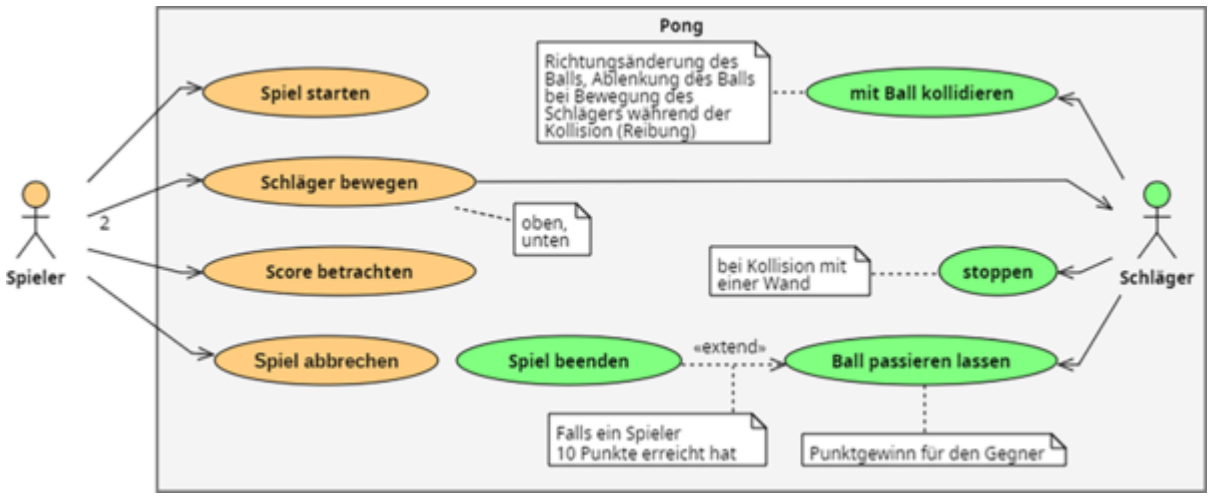
MMProg: Praktikum: WiSe 2017/18: Pong01

Wechseln zu:Navigation, Suche

Musterlösung: WK_Pong01, WK_Pong02

Inhaltsverzeichnis

1 Use Cases



Use Cases des Spiels Pong

Ziel dieser Aufgabe ist es, eine einfach Variante des Spieleklassikers Pong zu implementieren.

Pong wird von zwei Spielern gespielt. Ein Spieler kann das Spiel starten (Startknopf) und vorzeitig beenden (Stopp-Knopf). Nach Spielstart können die beiden Spieler jeweils einen Schläger am linken bzw. rechten Rand des Spielfeldes mit Hilfe der Richtungstasten des Keyboards nach oben und unten bewegen. Wenn ein Schläger mit einer Seitenwand kollidiert stoppt er. Er kann vom zugehörigen Spieler nur noch in Gegenrichtung bewegt werden.

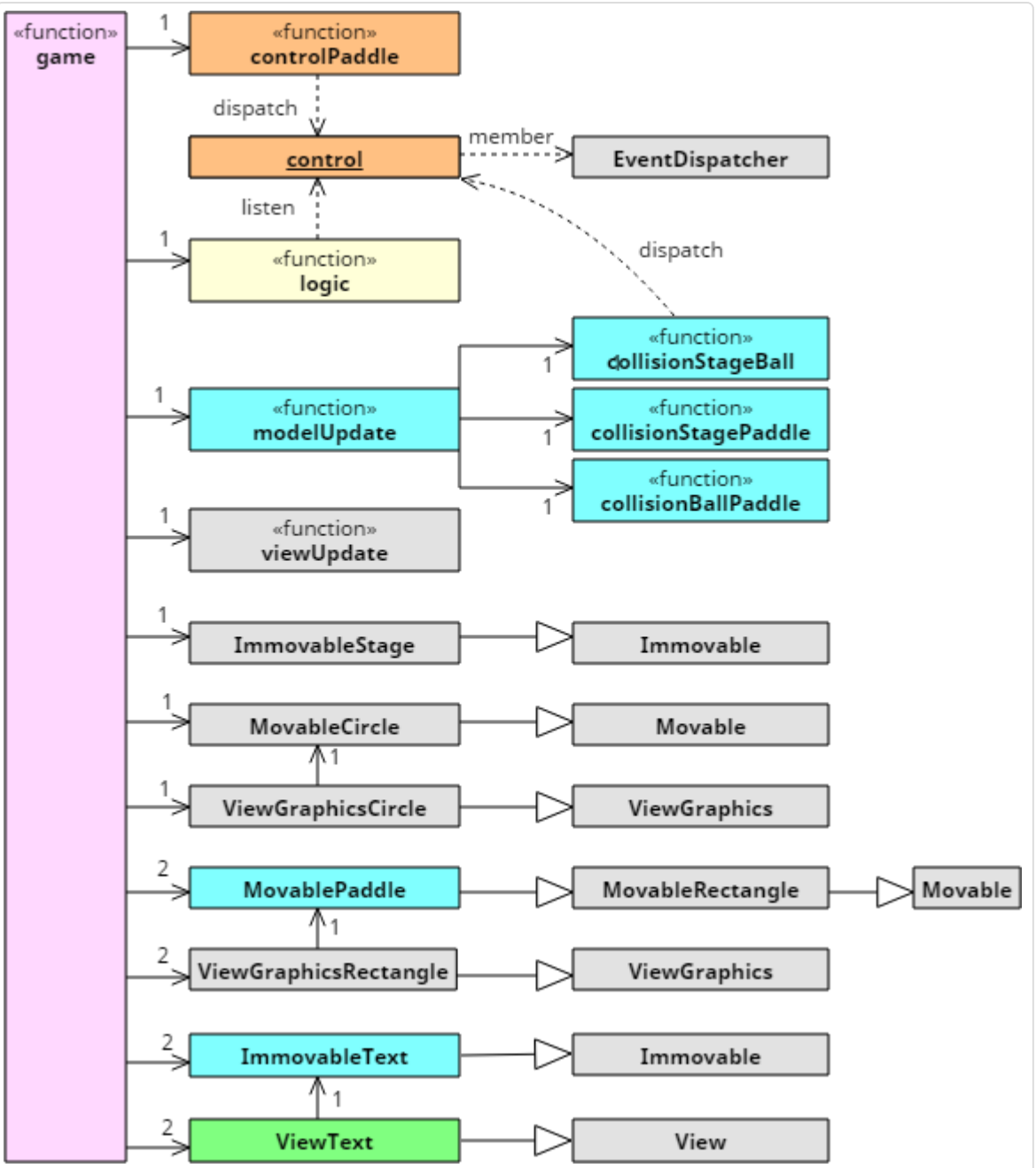
Nachdem das Spiel gestartet wurde, bewegt sich der Ball geradlinig im Spielfeld, wobei die Startrichtung zufällig gewählt wird. Kollisionen mit der oberen oder unteren Wand haben eine Richtungsänderung des Balls zur Folge (Einfallswinkel = Ausfallwinkel).

Eine Kollision eines Schlägers mit dem Ball hat ebenfalls eine Richtungsänderung des Balls zu Folge. Wird der Schläger im Moment der Kollision bewegt, so wird der Ball abhängig von der Bewegungsrichtung und Geschwindigkeit des Schlägers abgelenkt.

Mit den Wänden hinter den Schlägern kollidiert der Ball nicht. Passiert der Ball einen Schläger, verlässt er die Bühne durch Wand hinter dem Schläger und der der Gegner, d. h. der Akteur, der den anderen Schläger bedient, erhält einen Punkt.

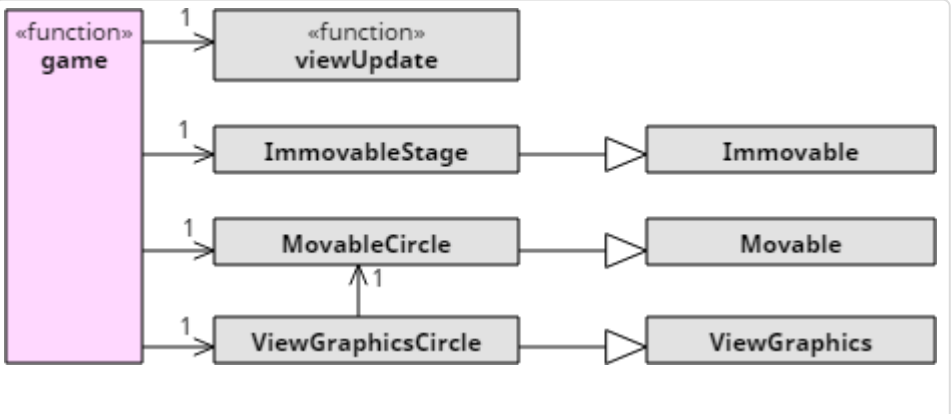
Ziel des Spiels ist es, den Ball möglichst lange im Spiel zu halten. Der aktuelle Punktestand (Score) wird den beiden Spielern jederzeit angezeigt.

2 Moduldiagramm



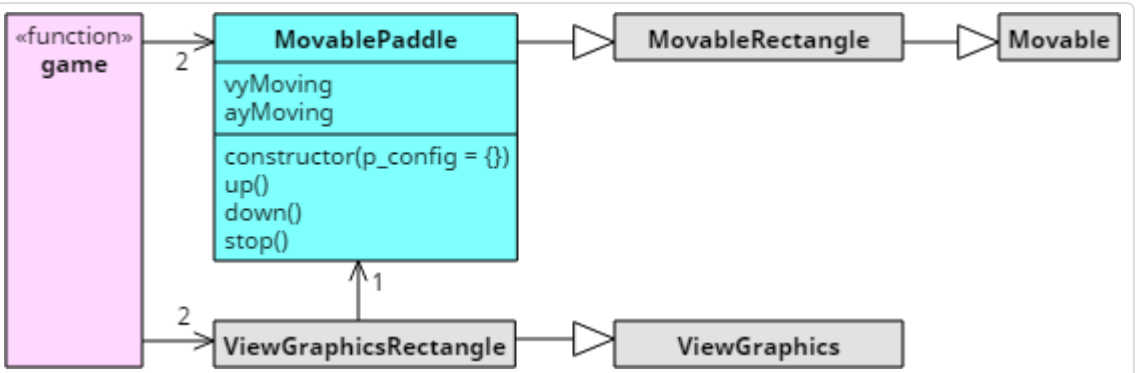
Klassendiagramm von `pong01` (ohne Properties)

In diesem Diagramm sind die wichtigsten Module der Anwendung Pong aufgelistet. Grau markiert sind Module aus der `wk`-Library, die unverändert wiederverwendet werden können (aber natürlich nicht müssen). Die farbig markierten Module müssen entweder erstellt oder – wenn sie schon existieren – an die Gegebenheiten der neuen Anwendung angepasst werden.



Stage-Modul, Ball-Module und View-Update-Modul

Für Pong wird eine Bühne benötigt, ein Ball und ein View-Update-Modul. Diese Element gibt es auch schon im der Mini-Moorhuhn-App (`game00`) und können eins zu eins übernommen werden. (Anmerkung: In Pong02 werden die Modelle die Views per Events über Änderungen benachrichtigen. Dann ist das View-Update-Modul überflüssig.)

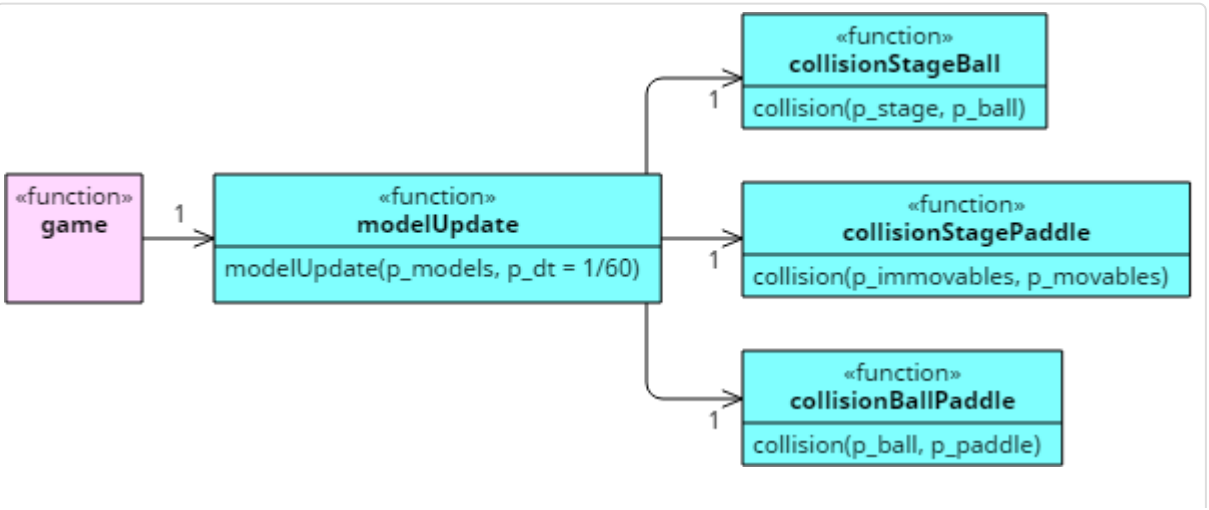


Paddle-Module

Neu hinzu kommen 2 Schläger (*paddles*), die es in den bisherigen Anwendungen noch nicht gab. Ein Schläger wird mit Hilfe eines beweglichen Rechtecks dargestellt. Das heißt, die Module `MovableRectangle` und `ViewGraphicsRectangle` aus der `wk`-Library kommen zum Einsatz. Falls eine gute Textur für den Schläger zur Verfügung steht, könnte man auch `ViewTextureRectangle` einsetzen. Im Break-out-Klassiker Bolo von Atari ist der Schläger sogar animiert (vgl. YouTube: Bolo (Atari ST) ab der vierten Minute). Dies könnte man mit der Klasse `ViewAnimatedRectangle` erreichen.

Es braucht nur die Klasse `MovablePaddle` implementiert zu werden. Diese erbt alle Attribute der Klasse `MovableRectangle` und definiert noch fünf weitere Attribute und Methoden:

`vyMoving`: Geschwindigkeit des Schlägers in y-Richtung, solange er sich bewegt
`ayMoving`: Beschleunigung des Schlägers in y-Richtung, solange er sich bewegt
`up`: setzt die aktuelle Geschwindigkeit des Schlägers auf `-vyMoving` und die Beschleunigung auf `-ayMoving`, sofern die aktuelle Geschwindigkeit gleich ist
`down`: setzt die aktuelle Geschwindigkeit des Schlägers auf `+vyMoving` und die Beschleunigung auf `+ayMoving`, sofern die aktuelle Geschwindigkeit gleich ist
`stop`: setzt die aktuelle Geschwindigkeit des Schlägers auf und die Beschleunigung auf , sofern die aktuelle Geschwindigkeit ungleich ist

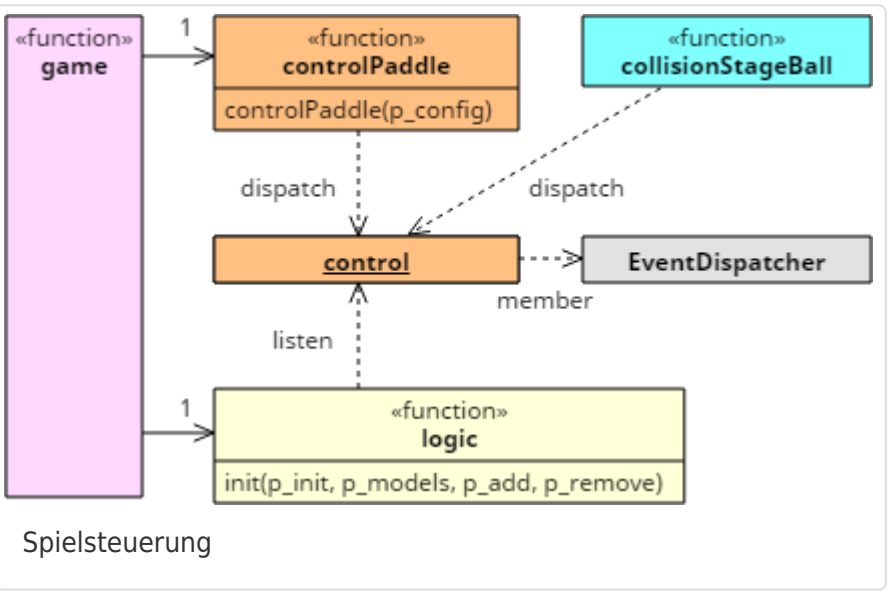


Kollisionserkennung und -behandlung

Im Spiel Pong gibt es drei Arten von Kollisionen, die behandelt werden müssen:

Einer der Schläger kollidiert mit der Wand: Der Schläger wird automatisch gestoppt.
Der Ball kollidiert mit einem Schläger: Der Ball ändert die Bewegungsrichtung.
Der Ball kollidiert mit der Wand: Der Ball ändert die Richtung (obere oder untere Wand) oder er verlässt die Bühne (rechte oder linke Wand). Über das Verlassen der Bühne muss die Logik informiert werden, da dies Auswirkungen auf den Spielstand hat.

Für jede der drei Kollisionsarten wird eine Kollisionsfunktion implementiert: `collisionStagePaddle`, `collisionBallPaddle` und `collisionStageBall`. Aufgerufen werden diese drei Methoden für alle fünf Fälle (es gibt je zwei Schläger, die mit der Wand oder dem Ball kollidieren können) in der Funktion `modelUpdate`. Leider muss diese Funktion dazu händisch angepasst werden. Besser wäre es, wenn man die Kollisionstests direkt in der Konfigurationsdatei festlegen könnte (Ausblick: Das wird in Pong02 realisiert).



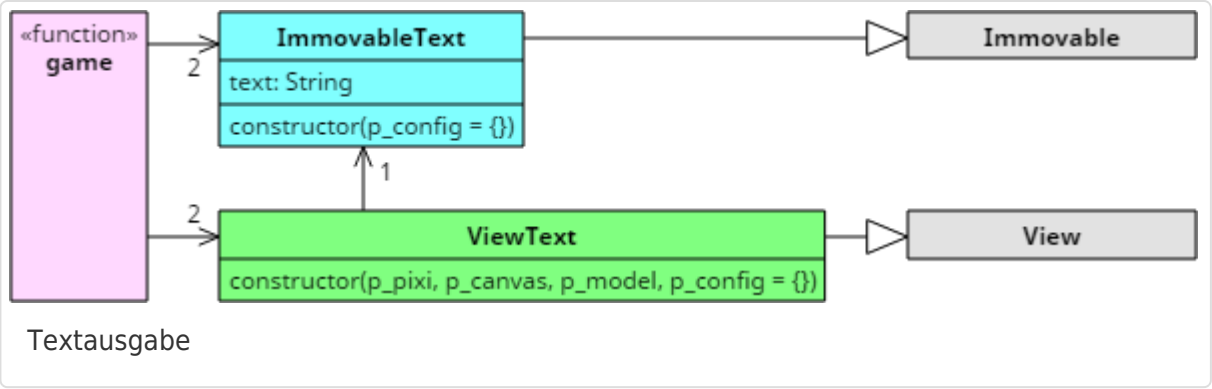
Spielsteuerung

Das Spiel wird mit Hilfe von Controllern und der Spiellogik gesteuert. Die Logik wird vom Game-Modul initialisiert. Ihr werden grundsätzlich irgendwelche Initialisierungsdaten, eine Liste (genauer: ein Objekt) mit allen Models sowie zwei Methoden zum Erstellen und Löschen von Spielobjekten übergeben. Das Logik-Modul greift auf das `control`-Objekt zu, welches im Falle von spielwichtigen Ereignissen Nachrichten in Form von Events verschickt. Während der Initialisierungsphase registriert sich das Logikmodul für diese Nachrichten. Die zugehörigen Callback-Funktionen sind für eigentliche Steuerung des Spiels zuständig.

Im Falle von Pong gibt es fünf Ereignisse, auf die das Logikmodul reagieren muss:

Ein Spieler möchte seinen Schläger steuern: Schläger nach oben, Schläger nach unten, Schläger anhalten
Der Ball verlässt hinter einem Schläger die Bühne: linke Seite, rechte Seite der Bühne

Für Nachrichten vom ersten Typ ist das Modul `controlPaddle` zuständig, für Nachrichten vom zweiten Typ das Kollisionsmodul `collisionStageBall`. (Anmerkung: Es gibt noch einen dritten potentiellen Sender von spielwichtigen Nachrichten: Die Views. Bei Pong gibt es keine derartige View, aber im Falle des Moorhuhnspiels existieren diese: Bei einem Klick auf die grafische Darstellung eines Moorhuhns wird die Logik über den Treffer informiert.)



Da laut Use-Case-Diagramm die Spieler den Score, d. h. die Zahl der eigenen und der gegnerischen Treffer (jederzeit) einsehen können, werden zwei Textfelder benötigt, um die Trefferzahlen zu visualisieren. Zu diesen Zweck werden die Klasse `ImmovableText` als Unterklasse von `Immovable` und `ViewText` als Unterklasse von `View` implementiert. Einem `ImmovableText`-Objekt werden die für unbewegliche Objekte üblichen Konfigurations-Daten übergeben (Position, Ankerpunkt etc.). `ViewText`-Objekte werden mit Hilfe der Pixi.js-Klasse PIXI.Text realisiert. Im Konfigurationsobjekt können alle Style-Attribute angegeben werden, die für Pixi.js-Textobjekte erlaubt sind: PIXI.TextStyle. Das Konfigurationsobjekt wird einfach als Style-Objekt an das neu erstelle Text-Objekt weitergereicht.

Achtung: Die in der aktuellen Implementierung der Klasse `Immovable` verwendete Berechnung des Ankerpunktes ist für Text-Objekte nicht geeignet. Immovable berechnet den Ankerpunkt (relative Position) aus dem Pivot-Punkt (absolute Position) und der Breite bzw. Höhe des Objektes. Das ist im Falle von Text-Objekten leider nicht so einfach möglich, da die Breite und Höhe einer Textbox von Font, Schriftgröße und Laufweite des Textes abhängt. Das heißt, sobald die View für einen gegebenen Text die Textbox erstellt hat, müsste sie das Modell über die aktuelle Breite informieren, damit diese den Ankerpunkt korrekt einstellen kann. Dies ist etwas komplexer und wird daher erst in Pong02 angepackt. Bis dahin ist es leider nicht möglich, einen Text rechtsbündig (Ankerpunkt in x-Richtung: 1) oder zentriert (Ankerpunkt in x-Richtung: 0,5) zu setzen.

3 Aufgaben

Laden Sie das leere Projekt WK_Pong01_empty auf Ihren Rechner. **Installieren Sie aber nicht die Node.js-Module**, das machen Sie später. Sie finden das leere Projekt im Repository-Pfad `https://glossar.hs-augsburg.de/beispiel/tutorium/es6` im Unterordner `empty`.

Erstellen Sie ein neues Projekt `pong01` und kopieren Sie die Ordner `src` und `web` (samt Inhalt) sowie alle Dateien, die Sie im Wurzelverzeichnis des Projektes `WK_Pong01_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 zwei Web-Anwendungen: `index00.html` und `index01.html`, die wie üblich eine zugehörige JavaScript-App einbinden. Beide Apps sind identisch aufgebaut. Es handelt sich um eine leicht modifizierte Variante der Musterlösung 6 von Praktikumsaufgabe Ball03b. Die „Moorhühner“ können per Mausklick „abgeschlossen“ werden.

Um Aufgabe 1 zu lösen, sollten Sie `game01` schrittweise abändern. Überprüfen Sie nach jedem

Teilschritt, ob die Anwendung noch läuft.

4 Aufgabe 1

Zur Implementierung von Pong können Sie auf Module aus dem Praktikum zurückgreifen. Im obigen Diagramm sind nur einige Module farbig eingezeichnet. Diese Module müssen Sie entweder erstellen oder – wenn sie schon existieren – an Ihre Gegebenheiten anpassen.

Die grauen Module können Sie direkt aus der `wk`-Library einbinden, die Sie unter `src/js/lib/wk` finden. Der zugehörige Import-Befehl lautet jeweils

```
import ... from 'wk/...';
```

Mehrere Beispiele finden Sie in der Datei `game01/game.js`. Um die WebStorm-Fehlermeldungen zu vermeiden, die angeben, dass ein `wk`-Modul nicht gefunden werden kann, sollten Sie Folgendes machen:

Rechtsklick auf `src/js/lib/wk` → `Mark Directory as` → `Resource Root`

Falls die `wk`-Modul dann immer noch als fehlerhaft markiert werden, sollten Sie WebStorm neu starten.

Achtung: In der aktuellen Version 2017.2.5 von WebStorm werden Import-Anweisungen der Art `import ... from 'wk/.../MODUL.js';` als fehlerhaft angezeigt. In der ursprünglichen Version des Projektes `WK_Pong01_Empty` wurden die Module der `wk`-Bibliothek noch auf diese Weise importiert. Wenn Sie die WebStorm-Fehlermeldung stört, sollten Sie in diesen Import-Befehlen die Datei-Endung `.js` entfernen: `import ... from 'wk/.../MODUL';`. (Für Webpack wäre das nicht nötig, da Webpack weiterhin beide Schreibweisen unterstützt.) Die Importbefehle derjenigen Dateien, deren Importpfad mit `./` oder `../` dürfen Sie dagegen nicht ändern. Diese **müssen** auf `.js` enden, da anderenfalls die zugehörigen Module im Verzeichnis `node_modules` gesucht werden, wo sie sich aber nicht befinden.

4.1 Vögel durch einen Ball ersetzen

Als erstes sollten Sie die Zahl der Vögel reduzieren, indem Sie einfach in der Konfigurationsdatei `json/config01.json` den Counter `@count` in `model.bird` entfernen. Außerdem sollten Sie den String `"birds"` durch `"ball"` ersetzen und die Array-Klammer `[ ]` um das Huhn-Model-Konfigurationsobjekt entfernen. Letzteres ist wichtig, da die Logik – um eine neue Spielrunde starten zu können – direkt, d. h. per Namen auf das Ball-Objekt zugreifen wird, sobald es die Bühne verlässt. Ein namentlicher Zugriff auf Array-Objekte wird vom Game-Modul (noch) nicht unterstützt.

(Funktioniert das Spiel noch? Es sollte nur noch ein Huhn herumflattern.)

Ersetzen Sie jetzt das animierte Huhn durch einen farbigen Kreis. Ein Beispiel für eine zugehörige View-Konfiguration finden Sie in der Konfigurations-Datei der Musterlösung Ball03 (04a): `view.orange` oder `view.blue`.

Ersetzen Sie in Ihrer Konfigurationsdatei `view.hen` durch eine der beiden Graphics-Konfigurationen und ersetzen Sie den View-Bezeichner `"hen"` unter `model.ball.view` durch den von Ihnen gewählten View-Bezeichner.

Wenn Sie jetzt das Programm laufen lassen, fliegt kein Ball mehr über die Bühne. In der Browser-Konsole wird folgender Fehler angezeigt:

```
v_json_map[c_config_view.class] is not a constructor
```

Das liegt daran, dass Sie in der Konfigurations-Datei angegeben haben, dass die Klasse `ViewGraphicsCircle` (aus `wk/view/ViewGraphicsCircle`) zum Rendern der View verwendet werden soll. Das Huhn wurde dagegen mit `ViewAnimatedBird` gerendert.

Das Game-Modul `game/game01.js` kann grundsätzlich jede beliebige Klasse zum Rendern verwenden. Allerdings muss es diese importieren und unter dem Namen, der in der Konfigurationsdatei verwendet wird, in der Hashmap `v_json_map` einfügen.

Verbessern Sie `game/game01.js` entsprechend.

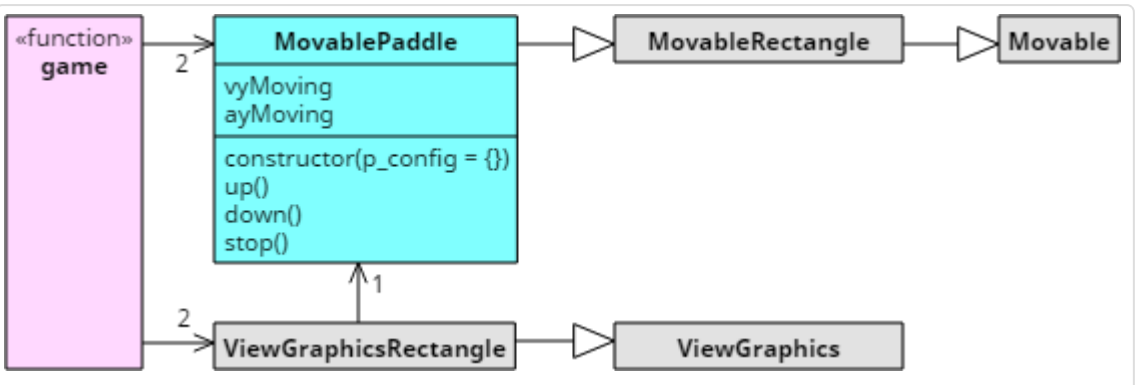
Wenn alles funktioniert, können Sie den Import der Klasse `ViewAnimatedBird` aus `game/game01.js` entfernen (`v_json_map` nicht vergessen). Die Datei `view/ViewAnimatedBird.js` können Sie löschen. Sie wird nicht mehr benötigt.

4.2 Feste Bühnengröße

Definieren Sie in der Konfigurationsdatei eine feste Bühnengröße (z. B. Breite: 600 Pixel, Höhe: 400 Pixel) **Achtung**: Geben Sie der Konfigurationsdatei jeweils eine Integerzahl und keinen String an! Das Hintergrundbild ist nicht sehr passend für Pong. Wenn Sie ein besseres haben, ersetzen Sie es (in der Datei `src/js/app01.js`), ansonsten löschen Sie in der Konfigurationsdatei einfach das Attribut `model.stage.view`.

Legen Sie die Startposition des Balls in die Mitte der Bühne und sorgen Sie dafür, dass er einen festen Radius hat. Die Geschwindigkeit sollte weiterhin zufällig gewählt werden. Allerdings werden Sie später vermutlich den Geschwindigkeitsbereich anpassen.

4.3 Die Schläger



Paddle-Module

Definieren Sie die Klasse `MovablePaddle` (`./model/MovablePaddle.js`). Sie erbt alle Attribute von `MovableRectangle` (`wk/model/MovableRectangle`) und enthält zunächst nur einen Konstruktor, der genauso definiert wird, wie in `wk/model/MovableRectangle`.

Ein Schläger wird vom Spieler gesteuert. Er kann ihn (indem er auf der Tastatur entsprechende Tasten

drückt) nach unten oder oben bewegen und er kann ihn wieder anhalten (indem er die jeweilige Steuertaste wieder loslässt). Ein Schläger hat neben den üblichen Attributen zwei weitere Attribute `vyMoving` und `ayMoving`, die festlegen, mit welcher Geschwindigkeit sich der Schläger in y-Richtung bewegen soll, solange der Spieler die entsprechende Taste bewegt. Schreiben Sie drei Methoden `down`, `up` und stop, die dafür sorgen, dass sich der Schläger in die richtige Richtung bzw. gar nicht bewegt.

```
down()
{ if (this.vy === 0)
  { this.vy = this.vyMoving;
    this.ay = this.ayMoving;
  }
}
```

Die Methode `up` wird analog mit umgekehrten Vorzeichen definiert und die Methode `stop` setzt die Geschwindigkeit und die Beschleunigung auf Null.

Importieren Sie `MovablePaddle` in `game/game01.js` und vergessen Sie nicht, die Klasse auch in `v_json_map` einzutragen. Und weil Sie schon gerade dabei sind, importieren Sie auch noch die Klasse `ViewGraphicsRectangle` aus der `wk`-Library.

Wenn Sie jetzt die Anwendung starten, wird diese Klassen noch nicht verwendet, aber es ist zumindest sichergestellt, dass sie gefunden werden.

Legen Sie jetzt in der Konfigurationsdatei die Modell zweier Paddles samt zugehöriger View an (wenn beide Paddle gleich aussehen sollen, dann reicht ein View-Objekt, das von beiden Paddle-Modellen referenziert wird). Als Klasse für die View verwenden Sie `ViewGraphicsRectangle`. Diese View Objekte werden genauso konfiguriert, wie `ViewGraphicsCircle`-Objekte.

Die Konfiguration der Paddle-Models ist etwas raffinierter. Bei den Paddles handelt es sich um schlanke (Breite), längliche (Höhe) Rechtecke , die jeweils parallel zu einer der senkrechten Seiten der Bühne stehen. Setzen Sie den Anchor des linken Paddles in die Mitte der linken Seite des Rechtecks (`xAnchor: 0.0, yAnchor: 0.5`) und den Anchor des rechten Paddles in die Mitte der rechten Seite des Rechtecks (`xAnchor: 1.0, yAnchor: 0.5`). Das Rechteck (genauer seinen Ankerpunkt) sollten Sie jeweils 5 Pixel vom Bühnenrand entfernt mittig platzieren (vgl. Musterlösung).

Für die Geschwindigkeitsattribute, die von den zuvor definierten Methoden verwendet werden, sollten Sie zunächst folgende Werte verwenden:

```
"vyMoving": 150,
"ayMoving": 500
```

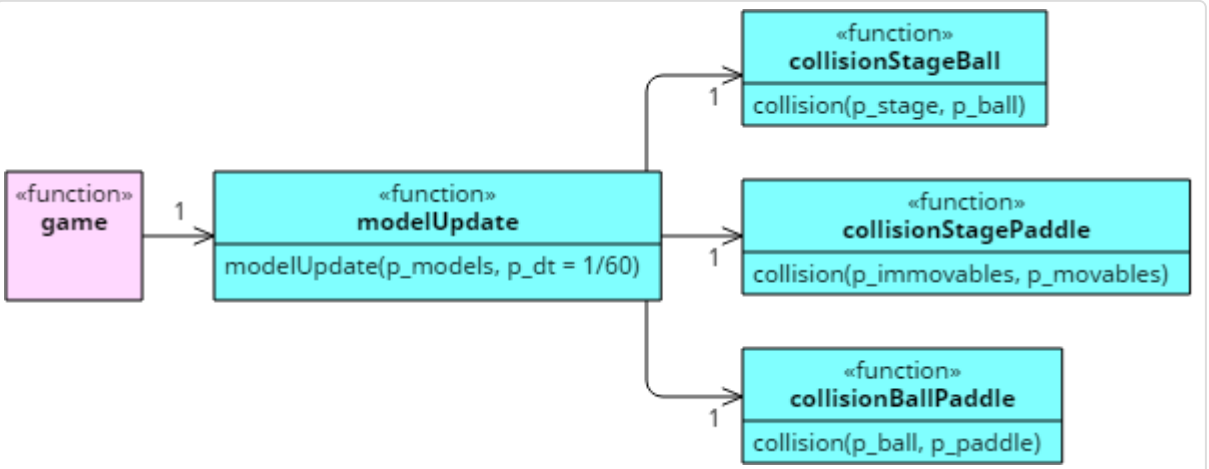
Damit wird erreicht, dass der Schläger sich beim Start zunächst langsam bewegt, aber relativ schnell beschleunigt. Später können Sie geeignetere Werte durch Experimentieren ermitteln.

Wenn Sie die Klasse `MovablePaddle` korrekt implementiert, die beiden Klassen `MovablePaddle` und `ViewGraphicsRectangle` korrekt in das Game-Modul importiert und die beiden Paddle samt View korrekt konfiguriert haben, sollten sich bei einem Start der App drei Element auf der Bühne befinden: ein Ball (der sich über die Bühne bewegt) und zwei Schläger (die sich nicht bewegen).

Testhalber können Sie in der Konfigurationsdatei einem Schläger mal eine Geschwindigkeit in y-

Richtung zuordnen. Sie werden feststellen, dass er sich permanent zwischen unterem und oberen Rand hin und her bewegt, ohne sich vom vertikalen Bühnenrand wegzubewegen.

4.4 Kollisionserkennung und -behandlung



Model-Update-Module

Es ist an der Zeit die Kollisionserkennung und -behandlung anzugehen. Es gibt drei unterschiedliche Fälle:

Schläger kollidiert mit der Wand: Falls der Schläger mit der Wand kollidiert, wird er mittels der zuvor implementierten Methode `stop` angehalten (und zurück auf die Bühne geschoben, falls er in die Wand eingedrungen ist). Es ist dann die Aufgabe des Spielers ihn mittels Tastatursteuerung in Gegenrichtung wieder in Bewegung zu setzen.
Ball kollidiert mit Schläger: Der Ball prallt am Schläger ab. Das funktioniert im Prinzip genauso, wie das Abprallen von der Wand: Bei einer Kollision wird die x-Geschwindigkeit negiert.
Ball kollidiert mit der Wand: Falls der Ball mit einer horizontalen Wand kollidiert, prallt er ab, wie bisher auch. Falls er mit einer senkrechten Wand kollidiert (rechte Seite des Balls kleiner linkem Rand der Bühne oder linke Seite des Balls größer rechtem Rand der Bühne!), wird die Logik informiert, dass der Ball von der Bühne verschwunden ist.

Insgesamt sind bei jeder Neuberechnung der Modellwelt fünf Fälle zu überprüfen:

Kollision vom Ball mit der Bühne
Kollision von Schläger1 mit der Bühne
Kollision von Schläger2 mit der Bühne
Kollision von Schläger1 mit dem Ball
Kollision von Schläger2 mit dem Ball

Legen Sie zunächst drei Dateien an:

```
model/CollisionStageBall.js
model/CollisionStagePaddle.js
model/CollisionBallPaddle.js
```

Fügen Sie jeweils eine Funktion (mit leerem Rumpf ein), die Sie exportieren:

```
collision(p_stage, p_ball)
collision(p_stage, p_paddle)
collision(p_ball, p_paddle)
```

Importieren Sie diese Funktionen in die Datei `model/ModelUpdate.js` und ersetzen Sie den Befehl

`collision(p_immovables, p_movables);` durch folgende fünf Befehle:

```
collisionStageBall  (p_models.stage, p_models.ball);
collisionStagePaddle(p_models.stage, p_models.paddle1);
collisionStagePaddle(p_models.stage, p_models.paddle2);
collisionBallPaddle (p_models.ball,  p_models.paddle1);
collisionBallPaddle (p_models.ball,  p_models.paddle2);
```

Beachten Sie, dass die Funktion `modelUpdate` im Parameter `p_models` auf alle Modelle, die in der Konfigurations-Datei im Objekt `model` definiert wurden, unter dem jeweiligen Konfigurationsnamen zugreifen kann. In der Musterlösung heißen die vier benötigten Objekte `stage`, `ball`, `paddle1` und `paddle2`. Wenn Sie Ihre Modell-Objekte in der Konfigurationsdatei anders benannt haben, müssen Sie die obigen Befehle entsprechend anpassen.

Anmerkung: Das Modul `collision.js` wird hier nicht verwendet. Es wäre sauberer, dieses Modul so zu verallgemeinern, dass es abhängig von den Klassen der zu testenden Objekte die jeweilige Kollisionsmethode aufrufen würde. Bei vielen Objekten, die kollidieren können, ist das besser, als Dutzende Einzelbefehle zur Kollisionserkennung zu schreiben. Aber hier würde das zu weit führen. Den Import-Befehl

```
import collision from './collision.js';
```

sollten Sie daher aus der Datei `modelUpdate.js` löschen.

Implementieren Sie nun die drei Methoden.

Kollision vom Ball mit der Bühne

Das ist zunächst ganz einfach. Kopieren Sie den Code aus `wk/model/collisionBB` und entfernen Sie die Tests für den linken und den rechten Bühnenrand. Später fügen Sie noch Code ein, der die Logik-Komponente informiert, wenn der Ball die Bühne verlässt.

Wenn Sie das Programm mehrfach starten, sollte jetzt der Ball vom oberen und unteren Rand abprallen, aber rechts und links von der Bühne verschwinden.

Kollision vom Schläger mit der Bühne

Dieser Test ist noch einfacher. Wenn der obere Rand des Schlägers kleiner oder gleich dem oberen Rand der Bühne ist, wird der Schläger auf die Bühne zurück geschoben (`p_paddle.top = p_stage.top;`) und angehalten `p_paddle.stop();`. Die Kollisionserkennung mit dem unteren Rand erfolgt auf die gleiche Art und Weise.

Kollision von Ball und Schläger

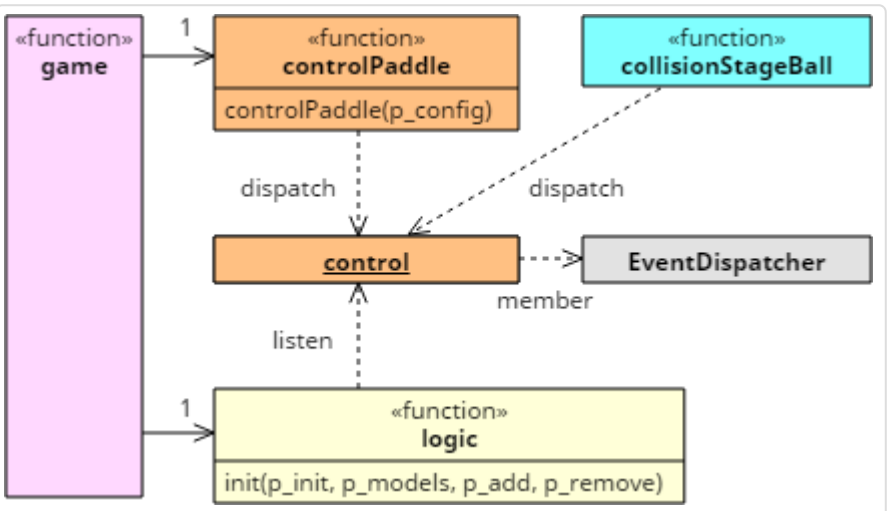
Das ist eine sehr aufwändige Operation, da 8 Fälle unterschieden werden müssten: Kollision des Balls mit einer der vier Seiten oder einer der vier Ecken. Folgender recht grober Test muss daher vorerst reichen (der Ball verhält sich allerdings ziemlich eigenartig, wenn er mit einer der Ecke oder Schmalseite des Schlägers kollidiert);

```
if (p_ball.y + 0.5*p_ball.r >= p_paddle.top &&
    p_ball.y - 0.5*p_ball.r <= p_paddle.btm
   )
{
  if (p_ball.vx > 0 &&  // The ball is moving from left to right.
      p_ball.rgt >= p_paddle.lft && p_ball.lft < p_paddle.rgt
     )
  { p_ball.rgt = p_paddle.lft;
    p_ball.vx = -p_ball.vx;
  }

  if (p_ball.vx < 0 &&  // The ball is moving from right to left.
      p_ball.lft <= p_paddle.rgt && p_ball.rgt > p_paddle.lft
     )
  { p_ball.lft = p_paddle.rgt;
    p_ball.vx = -p_ball.vx;
  }
}
```

Testen Sie Ihre Kollisionsfunktionen, indem Sie in der Konfigurationsdatei unterschiedliche Parameter eingeben: Startposition, Startgeschwindigkeit, Größe des Schlägers, evtl. auch unterschiedliche Geschwindigkeiten des Balls.

4.5 Spielsteuerung



Steuerungsmodule

Als nächstes kommt die Spielsteuerung an die Reihe. Die beiden Spieler sollen die der Lage sein, jeweils einen Schläger mittels Tastatur zu bewegen.

Zum Einsatz kommt hier das Observer-Pattern: Jedes Mal, wenn ein spezifischer Tastendruck erfolgt, schickt der Controller `PaddleControl` eine entsprechende Nachricht an alle Listener. Das Logik-Modul registriert sich als Listener für derartige Nachrichten, empfängt diese daher künftig und reagiert auf jede Nachricht, die es erhält, in geeigneter Form.

Sehen Sie sich zunächst einmal die die ursprüngliche Moorhuhn-Anwendung (app00) an. Dort gibt es

einen Controller, der bei einem Klick auf einen Vogel die Logik darüber informiert. Die Logik reagiert auf eine entsprechende Nachricht, indem sie den zugehörigen Vogel von der Bühne löscht (mittels der Remove-Funktion, die die Logik vom Game-Modul erhalten hat). Die Verknüpfung des Controllers mit den Vögeln erfolgt über das Game-Modul: In der Konfigurations-Datei `config00.json` gibt es das Attribut `view.hen.control`. Mit diesem Attribut wird der Moorhuhn-View die Klasse `controlBird` zugeordnet. Wie üblich wird diese Klasse im Game-Modul importiert und in der Hashmap `v_json_map` eingetragen.

Entfernen Sie diesen Controller zunächst vollständig (bevor Sie mit der Implementierung der Tastatur-Controller anfangen):

Entfernen Sie – sofern Sie dies noch nicht gemacht haben – das Objekt `view.hen` aus der Konfigurationsdatei `controll01.json` (und nicht etwa aus der JSON-Datei `config00.json`, die Sie zuvor analysiert haben)
Benennen Sie die Datei `control/controlBird.js` in `control/controlPaddle.js` um.
Benennen Sie die darin enthaltene Funktion ebenfalls um. Als Parameter wird dieser Controller ein Konfigurationsobjekt `p_config` erhalten. Den Rumpf der Funktion löschen Sie einfach.
Ändern Sie den Bezeichner `controlBird` in der Datei `game.js` geeignet ab.
Löschen sie in der Datei `logic/logic.js` den Inhalt des Rumpf der Funktion `init`.

Jetzt sollte die Anwendung wieder laufen, ohne dass sich hinsichtlich der Behandlung von Benutzereingaben irgendetwas geändert hätte. Sie haben ja in dieser Hinsicht auch noch nichts implementiert, sondern nur überflüssigen Ballast entfernt.

Dies folgt als nächster Schritt.

In der Anwendung `app00` wurden der Controller den View-Objekten der Moorhühner zugeordnet, da ein Klick auf ein Moorhuhn eine Reaktion des Spiel auslösen sollte. Diesmal soll der Controller die Tastatur überwachen. Da es sich für die Tastatur kein View-Objekt gibt, muss die Konfiguration diesmal etwas anders erfolgen. Das Game-Modul `game.js` wurde so implementiert, dass es neben der Erzeugung und Initialisierung von Model- und View-Objekten auch die Initialisierung von Control-Objekten übernehmen kann.

Fügen Sie in die Datei `config01.json` hinder dem Objekt `view` ein Objekt namens `control` ein. (Sie könnten es auch vor diesem Objekt oder sogar vor dem Objekt `model` einfügen. Die Reihenfolge der Attribute eine Objektes spielt keine Rolle – zumindest sollte sie keine Rolle spielen.) In das Objekt `control` fügen Sie zwei Objekte `paddle1` und `paddle2` zur Konfiguration der Steuerung der beiden Schläger ein.

Jedes dieser beiden Objekte enthält vier Attribute:

`control`: Hier muss der Name der zugehörigen Controller-Funktion angegeben werden. Im Fall der beiden Schläger ist dies die (bereits existierende aber noch nicht implementierte) Funktion `controlPaddle`.
`model`: Hier muss der Name des zu kontrollierenden Models angegeben werden. Im Model-Objekt weiter oben in der Konfigurationsdatei hatten Sie die Konfigurationen für zwei Schläger angelegt. Geben Sie im Controller jeweils den Namen eines dieser beiden Objekte an.
`up` und down: Hier muss jeweils ein Objekt angegeben werden, die eine Keyboard-Taste beschreibt. Beispielsweise beschreibt das Objekt

```
{"key": "ArrowUp",   "keyCode": 38}
```

die Pfeil-nach-oben-Taste. (Leider ist es derzeit noch notwendig sowohl den Namen einer Taste, als

auch den zugehörigen Key-Code anzugeben, da beispielsweise Android-Geräte mit (Bluetooth-)Tastatur die modernere Key-Variante noch nicht beherrschen.[1]). Allerdings ist die Verwendung von Key-Codes als *deprecated* (*angelehnt/überholt*) markiert worden, das die Codes browser- und tastaturabhängig sind (siehe z. B. MDN web docs – KeyboardEvent.keyCode[2]). Man sollte also bei einem Tastaturevent immer erst überprüfen, ober der Bezeichner der Taste den Vorgaben entspricht, bevor man auf den Key-Code zurückgreift. **Tipp:** Auf der Seite http://keycode.info/ können Sie eine beliebige Taste drücken und erhalten beides `keyCode` und `key`.

Sobald Sie die beiden Controller konfiguriert haben, sollten Sie Ihre Anwendung testen. Sie sollte fehlerfrei durchlaufen, aber auf Tastaturereignisse reagiert sie noch nicht. Es ist aber schon gut zu wissen, dass der Programmstart keinen Fehler zur Folge hat. Wenn sie das Modul `controlPaddle` nicht sauber ins Game-Modul integriert haben, erhalten Sie einen Fehler: `v_json_map[l_config.control] is not a function`. Probieren Sie es aus, indem Sie in der Datei `config01.json` einmal absichtlich einen Schreibfehler in das Wort `controlPaddle` einfügen.

Jetzt ist es an der Zeit, die Funktion `controlPaddle` (in der Datei `control/controlPaddle.js`) zu implementieren.

Schreiben Sie testhalber mal den Befehl

```
console.log(p_config);
```

in den Rumpf dieser Funktion. Im Konsolfenster des Browsers sollten beim Programmstart die beiden zuvor erstellten Konfigurationsobjekte ausgegeben werden.

Wie man ein Tastaturereignis abfängt (mit `window.addEventListener()...`) und behandelt (mittels Callback-Funktion) sollte Ihnen bekannt sein: HTML5-Tutorium: JavaScript: Hello World 03, insbesondere Datei `main.js`.

Fügen Sie **in** den Rumpf der Funktion `controlPaddle` zwei Funktionen `o_start_moving` und `o_stop_moving` (Präfix `o_` da es sich um Observer-Funktionen handelt; observer = event listener). jeweils mit Parameter `p_event` ein. Sorgen Sie dann mittels geeigneter `window.addEventistener`-Anweisungen dafür, dass die erste Funktion bei einem `keyDown`-Event ausgeführt wird und die zweite bei einem `keyUp`-Event. Bei jedem Aufruf soll die zugehörige Observer-Funktion ihren eigenen Namen sowie den Key-Bezeichner und den Key-Code der Taste ausgeben, die den Event veranlasst hat.

Wenn alles wunschgemäß funktioniert, sollten Sie jetzt abhängig von dem jeweiligen Tastendruck eine geeignete Nachricht per Event-Dispatcher verschicken. Löschen sie in der Datei `control/control.js` die überflüssige „Konstante“ `control.C_EVENT_BIRD_HIT` und fügen Sie dafür für die drei möglichen Schläger-Ereignisse drei neue Konstanten ein:

```
control.C_EVENT_PADDLE_UP   = 'EventPaddleUp';
control.C_EVENT_PADDLE_DOWN = 'EventPaddleDown';
control.C_EVENT_PADDLE_STOP = 'EventPaddleStop';
```

Gehen Sie zurück in die Datei `control/controlPaddle.js`. Dort sehen Sie, dass das Event-Dispatcher-Objekt `control`, für das Sie gerade ein paar Konstanten definiert haben, bereits importiert wird. Sie können dieses Objekt verwenden, um Nachrichten an andere interessierte Objekte zu verschicken.

Fügen Sie an den Anfang des Rumpfs der Funktion `controlPaddle` ein paar Konstantendefinitionen

ein, die die Attribute des Konfigurationsobjektes speichern:

```
const
  c_model_name   = p_config.model,
  c_key_up       = p_config.up.key,
  c_keycode_up   = p_config.up.keyCode,

  c_key_down     = p_config.down.key,
  c_keycode_down = p_config.down.keyCode;
```

Definieren Sie im Anschluss an die Konstanten eine Hilfsfunktion zum Dispatchen eines neuen Events:

```
function f_dispatch(p_type)
{ control.dispatchEvent(new CustomEvent(p_type, {'detail':
c_model_name})); }
```

Diese Funktion erwartet den Typ des Ereignisses, das verschickt werden soll, im Eingabe-Parameter `p_type`. Prinzipiell kann jeder beliebige String verwendet werden, Sie sollten sich aber (zunächst) auf die drei zuvor definiert „Konstanten“ `control.C_EVENT_PADDLE_UP`, `control.C_EVENT_PADDLE_DOWN` und `control.C_EVENT_PADDLE_STOP` beschränken.

Die Nachricht, d. h. das Event-Objekt, das verschickt wird, wird mittels `new CustomEvent(p_type, p_init)` erstellt (siehe MDN web docs – CustomEvent und auch MDN web docs – Creating and triggering events[3]).

Als Detail-Information, auf wen sich das Ereignis bezieht, wird der im Konfigurationsobjekt gespeicherte Name `p_config.model` (bzw. `c_model_name`) an den Empfänger der Nachricht übermittelt.

Implementieren Sie die Funktion `o_start_moving` so, dass das Ereignis `control.C_EVENT_PADDLE_UP` (mittels der zuvor definierten Hilfsfunktion) verschickt wird, wenn die im Konfigurationsobjekt angegebene „Up“-Taste betätigt wurde, und das Ereignis code>control.C_EVENT_PADDLE_DOWN, falls die „Down“-Taste betätigt wurde. Die Funktion `o_stop_moving` soll das Ereignis `control.C_EVENT_PADDLE_STOP` versenden, sobald eine der genannten Tasten losgelassen wird.

Tipp: Wenn Sie vor der Funktion `controlPaddle` den folgenden Kommentar einfügen,

```
/**
 * @param p_config {{model: String,
 *                   up:    {key: String, keyCode: Number},
 *                   down:  {key: String, keyCode: Number},
 *                  }
 *                 }
 */
```

zeigt Ihnen WebStorm weniger Warnings an. Die Typdefinition, die in geschweiften Klammern hinter dem Parameter `p_config` angegeben wird, beschreibt, wie das Konfigurationsobjekt aufgebaut sein

sollte. Tatsächlich enthält das Objekt, das übergeben wird noch ein weiteres Attribut (nämlich `"control": "controlPaddle"`). Das wird aber vom Game-Modul benötigt, um die korrekte Control-Funktion aufzurufen, die Control-Funktion selbst benötigt dieses Attribut nicht mehr.

Wie können Sie nun Ihren Code testen? Ganz einfach, indem Sie zugehörige Event-Listener schreiben. Wo dieser definiert wird, ist dem Event-Dispatcher vollkommen egal. Daher schreiben Sie die die Tests einfach gleich in das Modul `controlPaddle` selbst hinter die Funktion `controlPaddle`.

```
control.addEventListener
( control.C_EVENT_PADDLE_DOWN,
  function(p_event)
  { console.log('down', p_event.detail); }
);
```

(Analog für die beiden anderen Event-Typen.)

4.5.1 Behandlung von Tastaturevents durch die Logik

Wenn alles klappt, d. h., wenn Sie im Konsolfenster des Browsers korrekt benachrichtigt werden, sobald Sie der in der Konfigurationsdatei spezifizierten Tasten drücken, sollten Sie die drei Testbefehle ans Ende des Rumpfes der `init`-Funktion in der Datei `logic/logic.js` einfügen.

Da das Logikmodul ebenfalls das Objekt `control` importiert, sollte die Anwendung immer noch funktionieren. Das heißt, im Konsolfenster des Browsers sollten weiterhin die Tastenereignisse protokolliert werden.

Im Logikmodul haben Sie Zugriff auf die Modelle, die vom Game-Modul erzeugt wurden: `p_models`. Im Parameter `p_event` der drei Eventlistener erhalten Sie **den Namen** des Objektes, das mittels des aktuellen Tastaturevents gesteuert werden soll: `p_event.detail`. Damit ist es möglich, das Schläger-Model-Objekt selbst zu ermitteln: `p_models[p_event.detail]`. Für diese Objekte hatten Sie zuvor drei Methoden definiert: `up`, `down` und `stop`.

Rufen Sie in jedem der drei Eventhandler die jeweils passende Methode auf. Nun sollten die beiden Paddles gesteuert werden können. (Falls alles funktioniert, sollten Sie die `console.log`-Befehle in den Eventhandler auskommentieren oder löschen.)

4.5.2 Behandlung von Ball-Events durch die Logik

Jetzt fehlt noch die Reaktion der Logik auf das Verschwinden des Balls.

Fügen Sie in die Datei `control/control.js` zwei weitere „Konstanten“ ein:

```
control.C_EVENT_BALL_LEAVES_LEFT  = 'EventBallLeavesLeft';
control.C_EVENT_BALL_LEAVES_RIGHT = 'EventBallLeavesRight';
```

Sorgen Sie dann dafür, dass das Modul `collisionStageBall` das jeweils passende Ereignis meldet, wenn der Ball die Bühne auf der linken Seite verlässt (der rechte Rand des Balls ist kleiner als der linke Rand der Bühne) oder auf der rechten (der linke Rand des Balls ist größer als der rechte Rand

der Bühne). Detailinformationen brauchen Sie in dem Event-Objekt nicht zu speichern, das es ja nur einen Ball sowie einen linken und einen rechten Bühnenrand gibt. Die Logik erkennt schon am Ereignistyp, was passiert ist. **Achtung**: Vergessen sie nicht, das Modul `control.js` zu importieren.

Zum Testen der Events können Sie wie schon im Fall der Tastatur-Ereignisse zwei Event-Listener direkt im Modul `control.js` definieren und geeignete `console.log`-Befehle einfügen. Sobald dies funktioniert verschieben Sie die beiden Befehle wieder in den Rumpf der Init-Funktion des Logik-Moduls.

Fügen Sie **in** den Rumpf der Logikfunktion folgende Hilfsfunktion ein (da diese Funktion **innerhalb** des Rumpfes von `init` auf `p_remove` und `p_add` zugreifen kann):

```
function f_new_ball()
{ p_remove(p_models.ball);
  p_add(concretize(c_init_ball), 'ball');
}
```

Vergessen Sie nicht die Funktion `concretize` aus dem Modul `wk/Util` zu importieren.

Diese Funktion löscht das alte Ball-Objekt mittels der Remove-Funktion, die vom Game-Modul bereit gestellt wird. Und es erzeugt ein neues Objekt mit demselben Model-Namen `ball`, den das gelöschte Ball-Objekt hatte. Wie üblich muss bei der Erzeugung eine Spiel-Model-Objektes ein Konfigurationsobjekt übergeben werden. Fügen Sie eine derartiges Objekt zunächst direkt (im Anschluss an die Import-Befehle) in das Logik-Modul ein:

```
const
  c_init_ball
    = { "view": "ball", "class": "MovableCircle", "isMovable": true,
        "r":     15,
        "x":    300,
        "y":    200,
        "vx":   { "@min": 150, "@max": 300, "@positive": 0.5 },
        "vy":   { "@min": 150, "@max": 300, "@positive": 0.5 }
      };
```

Nun können Sie jeweils einen Aufruf dieser Hilfsmethode in die beiden Event-Listener für die Ereignisse `C_EVENT_BALL_LEAVES_LEFT` und `C_EVENT_BALL_LEAVES_RIGHT` einfügen.

Ab sofort sollte sofort, wenn der Ball die Bühne verlässt, ein neuer Ball in der Mitte der Bühne erzeugt werden und sich in zufällige Richtung los bewegen. (Wenn Sie Ihre Bühne nicht 600 mal 400 Pixel groß gewählt haben, müssen Sie die x- und die y-Position des Balls anpassen, damit er von der Mitte der Bühne aus startet.)

Anmerkung: Man könnte auch noch ein Timer-Objekt (WindowTimers.setTimeout()) erstellen, das das neue Ball-Objekt erst nach einer gewissen Zeitspanne erzeugt (z. B. 500 Millisekunden). Dazu müsste allerdings das Logik-Objekt auf die Game-Loop zugreifen können, um das Spiel anzuhalten., während kein Ball im Spiel ist. Das ist etwas für Pong02.

Eine Unsauberkeit sollten Sie noch beheben. Die Definition der Konstanten `c_init_ball` im Modul `logic` widerspricht dem Prinzip „Schreibe konfigurierbaren Code“. Wenn Sie beispielsweise in der

Konfigurationsdatei `config01.json` die Grüße der Bühne ändern, müssen Sie zurzeit zusätzlich in der Datei `logic/logic.js` die Position des Balls im Objekt `c_init_ball` ändern, damit er weiterhin von der Mitte aus startet. Die Erfahrung zeigt, dass das regelmäßig vergessen wird. Außerdem findet man, wenn die Programmierung schon etwas zurück liegt, die Datei, in der die Änderung erfolgen muss, i. Allg. nur schwer.

Das Objekt, das derzeit in der Konstanten `c_init_ball` gespeichert wird, sollte unbedingt in der Datei `config01.json` gespeichert werden und nicht in der Datei `logic.js`. Dabei gibt es allerdings zwie Schwierigkeiten:

1. Das Ball-Objekt soll nicht zum Startzeitpunkt der Anwendung erstellt werden, sondern vom Logik-Modul, wenn ein neues Spiel gestartet wird. Daher kann das Initialisierungsobjekt nicht im Objekt `model` innerhalb der Konfigurationsdatei werden. Alle Model-Objekte, deren Konfiurationsinformationen dort abgelegt sind, werden bereits zum Startzeitpunkt erstellt.
2. Jedesmal, wenn ein neues Ball-Objekt vom Logik-Modul erzeugt wird, soll es eine zufällige Geschwindigkeit haben. Das erfolgt mit Spezifikationen der Art `"vx": {"@min": 150, "@max": 300, "@positive": 0.5}`. Die Funktion `concretize` aus dem Modul `wk/Util` ersetzt das Objekt `{"@min": 150, "@max": 300, "@positive": 0.5}` durch einen Wert zwischen `150` und `300` und negiert diesen Wert in 50% der Fälle. Allerdings wird die Funktion `concretize` schon bei Programmstart auf die gesamte Konfigurationsdatei angewendet, so dass ein Element des Konfigurationsobjekts, das irgendeinem Modul übergeben wird, nie ein derartiges Konstrukt enthält. Es enthält stattdessen einen zufällig gewählten Wert aus dem angegebenen Bereich, der sich aber nicht mehr ändert.

   Beide Probleme können mit der aktuellen Implementierungen des Game-Moduls und der Funktion `concretize`.

   ad 1.) Objekte, die im Konfigurationsobjekt `init` eingefügt werden, werden dem Logik-Modul im Parameter `p_init` zur Verfügung gestellt (allerdings in konkretisierter Form). ad 2.) Die Funktion `concretize` ersetzt einen doppelten Klammeraffen durch einen einfachen Klammeraffen. Das heißt, die Konkretisierung von

   ```
   {"@min": 150, "@max": 300, "@positive": 0.5}
   ```

   liefert eine Zahl zwischen 150 und 300 oder zwischen -300 und -150 als Ergebnis, wohingegen die Konkretisierung von

   ```
   {"@@min": 150, "@@max": 300, "@@positive": 0.5}
   ```

   das Objekt `{"@min": 150, "@max": 300, "@positive": 0.5}` als Ergebnis liefert.

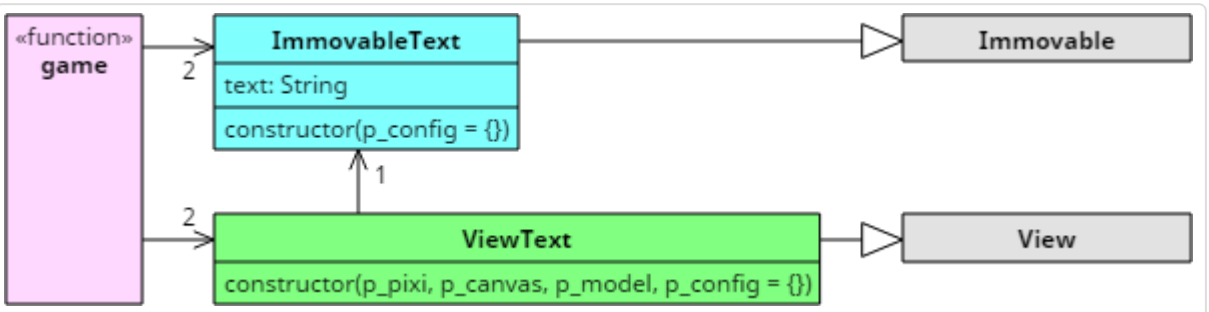
   Damit lässt sich das oben beschriebene Problem beheben. Fügen Sie in die Datei `config02` folgendes Objekt vor dem Objekt `model` ein:

```
"init":
{
  "ball":
  {
    "view": "ball", "class": "MovableCircle", "isMovable": true,
    "r":      15,
    "x":     300,
    "y":     200,
    "vx":    { "@@min": 150, "@@max": 300, "@@positive": 0.5 },
    "vy":    { "@@min": 150, "@@max": 300, "@@positive": 0.5 }
  }
},
```

Fügen Sie in die Funktion `init` des Moduls `logic` dem Befehl `console.log(p_init);` und sehen Sie sich das Ergebnis an.

Nun können sie die Konstante `c_init_ball` aus dem Logik-Modul löschen und in der Funktion `f_new_ball` den Aufruf `concretize(c_init_ball)` durch `concretize(p_init.ball)` ersetzen.

4.6 Textausgabe



Textmodule

Jetzt fehlt nur noch die Ausgabe des Punktestands. Für die Verwaltung des Punktestands brauchen Sie kein Punkte-Model. Es reichen zwei Variablen `v_score_player1` und `v_score_player2`, die jedes Mal, wenn der Ball die Bühne verlässt entsprechende der Seite, auf der der Ball die Bühne verlassen hat, um Eins erhöht wird.

Für die Ausgabe des Textes benötigen Sie die beiden Module `model/ImmovableText.js` und `view/ViewText.js`. Im ersten Modul die Klasse `ImmovableText` definiert, die alle Properties der Klasse `Immovable` erbt und das Attribut `text` zur Propertyliste hinzufügt. Da wird mit dem Befehl `super(mixin ({text: }, p_config));` im Konstruktor erledigt. (Zur Erinnerung: Den Befehl mixin finden Sie im Modul wk/util.)

Die Klasse `ViewText` wird analog zur Klasse `ViewGraphics` (`wk/view/ViewGraphics`) erstellt. Nur wird diesmal kein Grafik-Objekt erstellt (`new p_pixi.Graphics()`), sondern ein Text-Objekt (`new p_pixi.Text(p_model.text, p_config)`), dem der aktuelle Text aus dem Model-Objekt sowie das Konfigurationsobjekt mit beliebigen Style-Attributen als Argumente übergeben wird.

Wichtig ist noch, dass Sie eine Update-Funktion einfügen, damit die View stets den aktuell im Modell gespeicherten Text anzeigt:

```
update()
{ super.update();
  this.sprite.text = this.model.text;
}
```

Wie bereits ziemlich zu Beginn des Dokument erwähnt wurde, wird in Pong02 die Update-Funktion durch einen Event-Handling-Mechanismus ersetzt: Jedes Mal, wenn sich im Model etwas ändert, werden alle zugehörigen View-Objekte (es kann durchaus mehr als ein View-Objekt für ein Model-Objekt geben) über die Änderung informiert. In diesem Moment aktualisieren die View die zugehörige Darstellung. Damit ist die Update-Methode im View-Objekt überflüssig. Diese aktualisiert derzeit **jedes Mal**, bevor ein View-Objekt auf die Bühne gezeichnet wird, den darzustellenden Inhalt, auch wenn sich das zugehörige Model gar nicht verändert hat und somit an der Darstellung gar nichts geändert werden muss. Der Event-Handling-Mechanismus sorgt dafür, dass View-Updates nur dann durchgeführt werden, wenn sich im zugehörigen Model auch etwas geändert hat.

Nun ist es an der Zeit, die beiden View-Objekte in die Anwendung einzubinden. Dabei gehen Sie wie üblich vor:

Importieren der Module code>ImmovableText und `ViewText` in das Game-Modul. Außerdem muss die JSON-Map muss geeignet werden.
Konfiguration zweier Text-Objekte samt zugehöriger View (mit PIXI.TextStyle-Attributen!).
Positionieren Sie das eine Text-Objekt in der linken Spielfeldhälfte und das andere in der rechten. Denken Sie daran, das Anchoring von Texten funktioniert noch nicht.
Ändern des Textattribut-Wertes im Model-Objekt, wann immer ein Spieler einen Punkt erhält.

4.7 Abschlussarbeiten

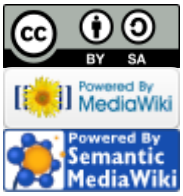
Haben Sie eigentlich jedes Mal daran gedacht, in Dateien, die Sie erstellt haben, Ihren Namen ins Kommentarfeld `@author` zu schreiben?

Es fehlt auch noch die Implementierung des Use Case: „Spielende, sobald einer der Spieler 10 Punkte erreicht hat“. Auch dafür muss das Logik-Modul auf die Game-Loop zugreifen können. Daher ist auch dies ein Punkt, der erst in Pong02 behandelt wird.

5 Quellen

1. MDN web docs – KeyboardEvent.key
2. MDN web docs – KeyboardEvent.keyCode
3. MDN web docs – Creating and triggering events

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


BY SA
Powered By
MediaWiki
Powered By
Semantic
MediaWiki