# MMProg: Praktikum: WiSe 2018/19: Ball01



Dieser Artikel erfüllt die GlossarWiki-Qualitätsanforderungen **nur teilweise**:

| | | | | |
|---|---|---|---|---|
| **Korrektheit**: 3 (zu größeren Teilen überprüft) | **Umfang**: 4 (unwichtige Fakten fehlen) | **Quellenangaben**: 3 (wichtige Quellen vorhanden) | **Quellenarten**: 5 (ausgezeichnet) | **Konformität**: 3 (gut) |

**Vorlesung MMProg**

Inhalt | EcmaScript01 | EcmaScript02 | EcmaScript03 | Ball 01| Ball 02 | Ball 03 | Pong 01

**Musterlösung**: Web-Auftritt (Git-Repository)

## Inhaltsverzeichnis



# 1 Vorbereitung

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

WK_GameLoop01
WK_GameLoop02

Machen Sie sich mit den Projekten vertraut. Die Web-Anwendung `src/js/app` des zweiten Projektes 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 Game-Loop-Klasse realisiert, die sie für dieses Praktikum nutzen

können und sollten. Diese Klasse basiert auf den Ergebnissen des ersten Projektes, 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.

Importieren Sie das leere Git-Projekt Ball01 in WebStorm. Laden Sie annschließend mittels `npm i` alle benötigten Node.js-Module in das Projekt.

Sie können Ihr Projekt zur Übung auch in Ihrem Git-Repository speichern. Das ist aber nicht so wichtig. Falls Sie dies machen möchten, müssen Sie es zuvor von meinem (schreibgeschützten) Repository lösen:

```
git remote remove origin
git remote add origin https://gitlab.multimedia.hs-augsburg.de:8888/BENUTZER/Ball01.git
```

# 2 Aufgaben

In Ihrem Projekt finden Sie 10 Web-Anwendungen vor: `index01.html` verwendet die JavaScript-Datei `app01.js`, die ihrerseits das Spiel `ball01.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. Beachten Sie, dass im Folgenden der Avatar nicht „Eule“, sondern „Ball“ genannt wird. Die Eule ist lediglich eine View des ballartigen Objektes.

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.

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

# 2.1 Aufgabe 1

Passen Sie zur Lösung dieser Aufgabe die Datei `src/js/ball/ball01.js` an. Lassen Sie sich durch den Code der Datei `ball.js` aus dem Beispiel `WK_GameLoop02` inspirieren. (Sie könnten auch `ball_interpolate.js` verwenden, aber das verwirrt Sie vermutlich zurzeit deutlich mehr, als dass es Ihnen nützt.)

**Lassen Sie den Ball horizontal vom linken bis zum rechten Fensterrand des Browsers fliegen.**

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

Berücksichtigen Sie, dass der Ball 150 Pixel breit ist, d. h. einen Radius von 75 Pixel hat, und dass der Ankerpunkt im linken oberen Eck des zugehörigen Bildes liegt. Sie müssen im Ball-Objekt auch die Radius des Balls speichern und diesen Wert bei der Kollissionserkennung und -behandlung berücksichtigen. Übrigens, wenn Sie `v_ball_view` wie in `ball.js` von `WK_GameLoop02` definieren, können Sie mittels `v_ball_view.offsetWidth/2;` die aktuelle Breite des Ballobjektes aus der View auslesen.

Ein Tipp noch: Wenn Sie das Ball-Objekt zunächst als leeres Objekt definieren und danach in der Reset-Funktion initialisieren, funktionieren Ihre Anwendungen auch noch, wenn Sie später einen Start/Stopp- und einen Pause-Button einfügen (vgl. z. B. `index_start_stop_toggling.html` in `WK_GameLoop02`). Sollten Sie keine Buttons zur Steuerung einfügen wollen, können Sie die Objekte auch direkt bei der Definition initialisieren.

Vergessen Sie nicht, die Sourcen mittels `npm run watch` zu übersetzen. Und beachten Sie, dass im Browser nur die Dateien `web/index`$i$`.html` funktionieren. Wenn Sie versuchen, `src/index`$i$`.html` erhalten Sie im Browser einen leeren Screen und in der Browser-Konsole drei Fehlermeldungen, dass gewissen Dateien nicht geladen werden können.

# 2.2 Aufgabe 2

Lassen Sie den Ball nicht waagerecht, sondern in der horizontalen Mitte des Browserfensters senkrecht von Fensterrand zu Fensterrand fliegen.

Um die Aufgabe zu lösen, müssen Sie für der Ball 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 der Ball von der Mitte des Browserfensters aus schräg über den Bildschirm fliegen. In x-Richtung soll er doppelt so schnell sein (200 Pixel/s) wie in y-Richtung (100 Pixel/s).

# 2.4 Aufgabe 4

Setzen Sie die x-Start-Position des Balls 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 nicht den Kollissionsbereich (die linke Wand) in einem Schritt 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_ball.x <= v_stage.left)
{ v_ball.vx = Math.abs(v_ball.vx);
}
if (v_ball.x >= v_stage.right - 2*v_ball.r)
{ v_ball.vx = -Math.abs(v_ball.vx);
}
```

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

Sie können und sollten sogar noch einen Schritt weiter gehen und den Ball wieder zurück auf die

Bühne schieben, wenn er 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 der Ball zu einem Zeitpunkt noch vor der Mauer, kann er beim nächsten Schritt schon drinnen stecken. Insbesondere bei sehr schnellen Objekten kann das dann zu den beobachteten Penetrationseffekten führen.

Also schieben Sie den Ball besser zurück auf die Bühne.

```
if (v_ball.x <= v_stage.left)
{ v_ball.x = v_stage.left;
  v_ball.vx = Math.abs(v_ball.vx);
}
if (v_ball.x >= v_stage.right - 2*v_ball.r)
{ v_ball.x = v_stage.right - 2*v_ball.r;
  v_ball.vx = -Math.abs(v_ball.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 sich der Ball ausgehend von der linken oberen Fensterecke im Uhrzeigersinn immer entlang des Fensterrandes bewegt.

Beachten Sie bitte: Hier ist es besonders wichtig, dass Sie den Ball wieder auf die Bühne zurückschieben, wenn er mit einer Wand kollidiert. Ansonsten verschwindet der Ball 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 die von Aufgabe 4. Im Folgenden arbeiten Sie wieder mit der normalen Kollissionserkennung und -behandlung. Positionieren Sie den Ball allerdings wieder im linken oberen Eck (in Aufgabe 4 hatten Sie sie außerhalb der Bühne platziert).

Die neue Position des Balls berechnen Sie mit Hilfe der Geschwindigkeit (velocity, `vx`, `vy`). Doch auch die Geschwindigkeig kann sich im Laufe der Zeit ändern. Dazu benötigt man die Beschleunigung (acceleration, `ax`, `ay`).

Fügen Sie zu Ihrem Ball 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 der Ball in jeder Sekunde um 400 bzw. 200 Pixel pro Sekunde mehr zurücklegt als zuvor.

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

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

```
v_ball.x += v_ball.vx * p_delta_s;
v_ball.y += v_ball.vy * p_delta_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 Zeit bewegt sich der Ball nur um das $0,0167$-fache (= $1,67$%) der Sekundengeschwindigkeit weiter. Dieser Faktor ist im Attribut `p_delta_s` enthalten. Dieser wird mit $1/60 = 0,0167$ initialisiert, sofern der Update-Methode kein anderer Wert übergeben wird.

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

```
v_ball.vx += v_ball.ax * p_delta_s;
v_ball.vy += v_ball.ay * p_delta_s;
```

Fügen Sie diesen Code in Ihre Model-Update-Funktion ein und starten Sie den Ball erneut. Wenn Sie jetzt Ihre Web-App laufen lassen, stellen Sie fest, dass der Ball zunächst beschleunigt, nach eine Kollision aber wieder abbremst. Nach der nächsten Kollision beschleunigt er 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 muss der Code

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

um

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

ergänzt werden.

# 2.7 Aufgabe 7

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

# 2.8 Aufgabe 8

Positionieren Sie der Ball im linken oberen Eck, geben Sie ihm 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 der Ball, wenn Sie die Web-App starten?

# 2.9 Aufgabe 9

Positionieren Sie den Ball im linken unteren Eck, geben Sie ihm 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 der Ball, wenn Sie die Web-App starten?

Allerdings ist es nicht ganz sauber, die Erdanziehung mittels `v_ball.ax` zu simulieren. Diese Beschleunigung ist die Beschleunigung, die vom Ball selbst ausgeht (die also von der Eule per Muskelkraft geleistet wird). Wenn der Ball seine Bewegungsrichtung ändert, ändert sich auch die Richtung der Beschleunigung (eine Eule beschleunigt immer in Flugrichtung, vgl. Aufgabe 6). Die Richtung der Erbbeschleunigung ändert jedoch bei einer Kollision nicht.

Verbessern Sie daher Ihre Anwendung vorlgendermaßen:

Fügen Sie eine Variable `v_g = 981` in die `let`-Anweisung ein. (981 wurde in Anlehnung an die Gravitationskonstante der Erde gewählt.)
Reduzieren Sie die Eigenbeschleunigung der Eule in $y$-Richtung auf . (Jetzt sollte sich die Eule wieder geradlinig bewegen.)
Ersetzen Sie un der Update-Funktion

```
v_ball.vy += v_ball.ay * p_delta_s;
```

durch

```
v_ball.vy += (v_ball.ay+v_g) * p_delta_s;
```

Das heißt, berücksichtigen Sie bei der Berechnung der $y$-Geschwindigkeit des Balles zusätzlich die Erdanziehungskraft. Nun sollte die Eule wieder einen sauberen Parabelflug vollführen.

Alles, was jetzt noch fehlt, ist ein Gummiband, mit dem Sie die Eule beschleunigen können. :-)

# 2.10 Aufgabe 10

Machen Sie eigene Experimente. Bringen Sie z. B. den Ball dazu, wie ein Flummi zu springen, indem sie seine 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 ändert sich dagegen nicht.

Nervig ist, dass der Ball 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

1. **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 2018/19