4 Level II: Moderne Casual Games

Bis hierhin haben wir hauptsächlich alte und technisch einfache Spielklassiker nachgebaut. Jetzt wollen wir uns moderneren Casual Games und dabei fortgeschrittenen technischen Standards widmen.

4.1 Scrolling

Bislang haben unsere Spiele nur auf einem einzigen Screen stattgefunden, doch auch bei einem Casual Game mag tiefere Immersion beim Spieler erst aufkommen, wenn sich die Spielewelt nicht nur auf die physische Grenze des Displays beschränkt, sondern das Display die Eingangstür zu tieferen Spielewelten bildet. Das Mittel zum Zweck hierfür lautet Scrolling.

Man stelle sich nur einmal ein paar der iPhone-Klassiker ohne Scrolling vor: »Doodle Jump«, »Angry Birds« oder »Canabalt« oder »Mega Jump« auf nur einem Bildschirm? Unvorstellbar langweilig!

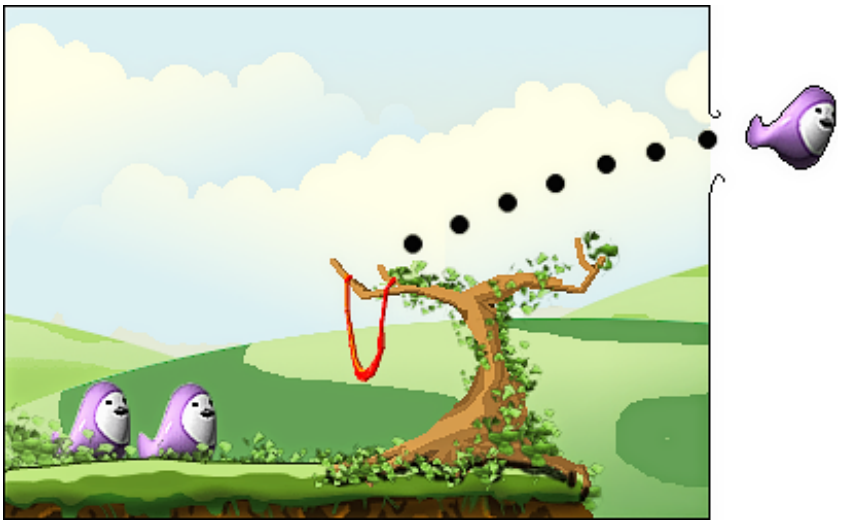

Abb. 4–1 *Macht ohne Scrolling so keinen Spaß.*

In den folgenden Workshops werden wir unsere Welten also in alle Richtungen erweitern und dabei weitere Spiele nachbauen. So werden wir unter anderem den zugrunde liegenden technischen Spielkonzepten von »Doodle Jump«, »R-Type«, »1942«, »Moon Patrol« und »Canabalt« begegnen und diese nachbauen. Doch zuvor werden wir uns um verschiedene Methoden des Scrollings beschäftigen.

Scrolling

Das deutsche Wort dafür lautet eigentlich »Bildlauf«, doch hat sich die englische Bezeichnung auch im deutschen Sprachraum durchgesetzt. »Scrolling« kommt von »scroll«, was im Englischen eine Schriftrolle bezeichnet, die vor allem im Altertum als Vorläufer des Buches existierte. Die mittelalterlichen Schriftrollen wurden von oben nach unten abgerollt, um die Texte lesen zu können, die Tora, die auch heute noch aus kultischen Gründen als Schriftrolle existiert, wird hingegen seitlich abgerollt.

In der Computertechnik hat »Scrolling« allerdings die Bedeutung, dass Inhalte oder Hintergründe pixelweise nachgeführt werden.

Das erste Spiel, das mit Scrolling (seitlich) aufwarten konnte, war Eugene Jarvis »Defender«. Damit erreichte er, dass das Spiel nicht mehr auf die »natürlichen« Bildschirmgrenzen beschränkt war, und schuf darüber hinaus die Illusion einer hohen Spielgeschwindigkeit. Prompt wurde es 1981 zum Spiel des Jahres gewählt und ist bis heute einer der meistverkauften Arcade-Automaten.

4.1.1 Page-Scrolling

Bei einem Spiel wie z. B. einem klassischen Adventure, bei dem es nicht unbedingt darauf ankommt, schnell reagieren zu können, genügt es, die Spielfigur aus dem Bildschirmrand hinaus zu führen und auf der anderen Seite auf gleicher Höhe wieder erscheinen zu lassen, nachdem die neuen Levelelemente auf den Screen gezeichnet wurden.

Dabei ist die Levelgrafik statisch, und nur die Spielfigur bewegt sich.

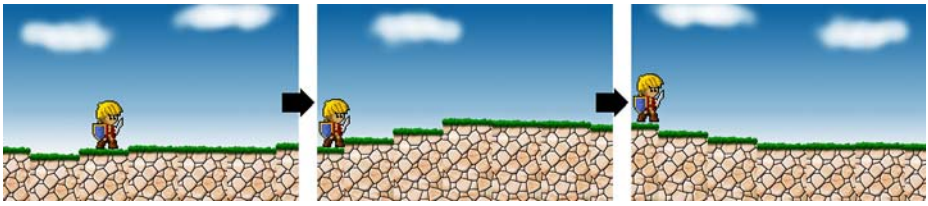

Abb. 4–2 *Die Spielfigur bewegt sich, der Hintergrund ist statisch.*

Diese Technik wurde früher angewendet, als die Computer noch nicht besonders leistungsfähig waren, der Code hierfür ist simpel: Je nachdem, ob die Spielfigur den Bildschirm oben/unten oder an den Seiten verlässt, wird entweder die `x`- oder die `y`-Koordinate beibehalten und die andere Koordinate auf den Wert (Bildschirmhöhe – `spieler.y`) oder (Bildschirmbreite – `spieler.x`) gesetzt. Auf dem iPhone wäre dies in Pseudocode bei horizontaler Vollbildansicht:

```
// Sprite umpositionieren
if (spieler.y < 0)           {spieler.y = screenHeight;}
if (spieler.y > screenHeight) {spieler.y = 0;}
if (spieler.x < 0)           {spieler.x = screenWidth;}
if (spieler.x > screenWidth)  {spieler.x = 0;}
…
/* neuen Level zeichnen */
```

Hier können wir nicht mehr einfach so die Bildschirmposition mit der Position des Sprites gleichsetzen, da der Level breiter als das Display ist, sonst käme unsere Spielfigur nie über den ersten Screen hinaus. Stattdessen müssen wir nun für die x- und y-Position des Sprites jeweils eine weitere Objekteigenschaft oder -variable einführen, die sich rein auf die Levelkoordinaten bezieht, das heißt, die besagt, wo innerhalb des Levels sich unser Sprite absolut befindet (und nicht nur relativ zum Viewport-Ursprung). Wir trennen ab sofort also die Darstellung des Sprites auf dem Display von seiner eigentlichen Position innerhalb des Levels. Bislang mussten wir einfach nur die Position der `UIImageView`, die das Sprite enthielt, abfragen und konnten von dieser Angabe aus alles Weitere berechnen. Jetzt arbeiten wir mit virtuellen Positionen im Hintergrund und transformieren diese auf das Display.

Um den Spieler also auch innerhalb des Levels weiterzubewegen, müssen die x- und y-Koordinaten, die sich auf den Level (und nicht auf den Bildschirm) beziehen, angepasst werden. Dies geschieht mit der folgenden Erweiterung des Pseudocodes:

```
if (spieler.y < 0)           {
                             spieler.y = screenHeight;
                             spielerPosImLevelY -= screenHeight;
}
if (spieler.y > screenHeight) {
                             spieler.y = 0;
                             spielerPosImLevelY += screenHeight;
}
if (spieler.x < 0)           {
                             spieler.x = screenWidth;
                             spielerPosImLevelX -= screenWidth;
}
if (spieler.x > screenWidth)  {
                             spieler.x = 0;
                             spielerPosImLevelX += screenWidth;
}
/* neuen Level zeichnen */
```

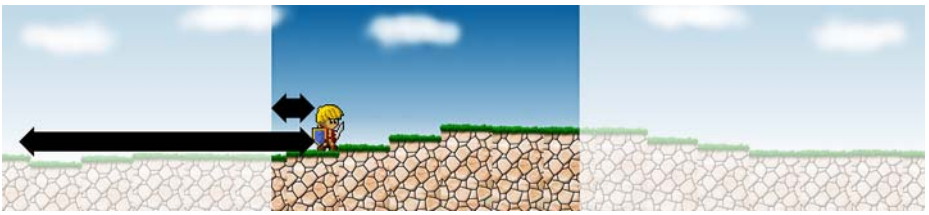

Abb. 4–3 *Unterschied zwischen Spieler.center.x (oberer Pfeil) und spielerPosImLevelX (unterer Pfeil)*

In der Abbildung wird nun auch deutlich, was der Unterschied zwischen den Objektvariablen `spieler.x` und `spielerPosImLevelX` ist: `spieler.x` ist die x-Koordinate des **Sprites** und kann auf dem iPhone praktisch die Werte 320 bzw. 480 nicht überschreiten. `spielerPosImLevelX` hingegen ist die Position des **Spielers** innerhalb des Levels. Sie entspricht der Anzahl der Pixel, die sich das Sprite in x-Richtung bereits bewegt hat, wenn der Wert am Levelstart 0 betrug. Analog verhält es sich natürlich mit den jeweiligen y-Koordinaten und -Werten.

Diese lassen sich allerdings noch leichter und vor allem performanter berechnen (indem wir wieder unseren alten Modulo-Trick verwenden):

```
spieler.center.x = spielerPosImLevelX % screenWidth;
spieler.center.y = spielerPosImLevelY % screenHeight;
```

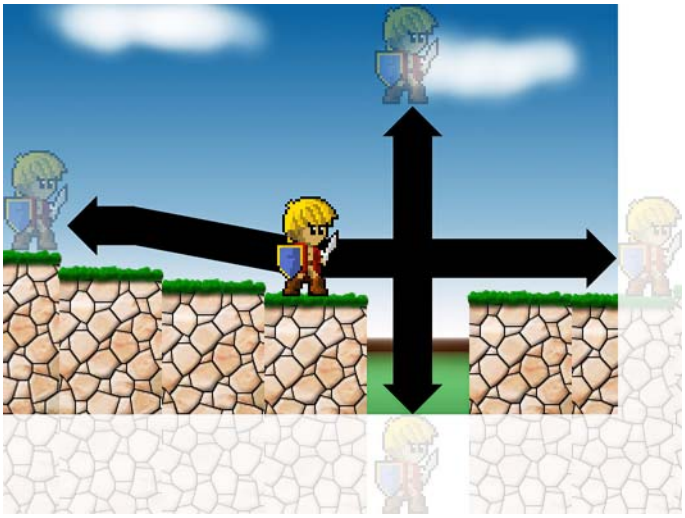

Abb. 4–4 *Oben und links wechselt ohne Korrektur bereits die Levelansicht, obwohl sich das Sprite noch im sichtbaren Bereich befindet, unten und rechts hingegen wird korrekt umgeblendet.*

Dies ist die einfachste Version des Bildschirmwechsels, allerdings wird diese Vorstufe des »Scrollens« heutzutage kaum noch mehr verwendet. Sie kann aber außerhalb des Arcade-Genres gerne als Stilmittel eingesetzt werden, um bewusst einen Retro-Effekt zu erzielen oder wenn der Spieler nicht sehen soll, was ihn im Folgescreen erwartet.

4.1.2 Einfaches Scrolling

Wesentlich spannender ist es aber, wenn der Level parallel zu den Aktionen der Spielfigur scrollt und die Spielfigur so selbst zum Mittelpunkt des Interesses wird, indem sie nun fix im Bildschirmmittelpunkt und somit im Zentrum des Geschehens gerendert und stattdessen die Levelgrafik bewegt wird.

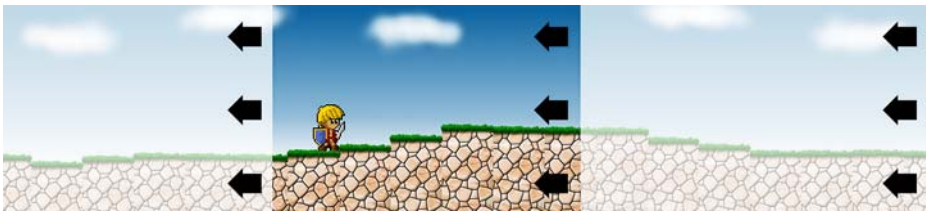

Abb. 4–5 *Die Spielfigur ist auf festen Displaykoordinaten fixiert, der Hintergrund scrollt vorbei.*

In diesem Fall wird die Spielfigur zwar animiert, aber die Hintergrundelemente werden verschoben, das heißt, die Kamera hat stets die Hauptfigur im Fokus und die Landschaft zieht vorbei, als würde man parallel in einem Zug sitzen und das Geschehen aus dem Fenster beobachten. Soll sich unser Sprite dabei in der Bildschirmmitte befinden (um z.B. gleichmäßig rechtzeitig auf alle Gegner reagieren zu können), würde unser Code wie folgt aussehen:

```
spieler.x = screenWidth/2;  // fixe Position
spieler.y = screenHeight/2; // fixe Position
```

bei automatischem Scrolling:

```
hintergrund.x --;
```

oder wenn nur bei Spielerbewegung gescrollt werden soll:

```
hintergrund.x = -spielerPosImLevelX;
```

Während beim Page-Scrolling die Displayposition des Sprites ständig zwischen 0 und Displaybreite bewegt wird und im Falle der Bildschirmüberschreitung der Levelhintergrund einmal um die Breite des Viewports versetzt wird, bewegt sich beim echten Scrolling die x-Variable gar nicht mehr, stattdessen wird der Levelhintergrund jeweils um ein Pixel in die Gegenrichtung versetzt. Bei vertikalem Scrolling wie z.B. bei einem Shooter, bei dem das Raumschiff nach oben fliegt und die Gegner von oben kommen, verhält es sich mit den jeweiligen y-Koordinaten ebenso.

4.1.3 Parallax-Scrolling

Parallax-Scrolling ist eine besondere Methode des echten Scrollings, bei dem ein Pseudo-3D-Effekt erzeugt wird, indem verschiedene Ebenen in unterschiedlichen Geschwindigkeiten scrollen. Das hinterste Layer (z.B. Gebirge und Wolken im

Hintergrund) bewegt sich am langsamsten, das zweithinterste Layer (näherer Hintergrund wie z.B. Häuser am Straßenrand) bewegt sich etwas schneller und das vorderste Layer (die eigentliche Spielebene mit Sprites, Plattformen und Gegnern) am schnellsten. Diese Form werden wir später bei unserem Spiel »The Little Jungle Sisters« näher kennenlernen.

Fassen wir also zusammen: Alle Scrolling-Methoden haben gemeinsam, dass alle abzubildenden Spielelemente zwei Koordinaten benötigen. Die eine, die wir bisher kannten, ist für die reine (visuelle) Darstellung eines Sprites auf dem Display zuständig. Sie ist losgekoppelt von der (virtuellen) Position des Sprites innerhalb eines Levels und stellt diese über Koordinatentransformation auf dem Display dar. Lediglich Sprite-Kollisionen können wir damit noch feststellen.

Die Unterschiede zwischen den Scrolling-Methoden sind vor allem technischer und performanter Natur: Wenn sich ohne Scrolling das Sprite und zwei Gegner auf dem Bildschirm bewegen, dann muss man nur drei ImageViews bzw. Sprites bewegen. Scrollt der Hintergrund allerdings mit, ist das eigene Sprite das einzige Element, dessen Position und Darstellung theoretisch nicht berechnet werden muss, aber auch hier kann es sein, dass sich eine der beiden Koordinaten verändert (z.B. weil der Spieler bei einem Jump'n'Run springt oder auf eine tiefer gelegene Plattform fällt oder in einem Weltraumshooter ein Ausweichmanöver fliegt). Alle anderen Elemente hingegen erhalten neue Positionen, auch jene, die noch gar nicht sichtbar sind! Es gibt für den Prozessor also eine ganze Menge zu tun. Damit wir trotzdem innerhalb der gewünschten 30 Frames per Second bleiben, müssen wir unsere Anstrengung im Code darauf konzentrieren, wie wir einen flüssigen Bildlauf mit so wenig Rechnungen wie möglich erreichen. Hier wird die Spieleprogrammierung also ein wenig komplizierter, aber das hält uns nicht davon ab, gleich einen Klassiker nachzubauen, der auf dem iPhone seit Jahren in den Top 10 der kostenpflichtigen Downloads zu finden ist, mittlerweile als Inbegriff für einen Vertical-Scroller gilt und 2010 sogar mit dem Apple Design Award ausgezeichnet worden ist. Um keine Urheberrechtsverletzungen zu riskieren, nennen wir das Spiel einfach »Noodle Jump« und lassen darin eine Makkaroni so weit wie möglich über Plattformen in die Höhe springen.