HTML5-Tutorium: Canvas: MiniPong 03

Wechseln zu:Navigation, Suche

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

Korrektheit: 3 (zu größeren Teilen überprüft)	**Umfang**: 3 (einige wichtige Fakten fehlen)	**Quellenangaben**: 5 (vollständig vorhanden)	**Quellenarten**: 5 (ausgezeichnet)	**Konformität**: 5 (ausgezeichnet)

HTML-Tutorium: MiniPong

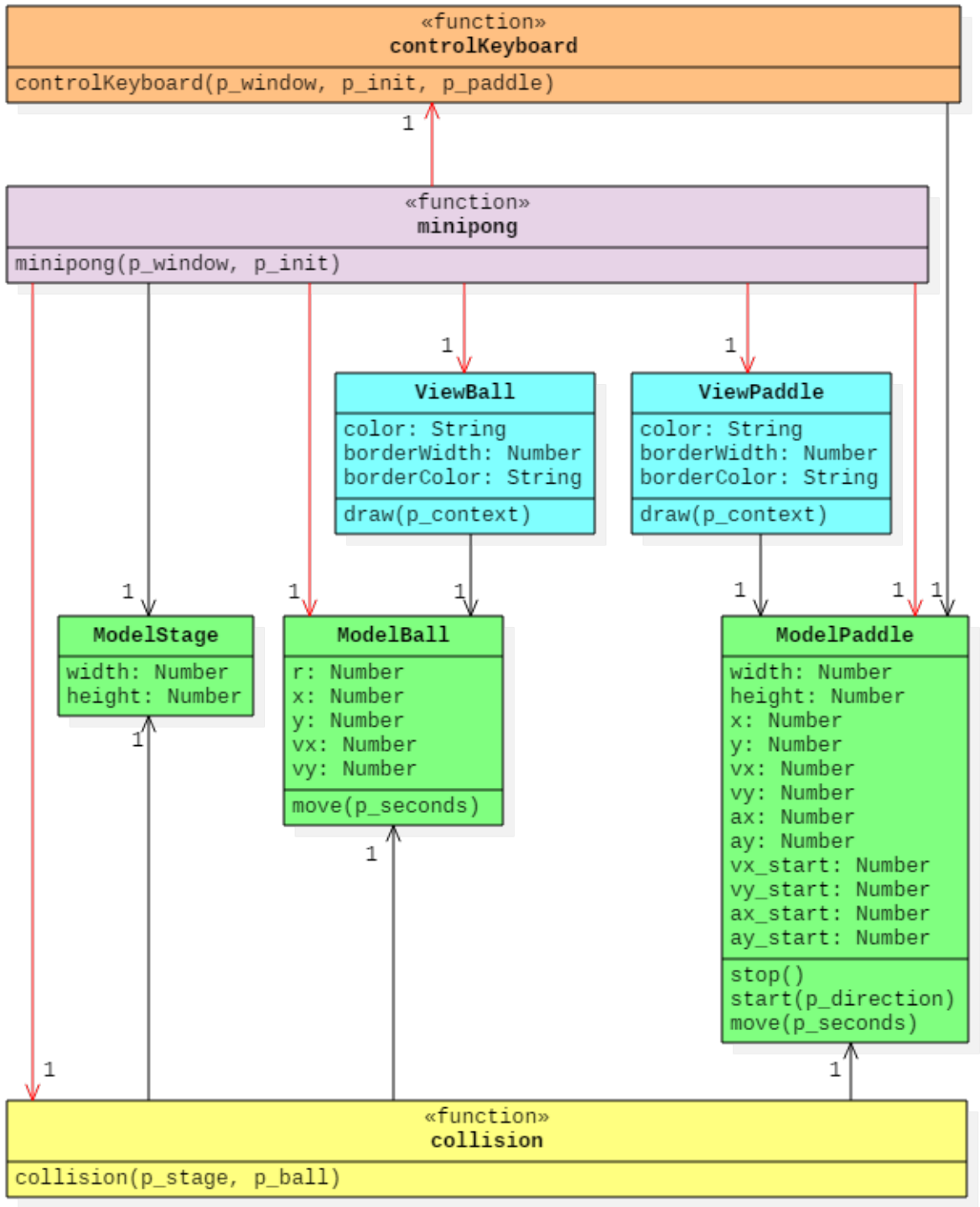
MiniPong: | Teil 1 | Teil 2 | Teil 3 | Teil 4 | Teil 5

Musterlösung: `index.html`, `index_penetration.html` (WK_MiniPong03 (SVN))
Im Penetrations-Beispiel verschwindet der Schläger bei andauernder Kollision mit der and langsam in dieser.

Ergänzungsaufgabe: `index_h.html`, `index_v.html` (WK_MiniPong03a (SVN))
Die Steuerung erfolgt mit den Pfeiltasten. Das zweite Paddle wird mit den Tasten „`w`“ und „`s`“ gesteuert.

Inhaltsverzeichnis

1 Ziel: Interaktion mit dem Schläger



Klassendiagramm

Im dritten Teil des Tutoriums wird beschrieben, wie man die Interaktion mit dem Schläger realisiert.

Das Klassenmodell im zweiten Teil des Tutoriums entwickelten Klassendiagramm wird um zwei Klassen erweitert: „`ModelPaddle`“ und „`ViewPaddle`“. Diese beiden Klassen haben im Prinzip dieselben Aufgaben wie die Klassen „`ModelBall`“ und „`ViewBall`“: Sie sollen die physikalischen Eigenschaften eines Schlägers simulieren (Model) und den Schläger auf der Bühne visualisieren (View). Auch hier kennt die View das Model, damit sie den Schläger an der richtigen Position in der richtigen Größe zeichnen kann. Umgekehrt hat das Model keinen Zugriff auf die View. Wozu auch?

Die Kollisionserkennung und -behandlung muss erweitert werden:

Kollisionsbehandlung zwischen Bühnenwand und Ball (existiert bereits)
Kollisionsbehandlung zwischen Bühnenwand und Schläger (dieser Teil des Tutoriums)
Kollisionsbehandlung zwischen Ball und Schläger (nächster Teil des Tutoriums)

Es fällt auf, dass das Model des Schlägers viel mehr Attribute enthält, als das Model des Balls. Das hängt damit zusammen, dass es zwei weitere Methoden gibt: „`start`“ und „`stop`“. Die Start-Methode wird aufgerufen, wenn der Spieler den Schläger mittels Tastatursteuerung (oder Ähnlichen) in Bewegung setzt. Er kann dabei die Bewegungsrichtung des Schlägers festlegen (nach links oder rechts bzw. nach oben oder unten). Die Stopp-Methode wird aufgerufen, wenn der der Spieler den entsprechenden Befehl dazu gibt oder wenn der Schläger mit der Wand kollidiert. Aus diesem Grund wird nicht nur die aktuelle Geschwindigkeit des des Schlägers („`vx`“ und „`vy`“) sondern auch die Startgeschwindigkeit („`vx_start`“ und „`vy_start`“) gespeichert. Die aktuelle Geschwindigkeit des Schlägers wird jeweils auf die Startgeschwindigkeit gesetzt, sobald er mittels `start` in Bewegung gesetzt wird.

Außerdem werden für den Schläger Beschleunigungsinformationen gespeichert. Damit wird erreicht, dass der Schläger immer schneller wird, solange er vom Spieler oder der Wand nicht gestoppt wird. (Man können ihn auch so konfigurieren, dass er immer langsamer wird.) Auch hier wird zwischen aktueller Beschleunigung („`ax`“ und „`ay`“) und Startbeschleunigung („`ax_start`“ und „`ay_start`“) unterschieden. Solange der Schläger sich nicht bewegt, ist die aktuelle Beschleunigung gleich 0. Wird er in Bewegung gesetzt, wird die Beschleunigung auf die Startwerte gesetzt.

Im vierten Teil des Tutoriums werden sowohl für den für den Ball als auch für den Schläger weitere Methoden und Attribute definiert.

Zu guter Letzt ist es notwendig die Benutzerinteraktion mit dem Schläger zu ermöglichen. Das ist die Aufgabe des Moduls „`keyboardController.js`“. In diesem Modul wird eine Funktion mit dem Namen „`keyboardController`“ definiert, welche je nach Benutzernutzer die Start- oder die Stoppmethode des Paddles aufruft.

2 Neues Projekt anlegen

Legen Sie ein neues Projekt mit dem Namen „`MiniPong03`“ an.

Kopieren Sie die Dateien der App3 des Projektes MiniPong02 (App3) in das neue Projekt.

Nehmen Sie folgende Anpassungen vor:

Umbenennung des Ordners „`app3`“ in „`app`“.
Umbenennung der Datei „`main3.js`“ in „`main.js`“.
Umbenennung der Datei „`index3.html`“ in „`index.html`“.
In der Datei „`main.js`“:
Ersetzen von „`app: 'app3'`“ durch „`app: 'app'`“.
In der Datei „`index.html`“:
Ändern des Titels in „`MiniPong03`“:
Ersetzen von „`js/main3`“ durch „`js/main`“ im Script-Befehl.

Nun sollte die Anwendung wieder laufen.

2.1 Initialisierung

Als nächstes müssen die Schlägereigenschaften in die Initialisierungs-Datei „`init.json`“ eingefügt werden.

In das Objekt „`model`“ werden folgende Daten eingefügt:

```
"paddle":
{
  "width":    50,
  "height":    8,
  "pos":     {"x": 175, "y": 287},
  "vel":     {"x": 100, "y": 0},
  "acc":     {"x": 500, "y": 0}
}
```

Der Schläger wird durch ein Rechteck dargestellt. Dieses Rechteck ist 50 Pixel breit und 8 Pixel hoch. Es wird mittig in der Nähe des unteren Randes der Zeichenbühne platziert:

$($Breite des Canvas $-$ Breite des Schlägers$)/2 = (400-50)/2 = 175$
$($Höhe des Canvas $-$ Höhe des Schlägers$\,-\,5) = 300- 8 - 5 = 287$

Wenn der Benutzer durch Tastendruck den Schläger in Bewegung setzt, bewegt er sich zunächst mit eine Geschwindigkeit von 100 Pixeln pro Sekunde entlang der x-Achse. Je länger der Benutzer die Taste drückt, desto schneller wird der Schläger. Er wird mit 500 Pixeln pro Sekunde beschleunigt.

In das Objekt „`view`“ werden folgende Daten eingefügt:

```
"paddle":
{
  "color":       "#aa55cc",
  "borderWidth": 1,
  "borderColor": "#000000"
}
```

Die View-Informationen unterscheiden sich nicht sonderlich von den Ball-View-Daten, außer, dass der Schläger eine andere Farbe haben soll.

Als letztes fügen wir hinter das View-Objekt noch ein neues Objekt ein:

```
"control":
{
  "player":
  {
    "left":  {"key": "ArrowLeft",  "keyCode": 37},
    "right": {"key": "ArrowRight", "keyCode": 39}
  }
}
```

Das Control-Objekt enthält Informationen zu den Steuermöglichkeiten, die der Spieler hat. Hier werden die Key-Codes der Tasten angegeben, mit denen der Benutzer den Schläger nach links bzw. rechts bewegen kann.

2.2 Modell des Schlägers

Das Model des Schlägers ist ähnlich aufgebaut, wie das Modell des Balls. Da das Paddle durch ein Rechteck dargestellt wird, gibt es natürlich keinen Radius, sondern eine Breite und eine Höhe, die gespeichert werden müssen. Außerdem werden nicht nur Positions- und Geschwindigkeitsangaben gespeichert, sondern auch Beschleunigungsinformationen gespeichert. Ein weiterer wesentlicher Unterschied ist, dass der Konstruktor die Geschwindigkeits- und Beschleunigungsdaten nicht in den Attributen „`vx`“, „`vy`“, „`ax`“ und „`ay`“ speichert, sondern in den Attributen „`vx_start`“, „`vy_start`“, „`ax_start`“ und „`ay_start`“.

Der Grund ist, dass der Schläger sich zunächst gar nicht bewegt. Zu Beginn sind die Geschwindigkeit und die Beschleunigung also gleich . Erst, wenn der Benutzer den Schläger in Bewegung setzt, werden die Startwerte für Geschwindigkeit und Beschleunigung in die eigentlichen Geschwindigkeits- und Beschleunigungsattribute kopiert. Und wenn der Benutzer den Schläger stoppt, werden Geschwindigkeit und Beschleunigung wieder auf Null gesetzt.

Der Konstruktor sieht daher folgendermaßen aus:

```
function ModelPaddle(p_init)
{
  this.width    = p_init.width;
  this.height   = p_init.height;

  this.x        = p_init.pos.x;
  this.y        = p_init.pos.y;

  this.vx_start = p_init.vel.x;
  this.vy_start = p_init.vel.y;

  this.ax_start = p_init.acc.x;
  this.ay_start = p_init.acc.y;

  this.stop(); // By default, the paddle does not move.
}
```

Um den Schläger starten und stoppen zu können, werden die beiden Methoden „`start`“ und .„`stop`“ definiert. Für den Ball ist dies nicht notwendig, da sich dieser selbstständig bewegt und nicht durch Benutzeraktionen gesteuert wird.

Die Methode „`stop`“ setzte die Geschwindigkeits- und Beschleunigungsattribute des Schlägers auf . Sie insbesondere auch vom Konstruktor aufgerufen, um die Attribute „`vx`“, „`vy`“, „`ax`“ und „`ay`“ zu initialisieren. Zum Zeitpunkt der Erzeugung bewegt sich der Schläger noch nicht, also ist der Wert dieser Attribute jeweils Null.

```
ModelPaddle.prototype.stop =
 function()
 { this.vx = 0;
   this.vy = 0;

   this.ax = 0;
   this.ay = 0;
 };
```

Die Start-Methode hat die Aufgabe, die Geschwindigkeit- und die Beschleunigungsattribute des Schlägers auf die Anfangswerte zu setzen, sobald sie aufgerufen wird. Allerdings ist dies nicht so einfach, wie im Falle der Stopp-Methode.

FOLGENDES FUNKTIONIERT DAHER NICHT:

```
ModelPaddle.prototype.start =
 function()
 { this.vx = this.vx_start;
   this.vy = this.vy_start;

   this.ax = this.ax_start;
   this.ay = this.ay_start;
 };
```

Zum einen soll der Schläger, wenn er einmal gestartet wurde, konstant schneller werden. Allerdings wird die Start-Methode üblicherweise öfters als einmal aufgerufen. Wenn der Benutzer beispielsweise den Schläger mit Hilfe von Pfeiltasten steuert, hält der die entsprechende Taste längere Zeit gedrückt. Nach kurzer Zeit setzt die Tastenwiederholungsfunktion des Betriebssystems ein und der Startbefehl wird regelmäßig erneut abgesetzt. Und jeder erneute Aufruf des Startbefehls hätte zur Folge, dass die Geschwindigkeit des Schlägers zurück auf die Startgeschwindigkeit gesetzt wird.

Um dies zu verhindern, muss überprüft werden, ob der Schläger gerade in Bewegung ist oder nicht. Nur im letzteren Fall darf der Schläger mit Hilfe der Start-Werte in Bewegung gesetzt werden.

Es gibt noch einen zweiten Punkt, der berücksichtigt werden muss: Der Benutzer kann bei einem horizontalen Schläger wählen, ob der Schläger nach links oder nach rechts bewegt werden soll. Bei einem vertikalen Schläger hat er die Wahl zwischen rauf und runter.

Die Start-Methode, die nur die horizontalen Richtungsänderungen unterstützt, sieht folgendermaßen aus:

```
ModelPaddle.prototype.start =
  function(p_direction)
  {
    // react only if the paddle is not already moving
    if (this.vx === 0 && this.vy === 0)
    {
      switch (p_direction)
      {
        case "left":
          this.vx = -this.vx_start;
          this.ax = -this.ax_start;
          break;
        case "right":
          this.vx =  this.vx_start;
          this.ax =  this.ax_start;
          break;

        case "up":
          this.vy = -this.vy_start;
          this.ay = -this.ay_start;
          break;
        case "down":
          this.vy =  this.vy_start;
          this.ay =  this.ay_start;
          break;
      }
    }
  };
```

Wenn sich der Schläger, aus welchen Gründen auch immer, bereits bewegt, macht sie gar nichts, nur wenn die Geschwindigkeit sowohl in x- als auch in y-Richtung gleich Null sind, startet sie den Schläger. Abhängig von der vom Benutzer vorgegebenen Bewegungsrichtung (`p_direction`) werden die Geschwindigkeits- und Beschleunigungs-Start-Werte in die entsprechenden Attribute kopiert. Beachten Sie, dass bei den Richtungen „`left`“ und „`up`“ sich jeweils die Vorzeichen gegenüber den Richtungen „`right`“ und „`down`“ ändern.

Jetzt fehlt im Modell nur noch die Methode „`move`“. Diese gibt es im Gegensatz zu den vorherigen beiden Methoden auch für Ball-Objekte.

Allerdings ist folgende Methode etwas allgemeiner:

```
ModelPaddle.prototype.move =
  function(p_seconds)
  {
    this.x += this.vx * p_seconds;
    this.y += this.vy * p_seconds;

    this.vx += this.ax * p_seconds;
    this.vy += this.ay * p_seconds;
  };
```

Hier wird bei jedem Aufruf nicht nur die Position verändert, sondern auch die Geschwindigkeit. Die Positionsänderung erfolgt anhängig von der aktuellen Geschwindigkeit und die Geschwindigkeitsänderung erfolgt abhängig von der aktuellen Beschleunigung. Wenn in der Initialisierungsdatei „`init.json`“ ein konstanter Beschleunigungswert vorgegeben ist, wird dadurch erreicht, dass der Schläger immer schneller wird, sobald er einmal gestartet wurde.

2.3 View des Schlägers

Der View Konstruktor und die zugehörige Draw-Methode unterscheiden sich nur in zwei wesentlichen Punkten:

1. Anstelle eines Kreises (`l_context.arc`) muss ein Rechteck gezeichnet (`l_context.rect`) gezeichnet werden.
2. Der Aufhängepunkt eines Rechtecks ist im Canvas-2D-Kontext die linke obere Ecke des Rechtecks ohne Rand. Der Aufhängepunkt eines Kreise ist dagegen dessen Mittelpunkt. Das muss bei Platzieren des Mini-Canvas, der die visuelle Darstellung des Paddels enthält, auf die Bühne berücksichtigt werden. Das heißt, die Draw-Methoden der beiden Objekte unterscheiden sich in dieser Hinsicht.

Konstruktor

```
function ViewPaddle(p_model, p_init, p_document)
{
  this.model       = p_model;

  this.color       = p_init.color;
  this.borderWidth = p_init.borderWidth;
  this.borderColor = p_init.borderColor;

  // Define a local canvas containing the view of the paddle.
  var l_canvas  = this.v_canvas = p_document.createElement("canvas"),
      l_context = l_canvas.getContext("2d");

  l_canvas.width  = p_model.width  + 2*this.borderWidth + 2;
  l_canvas.height = p_model.height + 2*this.borderWidth + 2;

  l_context.beginPath();
  l_context.rect(this.borderWidth, this.borderWidth,
                 p_model.width,    p_model.height
                );
  l_context.fillStyle = this.color;
  l_context.fill();
  if (this.borderWidth > 0)
  {
    l_context.lineWidth   = this.borderWidth;
    l_context.strokeStyle = this.borderColor;
    l_context.stroke();
  }
}
```

Draw-Methode

```
ViewPaddle.prototype.draw =
  function(p_context)
  {
    p_context.drawImage(this.v_canvas,
                        this.model.x - this.borderWidth,
                        this.model.y - this.borderWidth
                       );
  };
```

2.4 Das Spiel

Um Ihre View zu testen, müssen Sie ein Paddle, genauer gesagt: ein Paddle-Model und eine Paddle-View, in Ihr MiniPong-Spiel (`MiniPong.js`) einfügen. Das Model-Objekt müssen Sie ebenso wie das Ball-Objekt regelmäßig aktualisieren und das neue View-Objekt müssen Sie regelmäßig auf die Bühne zeichnen.

Zunächst einmal müssen Sie die Model- und die View-Klasse des Paddles als Module laden. Der Modulkopf der Datei „`minipong.js`“ muss entsprechend erweitert werden:

```
define
( ['app/model/ball', 'app/view/ball',
   'app/model/paddle', 'app/view/paddle',
   'app/model/collision'
 ],
 function(ModelBall, ViewBall,
          ModelPaddle, ViewPaddle,
          collision
         )  ...
```

Die eigentliche Funktion sieht mit den zuvor beschriebenen Erweiterungen folgendermaßen aus:

```
function minipong(p_window, p_init)
{
  var l_document     = p_window.document,
      l_canvas_init  = p_init.canvas, // contains width and height of the
canvas
      l_canvas       = l_document.getElementById("canvas"),
      l_context      = l_canvas.getContext("2d"),

      l_model_ball   = new ModelBall(p_init.model.ball),
      l_view_ball    = new ViewBall(l_model_ball, p_init.view.ball,
l_document),

      l_model_paddle = new ModelPaddle(p_init.model.paddle),
      l_view_paddle  = new ViewPaddle(l_model_paddle, p_init.view.paddle,
l_document),

      l_seconds      = 1 / p_init.model.fps,
      l_milliseconds = 1000 * l_seconds;

  l_canvas.width  = l_canvas_init.width;
  l_canvas.height = l_canvas_init.height;

  function f_update_model()
  {
    l_model_paddle.move(l_seconds);
    l_model_ball.move(l_seconds);

    // a posteriori collision detection and handling
    collision(l_canvas, l_model_ball);
  }

  function f_update_view()
  {
    // clear canvas
    l_context.clearRect(0, 0, l_canvas.width, l_canvas.height);

    // draw the paddle and the ball
    l_view_paddle.draw(l_context);
    l_view_ball.draw(l_context);

    p_window.requestAnimationFrame(f_update_view);
  }

  p_window.setInterval(f_update_model, l_milliseconds);
  p_window.requestAnimationFrame(f_update_view);
}
```

2.5 Kollisionsbehandlung

Als nächstes muss die Kollisionsbehandlung erweitert werden. Bisher wird nur die Kollision zwischen Ball und Bühnenwand erkannt und behandelt. Ebenso muss die Kollision zwischen Schläger und Bühnenwand berücksichtigt werden. Sobald der Schläger die Wand berührt wird er einfach gestoppt. Das heißt, der Benutzer soll den Schläger nicht von der Spielfläche bewegen können. Allerdings kann er ihn natürlich nach jeder Kollision in die Gegenrichtung bewegen. Es gibt noch eine dritte Art der Kollision: Der Schläger berührt den Ball. Diese Kollisionsart wird erst im vierten Teil des Tutoriums behandelt.

Benennen Sie in der Datei „`collision.js`“ zunächst die Funktion „`collision`“ in „`collisionStageBall`“ um. Legen Sie außerdem die zunächst leere Funktion „`collisionStagePaddle`“ an:

```
function collisionStagePaddle(p_stage, p_paddle)
{
}
```

Schreiben Sie nun eine neue Funktion „`collision`“, die die beiden zuvor definierten Hilfsfunktionen der Reihe nach aufruft:

```
function collision(p_stage, p_ball, p_paddle)
{
  collisionStageBall(p_stage, p_ball);
  collisionStagePaddle(p_stage, p_paddle);
}
```

Diese Funktion erwartet im Gegensatz zu vorher nun noch einen drittes Argument. Das heißt, in der Datei „`minipong.js`“ muss der Aufruf

```
collision(l_canvas, l_model_ball);
```

durch den Aufruf

```
collision(l_canvas, l_model_ball, l_model_paddle);
```

ersetzt werden.

Wenn Sie jetzt Ihre Anwendung test, sollte sie wieder laufen, d. h., der Ball sollte sich über die Bühne bewegen und der Schläger sollte gezeichnet werden.

Ergänzen Sie nun die Kollisionserkennung und behandlung für den Schläger. Sobald er eine Wand berührt, wir der mittels der Methode „`stop`“ angehalten. Bei der Kollisionserkennung muss man berücksichtigen, dass sich der Aufhängepunkt des Rechtecks in der linken oberen Ecke befindet. Das heißt, wenn sich der Schläger nach links (`p_paddle.vx < 0`) bzw. oben (`p_paddle.vy < 0`) bewegt, reicht es aus zu überprüfen, ob die x-Position (`p_paddle.x`) bzw. die y-Position

(`p_paddle.y`) des Schlägers den Bühnenrand berührt (`= 0`) oder gar außerhalb (`< 0`) liegt.

Für den rechten (`p_stage.width`) bzw. den unteren Rand der Bühne (`p_stage.height`) muss dagegen jeweils die Breite (`p_paddle.width`) bzw. die Höhe (`p_paddle.height`) des Schlägers zu dessen Position hinzu addiert werden, um den Schläger rechtzeitig zu stoppen.

```
function collisionStagePaddle(p_stage, p_paddle)
{
  if (   (p_paddle.vx < 0 && p_paddle.x <= 0)
      || (p_paddle.vx > 0 && p_paddle.x + p_paddle.width >=
p_stage.width)
     )
  { p_paddle.stop(); }

  if (   (p_paddle.vy < 0 && p_paddle.y <= 0)
      || (p_paddle.vy > 0 && p_paddle.y + p_paddle.height >=
p_stage.height)
     )
  { p_paddle.stop(); }
}
```

Testen können Sie diese Funktion allerdings noch nicht, da sich der Schläger noch bewegen lässt.

2.6 Benutzer-Interaktion

Um den Schläger in Bewegung zu setzen, wird ein neues Modul benötigt: Das Controller-Modul „`keyboard.js`“. Ein Control-Modul hat die Aufgabe, Benutzer-Interaktionen zu verarbeiten. Im Falle des Spiels MiniPong gibt es zwei wesentliche Benutzer-Interaktionen: Er kann den Schläger nach links oder nach rechts bewegen.

In diesem Spiel soll die Schläger-Steuerung mittels der Tastatur erfolgen. Andere Möglichkeiten wären eine Joystick-Steuerung (da könnte man bei analogen Joysticks sogar die Schlägergeschwindigkeit mittels Auslenkungswinkel festlegen) oder eine Touchsteuerung auf Smartphones.

In der Datei „`init.json`“ wurden die Steuertasten bereits festgelegt:

```
"control":
{
  "player":
  {
    "left":  {"key": "ArrowLeft",  "keyCode": 37},
    "right": {"key": "ArrowRight", "keyCode": 39}
  }
}
```

Es wurde jeweils der Name der Steuertaste sowie ihr Key-Code angegeben. Eigentlich gelten Key-Codes als veraltet und sollen daher nicht mehr verwendet werden, aber einige Browser unterstützen die Tastensteuerung per Tastennamen leider noch nicht.

Die Keyboard-Steuerung wird für jeden Spieler mit Hilfe einer Funktion namens „`controlKeyboard`“ aktiviert. Diese Funktion erwartet drei Argumente. Im Parameter „`p_window`“ wird Ihr das Browser-Fenster übergeben, in dem das Spiel läuft. Über dieses „Fenster“-Objekt kann man nicht nur auf das Browser-Fenster, sondern auch auf die Tastatur zugreifen. Letzteres wird hier benötigt.

Im Parameter „`p_init`“ erwartet der Keyboard-Controller ein Objekt, das zwei Attribute „`left`“ und „`right`“ enthält. Für jedes dieser Attribute muss der Tastenname und der zugehörige Key-Code definiert sein. Das in der Init-Datei definiert Objekt „`player`“ enthält genau diese Informationen.

Im Parameter „`p_model_paddle`“ muss der Funktion das Paddle übergeben werden, das mit den im Init-Objekt genannten Tasten gesteuert werden soll.

Die Funktion „`controlKeyboard`“ hat drei Aufgaben:

Wenn das Ereignis „die Bewege-dich-nach-links-Taste wurde gedrückt“ eintritt, muss Sie die Start-Funktion des Paddles folgendermaßen aufrufen: „`p_model_paddle.start("left")`“.
Wenn das Ereignis „die Bewege-dich-nach-rechts-Taste wurde gedrückt“ eintritt, muss Sie die Start-Funktion des Paddles folgendermaßen aufrufen: „`p_model_paddle.start("right")`“.
Wenn eine dieser Tasten losgelassen wird, muss sie den Schläger wieder anhalten: „`p_model_paddle.stop()`“.

Zu diesem Zweck liest Sie die Informationen über die zu verwendenden Tasten zunächst aus dem Init-Objekt „`p_init`“ aus und speichert diese Werte. Dann definiert sie zwei Callback- oder Observer-Funktionen (`o_...`), die Aufgerufen werden, sobald ein Tastaturereignis stattfindet.

Dass diese Methoden bei entsprechenden Tastenaktionen aufgerufen werden, erreicht die Keyboard-Control-Funktion mit Hilfe der Befehle „`p_window.addEventListener(...)`“ am Ende der Funktionsdefinition.

Der Eventhandler „`o_start_paddle_moving`“ überprüft, welche Taste gedrückt wurde. Falls es eine der beiden Steuertasten ist, setzt sie den Schläger in die richtige Richtung in Bewegung. Der Eventhandler „`o_stop_paddle_moving`“ überprüft, welche Taste losgelassen wurde, und stoppt die Schlägerbewegung, falls es sich um eine der beiden Steuertasten handelt.

Beide Funktionen wurden **innerhalb** der Funktion „`controlKeyboard`“ definiert, damit sie auf die in den Variablen „`l_key_left`“, „`l_keycode_left`“, „`l_key_right`“ und „`l_keycode_right`“ gespeicherten Werte zugreifen können. Diese Vorgehensweise ist in JavaScript durchaus üblich.

```
function controlKeyboard(p_window, p_init, p_paddle)
{
  var l_key_left      = p_init.left.key,
      l_keycode_left  = p_init.left.keyCode,

      l_key_right     = p_init.right.key,
      l_keycode_right = p_init.right.keyCode;

  function o_start_paddle_moving(p_event)
  {
    if (p_event.key === l_key_left || p_event.keyCode === l_keycode_left)
      p_paddle.start("left");
    else if (p_event.key === l_key_right || p_event.keyCode ===
l_keycode_right)
      p_paddle.start("right");
  }

  function o_stop_paddle_moving(p_event)
  {
    // If a key is released that controls the movement of
    // the paddle, stop the movement of the paddle.
    if (p_event.key === l_key_left  || p_event.keyCode === l_keycode_left
||
        p_event.key === l_key_right || p_event.keyCode ===
l_keycode_right
       )
      p_paddle.stop();
  }

  p_window.addEventListener("keydown", o_start_paddle_moving);
  p_window.addEventListener("keyup",   o_stop_paddle_moving);
}
```

Der Modulkopf der Datei „`minipong.js`“ muss nochmals erweitert werden, damit die zuvor definierte Funktion „`controlKeyboard`“ verwendet werden kann.

```
define
( ['app/model/ball', 'app/view/ball',
  'app/model/paddle', 'app/view/paddle',
  'app/model/collision',
  'app/control/keyboard'
 ],
 function(ModelBall, ViewBall,
          ModelPaddle, ViewPaddle,
          collision,
          controlKeyboard
         )
...
```

Nun kann der nachfolgende Aufruf dieser Funktion in den Konstruktor „`MiniPong`“ direkt nach dem Variablenblock eingefügt werden.

```
controlKeyboard(p_window, p_init.control.player, l_model_paddle);
```

Nun können Sie Ihre Anwendung testen. Wenn Sie alles richtig gemacht haben, können Sie jetzt den Schläger per Tastensteuerung nach links und rechts bewegen.

Allerdings werden Sie bald feststellen, dass die Kollisionsbehandlung fehlerhaft ist. Wenn der Schläger mit einer Wand kollidiert, stoppt er nicht wie gewünscht, sondern „bohrt“ sich langsam in die Wand. :-(

Dieses Fehlverhalten heißt Penetration (Eindringung). Bei komplexeren geometrischen Anwendungen kann es ziemlich aufwändig, dieses Problem zu beheben. Für die MiniPong-Anwendung ist die Problembehebung allerdings relativ einfach: siehe nächsten Abschnitt.

2.7 Verbesserung der Kollisionsbehandlung

Das zuvor beschriebene Penetrations-Problem entsteht durch die diskrete Berechnung der Modelländerungen und durch die A-posteriori-Kollisionserkennung und -behandlung. Dieses Verfahren funktioniert folgendermaßen:

Voraussetzung: Zum aktuellen Zeitpunkt gibt es keine Kollisionen (da im letzten Schritt alle Kollisionen aufgelöst wurden).
Ein paar Milisekunden später werden die neuen Positionen aller beweglichen Objekte berechnet. Die Objekte werden an die jeweils neue Position verschoben.
Danach (a posteriori, also auf der Basis der Erfahrung der letzten Berechnung) wird überprüft, welche Objekte kollidieren (Kollisionserkennung). Das sind diejenigen Objekte, die sich berühren (seltener, aber erwünschter Fall), und diejnigen, die sich überlappen (häufiger, aber unerwünschter Fall). Der Grund dass sich zwei Objekte überlappen, liegt darin, dass die Kollision eingentlich schon etwas vor dem Berechnungszeitpunkt stattgefunden hat.
Anschließend erfolgt die Kollisionsbehandlung: Kollidierende Objekte werden entfernt (z. B. mit Hilfe eine Explosionsanimation) oder ändern die Bewegungsrichtung – oder allgemeiner – die Geschwindigkeit (der Geschwindigkeitsvektor beschreibt die Bewegungsrichtung sowie die Bahngeschwindigkeit). Als Folge der Kollision können noch viele weitere Aktionen erfolgen: Erhöhung

oder Erniedrigung des Punktestands, Veränderungen der Fähigkeiten des Spieleravatars, Freischaltung eines neuen Levels etc.

Wenn man von kollidierenden Objekten, die sich nicht nur berühren, sondern überlappen, nicht nicht mindestens eines von der Bühne entfernt, sondern lediglich deren Geschwindigkeit ändert, kommt es zu Penetrationsproblemen. Deshalb muss in diesem Fall während der Kollisionsbehandlung eine zweite Aktion erfolgen: **Die fehlerhafte Überlappung muss rückgängig gemacht werden.**

Wenn sich beispielsweise der Schläger nach links bewegt

```
p_paddle.vx < 0
```

und er die linke Wand nicht nur berührt,

```
p_paddle.x == 0
```

sondern in diese ein wenig eingedrungen ist,

```
p_paddle.x < 0
```

muss der Schläger zurück auf die Bühne bewegt werden:

```
p_paddle.x = 0
```

Da dieser Befehl auch korrekt ist, wenn sich Wand und Schläger nur berühren, wird er einfach immer ausgeführt, wenn der Schläger mit der linken Wand kollidiert. Für die drei anderen Wände werden entsprechende Korrekturanweisungen in die Kollisionsbehandlung eingefügt. Man muss nur wieder berücksichtigen, dass der Aufhängepunkt des Rechtecks in der linken oberen Ecke liegt.

```
function collisionStagePaddle(p_stage, p_paddle)
{
  // If the paddle collides with the left wall of the stage,
  // stop it and move it back to the stage.
  if (p_paddle.vx < 0 && p_paddle.x <= 0) // collision detection
  {
    p_paddle.stop(); // collision handling
    p_paddle.x = 0;  // cancellation of penetration
  }

  // If the paddle collides with the right wall of the stage,
  // stop it and move it back to the stage.
  if (p_paddle.vx > 0 && p_paddle.x + p_paddle.width >= p_stage.width)
  {
    p_paddle.stop();
    p_paddle.x = p_stage.width - p_paddle.width;
  }

  // If the paddle collides with the top wall of the stage,
  // stop it and move it back to the stage.
  if (p_paddle.vy < 0 && p_paddle.y <= 0)
  {
    p_paddle.stop();
    p_paddle.y = 0;
  }

  // If the paddle collides with the bottom wall of the stage,
  // stop it and move it back onto the stage.
  if (p_paddle.vy > 0 && p_paddle.y + p_paddle.height >= p_stage.height)
  {
    p_paddle.stop();
    p_paddle.y = p_stage.height - p_paddle.height;
  }
}
```

Anmerkung
Bei der A-priori-Kollisionserkennung und -behandlung wird die Kollisionsberechnung ausgeführt, **bevor** die Objekte bewegt werden. Dabei wird ermittelt, zu welchen exakten Zeitpunkten sie innerhalb des aktuellen Zeitintervalls kollidieren werden. Dies wird bei der Berechnung der neuen Positionen berücksichtigt. Damit kann es nicht zu Penetrationsproblemen kommen. Allerdings ist diese Art der Kollisionsberechnung wesentlich aufwändiger. Vor allem muss man berücksichtigen, dass sich verschiedene Objekte zu verschiedenen Zeitpunkten kollidieren. Und man muss die Kollisionsberechnungen für alle diese Zeitpunkte durchführen.

Die bislang verwendete Berechnung der Kollisionen zwischen Ball und Bühnenwänden leiden übrigens auch an dem Penetratiosnproblem. Bei (fast) jeder Kollision dringen Sie ein wenige in die Wand ein. Sofern der Ball sehr schnell oder die Berechnungsintervalle sehr lang sind, kann es sein, dass der Ball so weit in die Wand dringt, dass der Mittelpunkt hinter der Wand zu liegen kommt. Und dann bleibt

der Ball in der Wand hängen (vgl. Teil 1 des Tutoriums, Abschnitt „Probleme der Kollisionsbehandlung“).

Man muss also auch hier bei einer erkannten Kollision die Überlappung zwischen Ball und Wand rückgängig machen. Man könnte jetzt einfach den Ball soweit zurück auf die Bühne schieben, bis er die Wand nur nicht berührt, aber nicht mehr in sie eindringt. Das wäre aber physikalisch nicht ganz korrekt. Die Wand wirkt für den Ball wie ein Spiegel. Wenn man sich vor einen Spiegel stellt, bemerkt man sofort, dass das Spiegelbild genauso weit von der Spiegelfläche entfernt ist, wie man selbst. Wenn also der Ball um einen Betrag d in die Wand eingedrungen ist, muss man ihn um den Betrag $2\cdot d$ zurück ins Spielfeld schieben. Würde man ihn nur um $1 \cdot d$ verschieben, würde er die Wand berühren. Verschiebt man ihn noch einmal um denselben Betrag, ist er an der Stelle, an der sein Spiegelbild um den Betrag d von der anderen Seite des Spiegel entfernt ist.

Die Verschiebung erfolgt immer senkrecht zur Spiegelfläche. Bei einer waagerechten Wand, muss er also in y-Richtung verschoben werden, und bei einer senkrechten Wand in x-Richtung. Insgesamt sieht die verbesserte Kollisionsfunktion somit folgendermaßen aus.

Wenn der Ball z. B. mit der linken Wand kollidiert, wenn also der Abstand des Balls von der Wand kleiner ist, als sein Radius,

```
p_ball.x <= p_ball.r
```

dann muss wie üblich seine Bewegungsrichtung umgedreht werden:

```
p_ball.vx = -p_ball.vx;
```

Außerdem muss er auf die korrekte Position der Bühne verschoben werden. Wie weit er in die Wand eingedrungen ist, kann man ganz leicht berechnen:

```
p_ball.r - p_ball.x
```

Also muss er senkrecht zur Wand **zweimal** in die Gegenrichtung verschoben werden. Da er in die linke senkrechte Wand eingedrungen ist, muss er um diesen Betrag waagerecht (d. h. in x-Richtung) nach rechts verschoben werden:

```
p_ball.x += 2*(p_ball.r - p_ball.x);
```

Für die anderen Wände erfolgt die Kollisionserkennung und -behandlung analog:

```
function collisionStageBall(p_stage, p_ball)
{
  // If the ball collides with the left or the right wall of the stage
  // mirror its x-velocity and move the ball back onto the stage.
  if (p_ball.x <= p_ball.r)
  {
    p_ball.vx = -p_ball.vx;
    p_ball.x += 2*(p_ball.r - p_ball.x);
  }
  if (p_ball.x >= p_stage.width - p_ball.r)
  {
    p_ball.vx = -p_ball.vx;
    p_ball.x -=  2*(p_ball.r - p_stage.width + p_ball.x);
  }

  // If the ball collides with the top or the bottom wall of the stage
  // mirror its y-velocity and move the ball back onto the stage.
  if (p_ball.y <= p_ball.r)
  {
    p_ball.vy = -p_ball.vy;
    p_ball.y += 2*(p_ball.r - p_ball.y);
  }
  if (p_ball.y >= p_stage.height - p_ball.r)
  {
    p_ball.vy = -p_ball.vy;
    p_ball.y -= 2*(p_ball.r - p_stage.height + p_ball.y);
  }
}
```

3 Erweiterung

Erweitern Sie die Anwendung so, dass Sie zwei Schläger unabhängig voneinander links und rechts am Bühnenrand bewegen können.

Musterlösung: `index_v.html` (WK_MiniPong03a (SVN))
Die Steuerung erfolgt mit den Pfeiltasten. Das zweite Paddle wird mit den Tasten „`w`“ und „`s`“ gesteuert.

4 Quellen

1. **Braun (2011)**: Herbert Braun; Webanimationen mit Canvas; in: c't Webdesign; Band: 2011; Seite(n): 44–48; Verlag: Heise Zeitschriften Verlag; Adresse: Hannover; 2011; Quellengüte: 5 (Artikel)
2. **Kowarschick (MMProg)**: Wolfgang Kowarschick; Vorlesung „Multimedia-Programmierung“; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 3 (Vorlesung)

Kategorien:
Multimedia-Programmierung/Tutorium
HTML5-Tutorium: Canvas: MiniPong
HTML5-Beispiel
Kapitel:Multimedia-Programmierung:Beispiele

Diese Seite wurde zuletzt am 7. Dezember 2016 um 10:11 Uhr bearbeitet.
Inhalt verfügbar unter CC BY-NC-SA 4.0, falls Dokument nach dem 5. 3. 2011 erstellt wurde, sonst CC BY-SA DE 3.0.