MMProg: Praktikum: WiSe 2017/18: GameLoop01

Wechseln zu:Navigation, Suche

MMProg-Praktikum

Inhalt | Game Loop 01 | Ball 02 | Ball 03 | Ball 03b | Pong 01

Musterlösung: SVN-Repository

Inhaltsverzeichnis

1 Vorbereitung

Im SVN-Repository finden Sie zwei WebStorm-Projekte zum Thema GameLoop:

WK_GameLoop01
WK_GameLoop02

Sie können diese Projekte folgendermaßen auf Ihrem Rechner installieren:

`VCS` → `Checkout from Version Control` → `Subversion`
Sofern noch nicht geschehen: Klick auf das grüne Plus-Symbol → Repository URL:
`https://glossar.hs-augsburg.de/beispiel/tutorium/es6`
Klick auf die Pfeilspitze vor diesem Pfad → Klick auf die Pfeilspitze vor `game_loop` → Klick **auf**

`WK_GameLoop02`
`Checkout` (`WK_GameLoop01` dient nur didaktischen Zwecken und wird für diese Praktikumsaufgabe nicht benötigt.)
Speichern Sie das Projekt irgendwo auf Ihrem Rechner. **Achtung: Speichern Sie das Projekt nicht innerhalb eines anderen Projekts (insbesondere nicht in Ihrem neuen Projekt) oder innerhalb eines Ordners, der bereits unter SVN-Kontrolle steht. Speichern meine Projekte in einem ganz anderen Ordner, der z. B. `KowaBeispiele` heißen könnte.**
Öffnen Sie das Projekt und geben Sie im WebStorm-Terminal `npm install` (oder kurz `npm i`) ein, um alle benötigten Node.js-Module zu installieren.

Machen Sie sich mit den Projekten vertraut. Die Web-Anwendung `src/js/app` des zweiten Projekts dient als Ausgangsbasis für diese Praktikumsaufgabe.

Im ersten Beispiel finden Sie diverse Game-Loop-Varianten. Schrittweise werden potentielle Probleme behoben. Dieses Projekt verfolgt das didaktische Ziel, Ihnen die Probleme und potentielle Lösungen im Zusammenhang mit JavaScript-Animationen zu verdeutlichen.

Im zweiten Beispiel wurde eine einfache Game-Loop-Klasse realisiert, die sie für dieses Praktikum nutzen können und sollten. Diese Klasse basiert auf den Ergebnissen des ersten Projekts, stellt aber – im Gegensatz zu einigen Web-Anwendungen des ersten Projekt – bislang keine Informationen über die aktuelle Frame-Rate bereit. Dies ist aber für diese Praktikumsaufgabe nicht von Interesse.

2 Aufgaben

Laden Sie das leere Projekt WK_GameLoop02_Empty auf Ihren Rechner (am Besten in den zuvor angelegten Ordner `KowaBeispiele`). **Installieren Sie diesmal aber nicht die Node.js-Module**, das machen Sie später. Sie finden das leere Projekt im zuvor aktivierten Repository-Pfad `https://glossar.hs-augsburg.de/beispiel/tutorium/es6` im Unterordner `empty`.

Erstellen Sie ein neues Projekt `praktikum01` und kopieren Sie die Ordner `src` und `web` (samt Inhalt) sowie alle Dateien, die Sie im Wurzelverzeichnis des Projekts `WK_GameLoop02_Empty` finden, mittels `Ctrl-/Apfel-C` `Ctrl-/Apfel-V` in Ihr eigenes Projekt. (Die Frage, ob WebStorm seinen eigenen File Watcher zum Übersetzen von ES6-Code in ES5-Code verwenden soll, beantworten Sie bitte mit „No“. Das erledigt Webpack für Sie.)

Sie können Ihr Projekt zur Übung auch im Subversion-Repository speichern. Das ist aber nicht so wichtig.

Nun können Sie in Ihrem eigenen Projekt die benötigten Node.js-Module installieren: `npm i`.

In Ihrem Projekt finden Sie nun 10 Web-Anwendungen vor: `index01.html` verwendet die gepackte Version von `app01.js`, die ihrerseits das Spiel `game01.js` einbindet. `index02.html`, `index03.html` etc. sind analog aufgebaut. Wenn Sie irgendeine dieser Anwendungen im Browser öffnen, sehen Sie eine Eule in der linken oberen Bildschirmecke. Diese sollen sie auf unterschiedliche Art und Weise animieren. Lösen Sie jede der folgenden Aufgaben in einer der vorgegebenen Apps. Scheuen Sie sich nicht davor, eigene Experimente durchzuführen. Wenn die vorgegebenen Apps nicht ausreichen sollten, legen Sie einfach noch ein paar an. (Vergessen Sie in diesem Fall nicht, `webpack.config.js` entsprechend zu erweitern.)

Schreiben Sie Ihre Lösungen der Aufgabe i in die Datei `game`i`.js`. Am einfachsten ist es, wenn Sie jeweils die Lösung der vorangegangenen Aufgabe kopieren und diese Kopie dann weiterentwickeln.

2.1 Aufgabe 1

Lassen Sie die Eule horizontal vom linken bis zum rechten Fensterrand des Browsers fliegen.

Tipp: http://ryanve.com/lab/dimensions/

Berücksichtigen Sie, dass die Eule 150 Pixel breit ist und das der Ankerpunkt im linken oberen Eck des zugehörigen Bildes liegt. Sie müssen daher im Eulen-Objekt auch die Breite der Eule speichern und diese Breite bei der Kollissionserkennung und -behandlung berücksichtigen.

Beachten Sie bitte, dass Sie zur Lösung dieser Aufgabe einfach die Datei `game01.js` anpassen müssen. Lassen Sie sich durch den Code der Datei `owl.js` aus dem Beispiel `WK_GameLoop02` inspirieren. (Sie können auch `owl_interpolate.js` verwenden, aber das verwirrt Sie vermutlich zurzeit mehr, als dass es Inhen nützt.) Eine Reset-Funktion, wie sie im genannten Beispiel definiert wird, brauchen Sie nicht zu implementieren, da Sie das „Spiel“ nicht mit Hilfe von irgendwelchen Buttons anhalten und zurücksetzen werden. Dies gilt auch für alle folgenden Aufgaben.

Vergessen Sie `grunt` oder besser noch `grunt watch` nicht. :-)

2.2 Aufgabe 2

Lassen Sie die Eule nicht waagerecht, sondern in der horizontalen Mitte des Browserfensters senkrecht von Fenstrrand zu Fensterrand fliegen.

Um die Aufgabe zu lösen, müssen Sie für die Eule zusätzlich eine y-Position und eine y-Geschwindigkeit definieren. Die Berechnung der aktuellen y-Position funktioniert analog zur Berechnung der aktuellen x-Position.

Vergessen Sie nicht auch die Kollissionserkennung und -behandlung für die beiden Bildschirmseiten `top` und `botton` zu implementieren.

Und Sie müssen natürlich die Render-Funktion anpassen, sodass die aktuelle y-Position beim Rendern auch berücksichtigt wird.

2.3 Aufgabe 3

Lassen Sie die Eule von der Browsermitte aus schräg über den Bildschirm fliegen. In x-Richtung soll sie doppelt so schnell sein (200 Pixel/s) wie in y-Richtung (100 Pixel/s).

2.4 Aufgabe 4

Setzen Sie die x-Start-Position der Eule auf `-100` und starten Sie die Web-App. Den Effekt, den Sie sehen, nennt man Penetration. Die Eule hängt in der Wand fest, da Sie in einem Schritt nicht den Kollissionsbereich (die linke Wand) verlässt. Daher besteht die Kollission fort und im nächsten Schritt ändert sie wieder ihre Flugrichtung.

Verbessern Sie das, indem Sie die Kollisionserkennung und -behandlung verbessern:

```
if (v_owl.x <= v_stage.left)
{
  v_owl.vx = Math.abs(v_owl.vx);
}
if (v_owl.x >= v_stage.right - v_owl.width)
{
  v_owl.vx = -Math.abs(v_owl.vx);
}
```

Passen Sie die Kollisionserkennung und -behandlung in y-Richtung analog an.

Sie können und sollten sogar noch einen Schritt weiter gehen und die Eule wieder zurück auf die Bühne schieben, wenn Sie in die Wand eingedrungen ist. In die Wand einzudringen ist zwar physikalisch nicht möglich, aber leider in einer Simulation der physikalischen Welt nur schwer zu vermeiden, da die Position nur alle 16,7 ms berechnet wird. War die Eule zu einem Zeitpunkt noch vor der Mauer, kann Sie beim nächsten Schritt schon drinnen stecken. Insbesondere bei sehr schnellen Objekten kann das dann zu PEnetrations-Effekten führen.

Also schieben Sie die Eule besser zurück auf die Bühne.

```
if (v_owl.x <= v_stage.left)
{
  v_owl.x = v_stage.left;
  v_owl.vx = Math.abs(v_owl.vx);
}
if (v_owl.x >= v_stage.right - v_owl.width)
{
  v_owl.x = v_stage.right - v_owl.width;
  v_owl.vx = -Math.abs(v_owl.vx);
}
```

Passen Sie die Kollisionserkennung und -behandlung in y-Richtung wiederum analog an.

2.5 Aufgabe 5

Ändern Sie die Kollisionserkennung und -behandlung so ab, dass die Eule ausgehend von der linken oberen Fensterecke im Uhrzeigersinn sich immer entlang des Fensterrandes bewegt.

Beachten Sie bitte: Hier ist es besonders wichtig, dass Sie die Eule wieder auf die Bühne zurückbewegen, wenn sie mit einer Wand kollidiert. Ansonsten verschwindet die Eule schnell im Nirgendwo.

(Anmerkung: Dies war einmal eine Aufgabe im Rahmen des Prüfungspraktikums, wobei die Lösung zur 3. Aufgabe vorgegeben war.)

2.6 Aufgabe 6

Kopieren Sie diesmal nicht die Lösung von Aufgabe 5, sondern von Aufgabe 4. Im folgenden arbeiten Sie wieder mit der normalen Kollissionserkennung und -behandlung. Positionieren Sie die Eule allerdings wieder im linken oberen Eck (in Aufgabe 4 hatten Sie sie außerhalb der Bühne platziert).

Die neue Position der Eule berechnet man mit Hilfe der Geschwindigkeit (velocity). Doch auch die Geschwindigkeit kann sich ändern. Dazu benötigt man die Beschleunigung (acceleration).

Fügen Sie zu Ihrer Eule zwei weitere Attribute `ax` (Beschleunigung in x-Richtung) und `ay` (Beschleunigung in y-Richtung) hinzu. Setzen Sie die Initialwerte auf `400` (Pixel pro Sekunde) bzw. `200` (Pixel pro Sekunde). Das heißt, Sie möchten, dass die Eule in jeder Sekunde um 400 bzw. 200 Pixel pro Sekunde mehr zurücklegt als zuvor.

Wenn Sie jetzt die Web-App starten, beschleunigt die Eule allerdings noch nicht.

In Ihrem Code wird die Position 60 mal pro Sekunde mit Hilfe des folgenden Codes aktualisiert:

```
v_owl.x += v_owl.vx * p_frac_s;
v_owl.y += v_owl.vy * p_frac_s;
```

Die Geschwindigkeit wird zur Position hinzuaddiert. Allerdings ist die Geschwindigkeit in Pixeln pro Sekunde angegeben. Die Modellaktualisierung passiert jedoch alle $0,0167$ Sekunden. In dieser Zei bewegt sich die Eule nur um das $0,0167$-fache (= $1,67$%) der Sekundengeschwindigkeit weiter.

Auf genau dieselbe Weise wird die aktuelle Geschwindigkeit mit Hilfe der Beschleunigung berechnet:

```
v_owl.vx += v_owl.ax * p_frac_s;
v_owl.vy += v_owl.ay * p_frac_s;
```

Fügen Sie diesen Code in Ihre Model-Update-Funktion ein und starten Sie die Eule erneut. Wenn Sie jetzt Ihre We-App laufen lassen, stellen Sie fest, dass die Eule zunächst beschleunigt, nach eine Kollission aber wieder abbremst. NAch der nächsten Kollission beschleunigt sie wieder.

Um diesen Effekt zu vermeiden, müssen Sie jedes mal, wenn Sie bei der Kollissionsbehandlung das Vorzeichen der Geschwindigkeit ändern, das Vorzeichen der zugehörigen Beschleunigung analog ändern.

Beispielsweise mus der Code

```
v_owl.vx = -Math.abs(v_owl.vx);
```

um

```
v_owl.ax = -Math.abs(v_owl.ax);
```

ergänzt werden.

2.7 Aufgabe 7

Schreiben Sie die Kollissionsbehandlung so um, dass im Falle einer Kollission der Eule mit dem unteren Fensterrand sowohl die Geschwindigkeit als auch die Beschleunigung jeweils in x- und in y-Richtung auf Null gesetzt wird. Das heißt, die Eule bleibt bei einer Kollision mit dem unteren Rand stehen stehen.

2.8 Aufgabe 8

Positionieren Sie die Eule im linken oberen Eck, geben Sie ihr eine Geschwindigkeit von 300 Pixeln in x-Richtung und von 0 Pixeln in y-Richtung. Die Beschleunigung in x-Richtung beträgt 0, die Beschleunigung in y-Richtung 800. (Diese Zahlen sind gut für meinen Monitor geeignet, der eine Auflösung von 1600x900 hat. Wenn Sie einen deutlich kleineren oder größeren Monitor haben, müssen Sie die Zahlen evtl. anpassen.)

Welche Kurve beschreibt die Eule, wenn Sie die Web-App starten?

2.9 Aufgabe 9

Positionieren Sie die Eule im linken unteren Eck, geben Sie ihr eine Geschwindigkeit von 300 Pixeln in x-Richtung und von -400 Pixeln in y-Richtung. Die Beschleunigung in x-Richtung beträgt 0, die Beschleunigung in y-Richtung 200.

Welche Kurve beschreibt die Eule, wenn Sie die Web-App starten?

Jetzt fehlt nur noch en Gummiband, mit dem Sie die Eule beschleunigen können. :-)

2.10 Aufgabe 10

Machen Sie eigene Experimente. Bringen Sie z. B. die Eule dazu, wie ein Flummi zu springen, indem sie ihre Geschwindigkeit in x- und y-Richtung bei einem Bodenkontakt um einen gewissen Prozentsatz reduzieren, aber nicht gleich auf null setzen. (Die Beschleunigung, die die Erdanziehung simuliert, bleibt die ganze Zeit über gleich). Achtung: Bei einem Bodenkontakt, dreht sich das Vorzeichen der y-Geschwindigkeit um, das der x-Geschwindigkeit bleibt gleich.

Nervig ist, dass die Eule zum Schluss immer noch ganz leicht hüpft und gar nicht zur Ruhe kommt. Vielleicht können Sie auch dieses Problem beheben.

Oder probieren Sie etwas ganz anderes aus.

3 Quellen

Kowarschick (MMProg): Wolfgang Kowarschick; Vorlesung „Multimedia-Programmierung“; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 3 (Vorlesung)

Kategorien:
Multimedia-Programmierung/Tutorium
Praktikum:MMProg:WiSe 2017/18