegebenen Code neue Properties hinzuzufügen, wenn Du sie brauchst. Achte darauf, dass der Code trotzdem verständlich bleibt und dass Du ihn dem Tutor noch präzise erklären kannst.

Viel Spaß!
