KAPITEL

12 Greenfoot und die Kinect

Lernziele

Themen: Programmierung mit der Microsoft Kinect

Konzepte: Body-Tracking

Wir haben bisher gelernt, viele verschiedene und nützliche Programmierkonstrukte zu verwenden – jetzt sind wir bereit, diese in einem anderen Kontext anzuwenden. In diesem Kapitel werden wir keine neuen Programmierkonzepte oder Java-Konstrukte einführen, sondern unsere Programmierfähigkeiten einsetzen, um ein spannendes und interessantes Spielzeug auszuprobieren: die Microsoft Kinect.

Die Kinect ist ein Sensormodul, das eine Farbkamera, einen Infrarotprojektor und eine Infrarotkamera enthält (Abbildung 12.1). Der Infrarotprojektor und die Kamera ermöglichen es, die Tiefe des Bildes vor dem Gerät zu erfassen: die Kinect kann also in drei Dimensionen „sehen".

Abbildung 12.1
Die Microsoft Kinect[1].

Für die Übungen in diesem Kapitel benötigst du Zugang zu einer Kinect. Falls du keine besitzt, kannst du dieses Kapitel ruhig überspringen. Wenn du mit dem Gedanken spielst, dir eine Kinect für die Programmierung mit Greenfoot anzuschaffen, solltest du vorher unbedingt die Bemerkungen unter *Purchasing a Kinect* auf der Greenfoot-Website lesen.[2] Nicht alle Modelle der Kinect funktionieren mit Greenfoot.

In diesem Kapitel werden wir uns eine Reihe von Szenarien ansehen, die die Kinect einsetzen. Wir werden zuerst einige fertige Szenarien untersuchen, um die grund-

1 Bildquelle: Official Windows Magazine/Getty Images.

2 *http://greenfoot.org/doc/kinect#purchase*
Hier findest du Hinweise (in Englisch), welches Modell der Microsoft Kinect sich zum Einsatz mit Greenfoot eignet.

legenden Mechanismen zu verstehen, und anschließend selbst ein wenig Code dafür schreiben.

12.1 Was kann die Kinect?

Die Kinect kann Menschen und ihre Bewegungen vor ihren Kameras erkennen. Durch Greenfoot und die Kinect können wir Informationen über die Anzahl der anwesenden Personen vor dem Gerät erhalten, über ihre Position und ihre Körperbewegungen. Dadurch können wir Programme schreiben, die durch unsere Körperbewegungen vor dem Computer gesteuert werden.

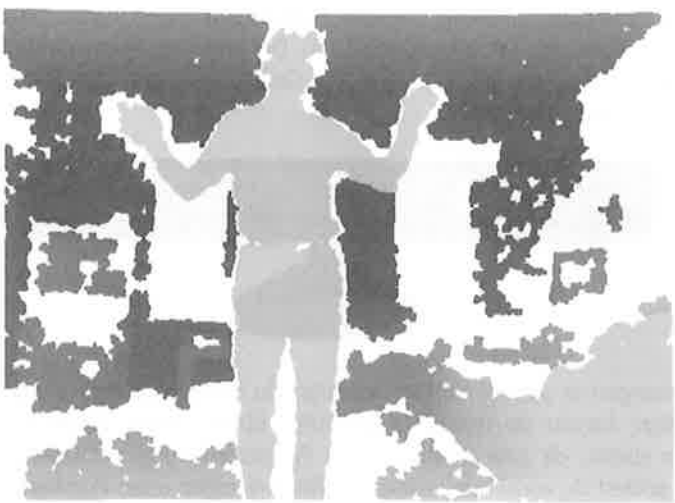
Kamerabild

Als Erstes (und einfachstes) liefert uns die Kinect ein Bild von ihrer Kamera. Dies ist ein ganz normales Bild wie von jeder anderen Webcam. Wir können dieses Bild in Greenfoot benutzen.

Tiefenbilder

Zusätzlich zu dem normalen Farbbild können wir Tiefeninformationen bekommen. „Tiefe" ist der Abstand von der Kamera zu jedem Punkt davor. Abbildung 12.2 zeigt die Tiefe als ein Graustufenbild: die Helligkeit jedes Pixels zeigt seinen Abstand von der Kamera.

Abbildung 12.2
Das Tiefenbild.



Körperkonturen

Die Kinect kann Personen in ihrem Sichtfeld erkennen und einigermaßen zuverlässig ihre Konturen im Kamerabild festhalten (Abbildung 12.3). Sie kann zwischen mehreren Personen unterscheiden und Informationen über die Position eines menschlichen Körpers geben, und zwar separat für jede Person in ihrem Sichtfeld.

Abbildung 12.3
Erkennen von Körperkonturen.

Skelettales Tracking

Die nützlichste Funktion, die Kinect anbietet, ist das *skelettale Tracking*: das Identifizieren der Position in allen drei Dimensionen von verschiedenen Punkten unseres Körpers (Abbildung 12.4). Dadurch wird es ganz einfach, Programme zu schreiben, die durch Körperbewegungen gesteuert werden. Wir können zum Beispiel ein Szenario schreiben, in dem eine Aktion ausgelöst wird, wenn wir einen Greenfoot-Akteur mit unserer rechten Hand berühren oder wenn unser linker Fuß über unser rechtes Knie angehoben wird. Man braucht nicht viel Vorstellungskraft, um Ideen zu entwickeln, wie wir diese Eingaben benutzen können, um interessante Szenarien zu schreiben, die man gerne spielt.

Konzept

Skelettales Tracking ist das Identifizieren und Protokollieren („Tracking") von einigen definierten Punkten auf einem Körper, wie Händen, Ellbogen, Kopf, Füße und so weiter.

Abbildung 12.4
Skelettales Tracking.

12.2 Die Software installieren

Konzept

Kommunikation mit Hardwarekomponenten wird in der Regel durch **Treibersoftware** bewerkstelligt.

Bevor du die Microsoft Kinect mit Greenfoot benutzen kannst, musst du Treibersoftware installieren. Diese Software ist für die Kommunikation mit der Hardware zuständig und stellt eine Softwareschnittstelle zur Verfügung, um das Gerät zu steuern.

Wenn Greenfoot-Kinect-Szenarien ausgeführt werden, dann muss im Hintergrund der *Greenfoot-Kinect-Server* laufen, damit unser Greenfoot-Szenario mit der Kinect-Hardware kommunizieren kann. Der Greenfoot-Kinect-Server wiederum verwendet eine Bibliothek namens *OpenN1*. Abbildung 12.5 zeigt die Kommunikationsebenen zwischen deinem Kinect-Szenario und der Kinect-Hardware.

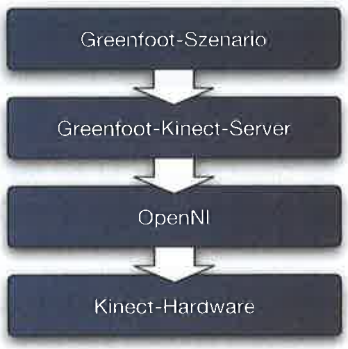


Abbildung 12.5 Kommunikationsebenen zwischen Greenfoot und der Kinect.

Ausführliche Installationsanweisungen für diese Softwarekomponenten findest du auf der Greenfoot-Website unter *http://www.greenfoot.org/doc/kinect*.

Es gibt hier Anleitungen für Windows, Linux (Ubuntu) und Mac OS X. Beachte die Anweisungen sorgfältig, bevor du zu den folgenden Beispielen weitergehst.

Den Greenfoot-Kinect-Server ausführen

Wenn wir unsere Greenfoot-Kinect-Szenarien ausführen möchten, müssen wir zuerst den Greenfoot-Kinect-Server starten. Dieser wird nun ununterbrochen im Hintergrund laufen und solange wir mit Greenfoot und der Kinect arbeiten, dürfen wir ihn nicht beenden.

Die oben erwähnten Installationsanweisungen erklären ausführlich, wie dieser Server gestartet wird. (Falls du ihn über ein Terminal ausführst, dann wirst du eine Menge Warnhinweise bekommen. Mach dir darüber keine Gedanken – das ist normal.)

12.3 Die ersten Schritte mit Greenfoot und Kinect

Für den Anfang wollen wir uns zunächst ein sehr einfaches Beispiel ansehen: die Anzeige des Kamerabilds auf dem Bildschirm. Wir werden ein Szenario namens *simple-camera* verwenden, das du bei den Buchszenarien findest.

Übung 12.1 Verbinde deine Kinect mit deinem Computer. Starte den Greenfoot-Kinect-Server. Dann starte Greenfoot, öffne das Beispiel *simple-camera* und führe es aus.

Wenn die Installation geklappt hat, dann solltest du jetzt das Bild von der Kinect-Kamera in deiner Welt sehen.

Im Klassendiagramm des Szenarios findest du eine Reihe von Klassen. Die Klassen **`KinectWorld`**, **`Joint`**, **`KinectClient`**, **`Point3D`** und **`UserDate`** sind Teil der Greenfoot-Kinect-Infrastruktur – sie sind in allen Szenarien vorhanden, die mit Kinect zu tun haben. Die einzige Klasse, die es nur in diesem Szenario gibt und die unseren Code enthält, ist **`MyWorld`**.

Bevor wir unseren eigenen Code ausprobieren, wollen wir uns noch kurz die Standardklassen für Greenfoot-Kinect ansehen.

KinectWorld

Diese Klasse sollte als Oberklasse für all deine eigenen Welt-Klassen in Kinect-Szenarien verwendet werden. Sie stellt die grundlegende Kommunikation mit der Kinect zur Verfügung und gibt dir Zugang zu den Daten (Kamerabild und Benutzerdaten), die wir von der Kinect erhalten. Die Methoden dieser Klasse solltest du dir irgendwann einmal ansehen. Dazu kannst du einfach vom Editor zur Dokumentationsansicht wechseln – die Implementierung ist im Moment nicht wichtig.

KinectClient

Diese Klasse wird intern von **`KinectWorld`** verwendet – du solltest sie nicht direkt benutzen.

UserData

Dies ist eine Klasse, in der Objekte definiert werden, die Informationen über einen Benutzer speichern, der von Kinect protokolliert wird. Du kannst pro Nutzer, der für die Kinect sichtbar ist, ein Objekt erhalten. **`UserData`**-Objekte bekommst du über die Methoden deiner **`KinectWorld`**.

Joint

Über das **`UserData`**-Objekt wiederum kannst du Informationen über bestimmte Punkte am Körper des Nutzers, sogenannte Joints, bekommen (dies sind wichtige Gelenke wie die rechte Hand, der Ellbogen, das Knie). Dazu dienen die Objekte der Klasse **`Joint`**.

Point3D

Ein Teil der **`Joint`**-Information schließlich ist die (dreidimensionale) Position des Gelenks, auf die du in Form eines **`Point3D`**-Objekts Zugriff hast.

Der Ordner mit den Buchszenarien enthält ein Szenario namens *kinect-start*, in dem sich ausschließlich diese Klassen befinden. Du kannst es als Ausgangsszenario für deine eigenen Projekte verwenden.

12.4 Die einfache Kamera

Der einzige Code im Szenario *simple-camera*, der nur zu diesem Szenario gehört, befindet sich in der Klasse **MyWorld** (Listing 12.1). Lass uns zuerst einen Blick darauf werfen.

Listing 12.1: Die Welt-Klasse des Szenarios *simple-camera*.

```
import greenfoot.*;

/**
 * Dies ist eine sehr einfache Demo-Welt, die Kinect verwendet:
 * Diese Welt zeigt das Bild von der Kinect-Kamera als Hintergrund der Welt.
 *
 * @author Michael Kölling
 * @version 1.0
 */
public class MyWorld extends KinectWorld
{
    /**
     * Konstruktor für unsere Welt. Es gibt nichts zu tun.
     */
    public MyWorld()
    {
        super();
    }

    /**
     * In jedem Act-Zyklus wird das Bild von Kinect genommen und und als
     * Hintergrundbild benutzt. (Nicht vergessen: Die Act-Methode der
     * Oberklasse muss in allen Kinect-Scenarios aufgerufen werden.)
     */
    public void act()
    {
        super.act();

        GreenfootImage cameraImage = getThumbnailUnscaled();
        setBackground(cameraImage);
    }
}
```

Der Konstruktor von **MyWorld** ruft einfach den Standardkonstruktor von **KinectWorld** auf.

Von der Kinect-Hardware erhalten wir – als Voreinstellung – ein Bild von 640 × 480 Pixel und der Standardkonstruktor von **KinectWorld** erzeugt eine Welt derselben Größe, sodass dies zueinander passt. (Bei der Verwendung anderer Konstruktoren kannst du den Welten auch unterschiedliche Größen geben und dann die Kinect-Eingabe darauf anpassen, aber das ist jetzt nicht nötig.)

Der interessante Code in dieser Klasse ist in der **act**-Methode. Die erste Anweisung ist ein Aufruf der **act**-Methode der Oberklasse. Dieser Aufruf ist wichtig: Die **act**-Methode von **KinectWorld** verwaltet die Kommunikation mit dem Server und sie muss immer aufgerufen werden. Lösche deshalb diese Zeile nicht.

Spannend wird es in den letzten beiden Zeilen: Hier rufen wir die Methode **`getThumbnailUnscaled`** auf, die uns das Kamerabild der Kinect in seiner Standardgröße liefert, und zwar als ein **`GreenfootImage`**. Die letzte Zeile übernimmt dieses Bild dann als Welthintergrund.

Übung 12.2 Sieh dir die Methoden der Klasse **`KinectWorld`** an. Du kannst diese Methoden alle in deiner eigenen Welt-Klasse aufrufen. Welche der Methoden liefern dir ein Kamerabild? Was ist der Unterschied zwischen ihnen? Halte deine Antwort schriftlich fest.

Übung 12.3 Über welche Methoden kann man Nutzerdaten erhalten? Was ist der Unterschied zwischen ihnen?

12.5 Der nächste Schritt: greenscreen

Das Bild von der Kamera ist ein hübscher erster Schritt, aber dazu hätten wir auch irgendeine Webcam nehmen können. Wir wollen uns nun die erste von Kinects Fähigkeiten anschauen: Benutzer identifizieren.

Ein „Greenscreen" ist eine Technik, die schon seit Langem in der Film- und Fernsehproduktion eingesetzt wird, um Schauspieler vor Hintergründen zu platzieren, an denen sie nicht wirklich sind. (Führe eine Internetsuche durch, um mehr darüber zu erfahren!) Dazu werden in der Regel Schauspieler vor einen grünen Hintergrund gestellt, danach wird jegliche grüne Farbe aus der entstandenen Aufnahme herausgefiltert und ersetzt. Mithilfe der Kinect können wir leicht etwas Ähnliches machen, weil die Kinect zwischen menschlichen und nicht menschlichen Objekten und Hintergrund in seinem Sichtfeld unterscheiden kann.

Übung 12.4 Öffne das Szenario *greenscreen* und führe es aus. Was macht es?

Gegenüber dem Szenario *simple-camera* haben wir zwei Veränderungen vorgenommen:

- Wir haben den Code, der das Bild anzeigt, zu einem Akteur verschoben, anstatt ihn als Welthintergrund anzuzeigen. Nun platzieren wir einen einzelnen Akteur in die Mitte der Welt und dieser Akteur nimmt das Kinect-Bild als sein Akteur-Bild. Das Ergebnis ist ähnlich wie vorhin: Das Bild hat dieselbe Größe wie die Welt und füllt den gesamten Welt-Bereich aus.
- Wir haben den Aufruf der **`getThumbnailUnscaled`**-Methode durch einen Aufruf von **`getCombinedUserImage`** ersetzt. Diese Methode gibt uns ebenfalls das Kamerabild, filtert aber den Hintergrund heraus und liefert uns damit ein Bild, auf dem nur der sichtbare Benutzer ist.

Die Darstellung der Benutzer ohne Hintergrund ist also sehr einfach, da die Kinect den schwierigen Teil der Aufgabe für uns übernimmt. Da das Benutzerbild als Akteur angezeigt wird, können wir jetzt unseren eigenen Hintergrund einbringen.

Übung 12.5 Lege ein Hintergrundbild für die Welt fest. Dazu kannst du die Funktion BILD AUSWÄHLEN aus dem Kontextmenü von **`MyWorld`** verwenden. Das Szenario bietet dir ein Bild namens *weather-map.png* an, das du nehmen kannst.

Übung 12.6 Probiere das Szenario mit verschiedenen anderen Hintergrundbildern aus.

Diese beiden Beispiele – *simple-camera* und *greenscreen* – benutzten nur das Bild des Benutzers, entweder als Welthintergrund oder als Akteur-Bild. Dies ist ein interessanter erster Schritt, aber wenn wir mit anderen Objekten auf dem Bildschirm interagieren möchten, dann müssen wir noch einen Schritt weitergehen: Wir müssen herausfinden, an welcher Stelle des Bildschirms sich der Benutzer befindet. Zum Glück hilft uns Kinect auch dabei.

12.6 Strichmännchen: Benutzer nachzeichnen

Das Szenario *stick-figure* im Buchordner zeigt ein erstes Beispiel für das Nachzeichnen („Tracking") von Benutzern. Hier werden wir nicht das Kamerabild des Benutzers anzeigen, sondern die Koordinaten von Joint-Punkten des Benutzers verwenden, um ein Bild zu zeichnen.

Übung 12.7 Öffne das Szenario *stick-figure* und führe es aus. Stelle dich vor deine Kinect und beobachte, was passiert.

Wenn du das Szenario ausprobierst, siehst du, dass der Benutzer vor der Kamera als Strichmännchen nachgezeichnet wird (Abbildung 12.6). Vielleicht ist dir auch aufgefallen, dass es manchmal schwierig ist festzustellen, ob du im Sichtfeld der Kinect-Kamera bist und wo genau du stehen musst, damit du nachgezeichnet werden kannst. Um dieses Problem zu lösen, werden wir zuerst ein Miniaturbild, ein sogenanntes *Thumbnail-Bild*,[3] in dieses Szenario einfügen, bevor wir fortfahren zu untersuchen, wie das Strichmännchen gezeichnet wird.

Ein Thumbnail-Bild ist ein kleines Bild, das uns hilft, einen schnellen Überblick zu gewinnen. Es heißt „Thumbnail" aufgrund seiner geringen Größe; es ist nicht dazu gedacht, Details zu erkennen, sondern nur um die wesentlichen Informationen zu

3 „Thumbnail" heißt übersetzt „Daumennagel".

bekommen. Wir werden sehen, dass wir uns mithilfe eines Thumbnail-Bilds in vielen Kinect-Szenarien besser vor der Kamera platzieren können.

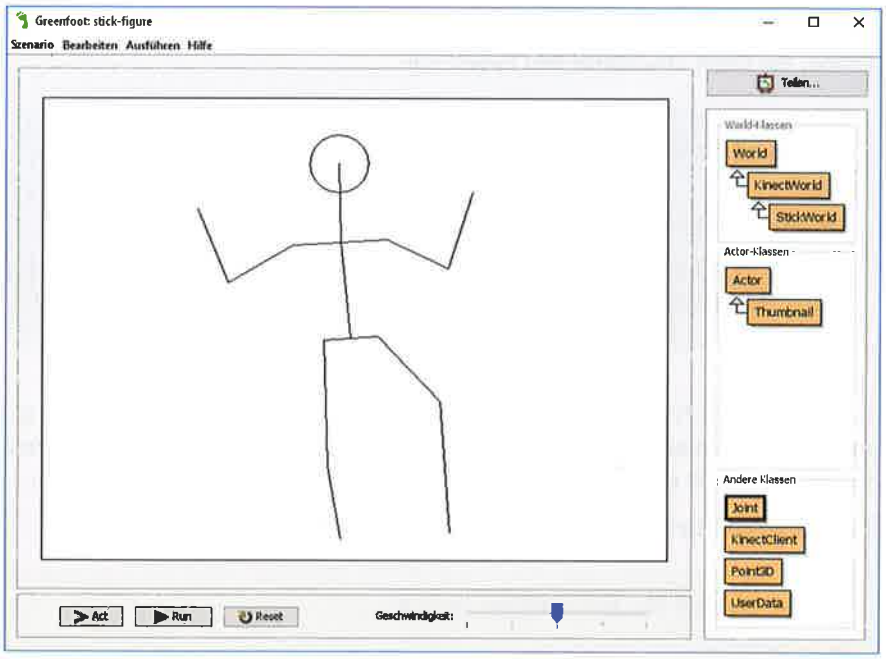


Abbildung 12.6
Das Beispielszenario *stick-figure*.

Im Szenario *stick-figure* findest du eine Klasse namens **Thumbnail**.

Übung 12.8 Führe dein Szenario *stick-figure* aus. Erzeuge interaktiv ein Objekt der Klasse **Thumbnail** und platziere es in der Welt. Was siehst du?

Übung 12.9 Füge Code zu der **StickWorld**-Klasse hinzu, mit dem ein **Thumbnail**-Objekt erzeugt und automatisch hinzugefügt wird. Platziere das Objekt dort, wo du möchtest.

Übung 12.10 Ändere den Ort deines Thumbnail-Bilds, sodass es jetzt in der unteren rechten Ecke deines Bildschirms angezeigt wird.

Die Implementierung der Klasse **Thumbnail** ist recht einfach. Sie enthält nur diese zwei Codezeilen in ihrer **act**-Methode:

```
KinectWorld world = (KinectWorld)getWorld();
setImage(world.getThumbnailUnscaled());
```

Wir können sehen, dass der Thumbnail-Akteur nur das Thumbnail-Bild von **KinectWorld** erhält und als eigenes Bild verwendet.

Im Konstruktor der Klasse **StickWorld** befindet sich der zweite Teil dazu:

```
public StickWorld()
{
    super(THUMBNAIL_WIDTH, THUMBNAIL_HEIGHT, 1.0, false);
}
```

StickWorld ruft also einen anderen **KinectWorld**-Konstruktor auf, und zwar einen, der vier Parameter hat. Die ersten beiden Parameter (die in diesem Fall Konstanten sind) bestimmen die Größe des Thumbnail-Bilds, das wir auf Anfrage von **KinectWorld** erhalten möchten. Auf diese Weise definieren wir die genaue Größe, die unser Thumbnail-Bild haben soll.

Übung 12.11 Was ist die genaue Größe des Thumbnail-Bilds in diesem Beispiel?

Übung 12.12 Verdoppele die Größe des Thumbnail-Bilds.

Übung 12.13 Welche Bedeutung haben die anderen beiden Parameter des **KinectWorld**-Konstruktors, der in dem Aufruf von **super** aufgerufen wird?

Nun, da wir ein Thumbnail-Bild angezeigt haben, ist es einfacher, uns selbst vor der Kamera zu platzieren und zu wissen, wo wir stehen müssen. Wir können nun untersuchen, wie dieses Szenario es schafft, das Strichmännchen zu zeichnen.

Der Code dazu befindet sich in der Klasse **StickWorld** (Listing 12.2).

Listing 12.2: Strichmännchen der Benutzer nachzeichnen.

```
/**
 * Aktion: Benutzer als Strichmännchen anzeigen.
 */
public void act()
{
    super.act();
    UserData[] trackedUsers = getTrackedUsers();
    paintStickFigures(trackedUsers);
}

/**
 * Malt für jeden Benutzer, den wir sehen können, Strichmännchen
 * auf den Welthintergrund.
 */
private void paintStickFigures(UserData[] trackedUsers)
{
    eraseBackground();

    for (UserData user: trackedUsers)
    {
        user.drawStickFigure(getBackground(), 60);
    }
}

/**
 * Löscht den Welthintergrund.
 */
private void eraseBackground()
{
    getBackground().setColor(Color.WHITE);
    getBackground().fill();
}
```

Wenn wir uns den Code genau anschauen, stellen wir fest, dass das Zeichnen des Strichmännchens einfach ist, weil die Klasse **UserData** eine Methode namens **drawStickFigure** zur Verfügung stellt. Für uns bleibt nur noch Folgendes zu tun:

- Bei jedem ACT-Schritt erhalten wir mithilfe der Methode **getTrackedUsers** ein Feld aller nachgezeichneten Benutzer. Wir übergeben dieses Feld unserer eigenen **drawStickFigure**-Methode.
- Dann löschen wir den Welthintergrund (um damit früher gezeichnete Strichmännchen zu entfernen) und zeichnen neue Strichmännchen, indem wir die Methode **drawStickFigure** für jeden Nutzer in unserem Feld aufrufen.

Wie man sieht, wird das Zeichnen eines Strichmännchens eines Benutzers gut unterstützt. Diese Funktion wird sich noch in vielen unterschiedlichen Kinect-Szenarien als nützlich erweisen. Außerdem sollte unser Code in der Lage sein, gleichzeitig Strichmännchen mehrerer Benutzer zu verarbeiten.

Übung 12.14 Suche die Methode **drawStickFigure** in der Klasse **UserData**. Wie lauten die beiden Parameter?

Übung 12.15 Führe das Szenario noch einmal aus und stelle mehr als eine Person vor die Kamera. Zeichnet das Szenario jeden einzelnen korrekt nach?

Übung 12.16 Wie viele Personen können gleichzeitig nachgezeichnet werden? Kannst du eine Grenze angeben? Probiere aus, so viele Leute wie möglich vor die Kamera zu bringen.

12.7 Mit den Händen malen

Die Beispiele, die wir bisher gesehen haben, waren größtenteils fertig, wenn wir sie geöffnet haben, und wir haben sie benutzt, um an ihnen einige grundlegende Konzepte zu erlernen. Es gab für uns nicht viel zu tun.

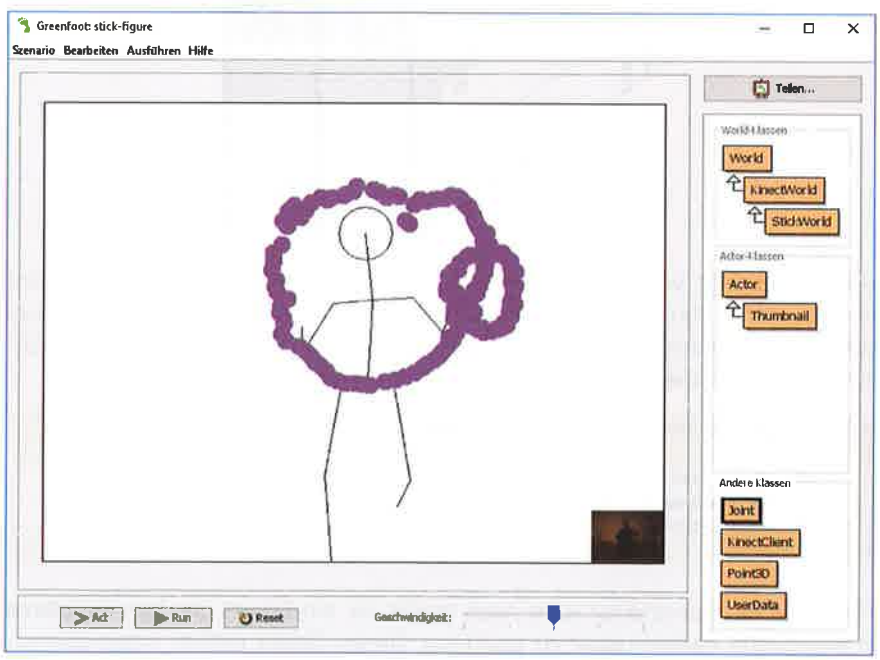


Abbildung 12.7
Mit deinen Händen malen.

Wir haben jetzt ein Grundverständnis der Kinect-Schnittstelle gewonnen, nun können wir beginnen, eigenen Code zu schreiben. Unser nächstes Ziel ist es, ein Szenario zu schreiben, in dem wir mit durch die Bewegung unserer Hände und Füße auf den Bildschirm malen können (Abbildung 12.7).

Übung 12.17 Öffne das Szenario *body-paint-start* und starte es. Probiere aus, was es macht.

Konzept

Protokollierte Benutzer sind Benutzer, die aktuell von der Kinect-Hardware erkannt und analysiert werden.

Wir beginnen wieder wie üblich, indem wir ein Szenario starten, *body-paint-start*, das rudimentären Code enthält. Wenn du es ausprobierst, wirst du feststellen, dass es sich erst einmal sehr ähnlich wie das Beispiel *stick-figure* verhält.

Der erste sichtbare Unterschied ist, dass es hier eine neue Klasse gibt: **Canvas**.

In diesem Szenario wird für jeden protokollierten Benutzer eine eigene Ebene in Form eines **Canvas**-Objekts erzeugt wird. Jedes **Canvas**-Objekt hat ein transparentes Bild in der Größe des gesamten Bildschirms. Da das Bild anfangs leer ist (völlig transparent), ist es unsichtbar, aber dadurch bekommt jeder Nutzer eine eigene transparente Ebene, auf der er malen kann (Abbildung 12.8).

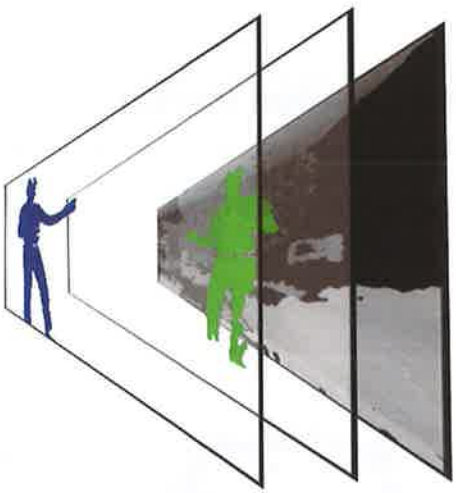


Abbildung 12.8 Mehrere Ebenen vor dem Welthintergrund.

Listing 12.3 zeigt, wie dies erreicht wird. Ein neues (leeres) **GreenfootImage** wird erzeugt und als Akteur-Bild festgelegt. Die Zeichenfarbe des Bildes wird mit einer zufälligen Farbe initialisiert, sodass die nächsten Maloperationen (wahrscheinlich) mit unterschiedlichen Farben für unterschiedliche Benutzer beginnen.

Listing 12.3: Der **Canvas**-Konstruktor.

```
public Canvas(int width, int height, UserData user)
{
    this.user = user;
    setImage(new GreenfootImage(width, height));
    getImage().setColor(randomColor());
}
```

Alles, was der Konstruktor sonst noch macht, ist, ein Objekt des Typs **UserData** in einem Feld namens **user** zu speichern. Dieses Objekt enthält Informationen

über den Benutzer, der dieser Ebene zugeordnet ist, und wir können es verwenden, um uns genauer über die Haltung des Benutzers zu informieren.

Übung 12.18 Der Konstruktor aus Listing 12.3 verwendet eine Methode namens **randomColor**. Diese Methode ist in derselben Klasse implementiert. Untersuche diese Methode und erkläre schriftlich, wie sie eine zufällige Farbe auswählt. Wie viele unterschiedliche mögliche Farben kann sie zurückgeben? Kannst du eine weitere Farbe hinzufügen?

Nun können wir mit dem Malen beginnen, indem wir Code in unsere **act**-Methode schreiben.

Malen mit der Hand

Das Erste, was wir machen wollen, ist, etwas Farbe auf unsere Leinwand – unsere Ebene – zu bringen, und zwar an der Stelle, an der sich die rechte Hand des Benutzers befindet. Dazu malen wir einen mittelgroßen Kreis auf das Bild unseres Akteurs:

```
getImage().fillOval(x, y, 20, 20);
```

In diesem Beispiel sind die beiden **20**-Parameter die Breite und die Höhe des gemalten Ovals, wodurch ein Kreis mit Durchmesser 20 entsteht. Die Parameter **x** und **y** müssen durch die Position der Hand des Benutzers ersetzt werden. Nun müssen wir uns nur noch erarbeiten, wie wir die Position der Hand des Benutzers auf dem Bildschirm in Erfahrung bringen.

Der allgemeine Weg, an Informationen über Benutzer zu kommen sieht so aus: Wir beginnen mit der Methode **getTrackedUsers** aus der **KinectWorld**-Klasse (Abbildung 12.9), diese liefert uns ein Feld aus **UserData**-Objekten mit einem Eintrag für jeden protokollierten Benutzer. Danach können wir mehrere Methoden des **UserData**-Objekts verwenden (nur einige davon sind in Abbildung 12.9 dargestellt), um an detaillierte Informationen über jeden Nutzer zu gelangen. Für uns ist im Moment die Methode **getJoint** am interessantesten.

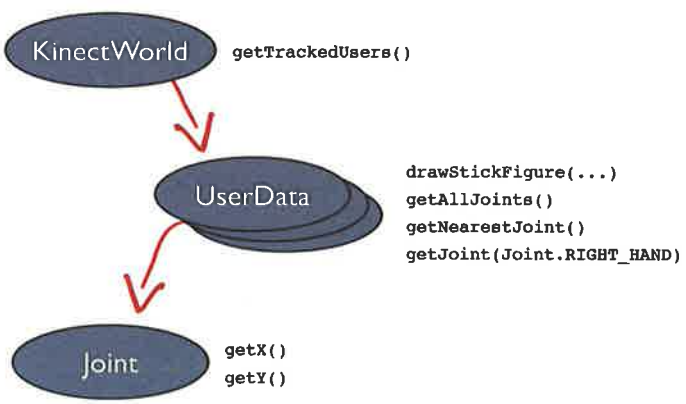


Abbildung 12.9
Informationen über Benutzer einholen.

getJoint gibt uns ein Objekt vom Typ **Joint** zurück, das wir nun verwenden können, um die *x*- und *y*-Koordinaten der Joint-Punkte zu erhalten.

Übung 12.19 Die Methode **getJoint** erwartet einen Parameter, der angibt, an welchen Joint-Punkten wir interessiert sind. Wir verwenden in der Regel hier Konstanten, die in der Klasse **Joint** definiert sind (Abbildung 12.9). Sieh dir die Dokumentation der Klasse **Joint** an, um festzustellen, welche Konstanten dort verfügbar sind. Wie viele unterschiedliche Joint-Punkte können wir darüber herausfinden?

In der **act**-Methode unserer **Canvas**-Klasse wurde das **UserData**-Objekt bereits geholt und im Feld **user** gespeichert. Wir können daher direkt dazu übergehen, den Joint-Punkt für die rechte Hand anzufordern:

```
Joint rightHand = user.getJoint(Joint.RIGHT_HAND);
```

Sobald wir den Joint-Punkt der rechten Hand besitzen, können wir dessen *x*- und *y*-Koordinaten extrahieren und diese verwenden, um unseren Kreis auf den Bildschirm zu malen:

```
getImage().fillOval(rightHand.getX(), rightHand.getY(), 20, 20);
```

Das ist alles, was du machen musst, um mit deiner Hand zu malen. Probiere es aus!

Übung 12.20 Füge den oben besprochenen Code in der **act**-Methode deiner **Canvas**-Klasse ein. Du benötigst nur diese zwei Zeilen, die hier gezeigt sind. Teste deinen Code!

Du solltest jetzt sehen, dass wir nun (ununterbrochen) an der Stelle malen, an der sich die rechte Hand befindet.

Das Malen beenden

Um unser Malen ein wenig mehr zu steuern, wäre es schön, wenn wir die rechte Hand auch mal bewegen könnten, ohne zu malen. Daher wollen wir unseren Code dahingehend ändern, dass er nur malt, wen die rechte Hand der nächstgelegene Joint-Punkt zur Kamera ist. Auf diese Weise können wir malen, indem wir unsere Hand nach vorne halten, und aufhören, indem wir sie zurückziehen.

Die Klasse **UserData** besitzt eine Methode namens **getNearestJoint**, die wir verwenden können, um dies zu erreichen. Versuche es zuerst selbst, bevor du weiterliest (wir werden das Vorgehen nach der Übung besprechen).

Übung 12.21 Verändere deinen Code so, dass nur gemalt wird, wenn die rechte Hand der nächstgelegene Joint-Punkt zur Kamera ist. Finde dazu den nächstgelegenen Joint-Punkt und prüfe, ob es sich dabei um die rechte Hand handelt.

Wenn du für diese Übung die Lösung selbst erarbeitet hast, dann wird du festgestellt haben, dass wir einfach eine **if**-Anweisung hinzufügen können, um den nächsten Joint-Punkt mit der Konstanten der rechten Hand zu vergleichen.

```
Joint rightHand = user.getJoint(Joint.RIGHT_HAND);
if (user.getNearestJoint() == Joint.RIGHT_HAND)
{
    getImage().fillOval(rightHand.getX(), rightHand.getY(),
    20, 20);
}
```

Wir wollen noch zwei weitere Verbesserungen einbringen – aber dieses Mal musst du den Code selbst schreiben.

Dein Bild löschen

Wenn wir irgendetwas gemalt haben, aber nicht ganz glücklich damit sind oder neu anfangen möchten, dann wäre es schön, wenn wir den Bildschirm (bzw. genauer das Bild des Akteurs) löschen könnten. Dies ist ganz leicht mithilfe der **clear**-Methode von **GreenfootImage** zu erreichen:

```
getImage().clear();
```

Jetzt möchten wir noch eine Funktion hinzufügen, mit der Benutzer den Bildschirm löschen können, indem sie ihre linke Hand über ihren Kopf heben. Das Prinzip ist ziemlich einfach: Wir holen uns die *y*-Koordinate der linken Hand und die des Kopfes und prüfen, welcher Wert größer ist.

Übung 12.22 Füge Code zu deiner **act**-Methode hinzu, mit der der Bildschirm gelöscht wird, wenn der Benutzer die linke Hand über den Kopf hebt.

Farben ändern

Die nächste Verbesserung ist vom Aufbau her recht ähnlich: die Malfarbe ändern, wenn der Benutzer seinen rechten Fuß hochhebt. (Das kannst du erkennen, wenn du testest, ob sich der rechte Fuß oberhalb des linken Knies befindet.) Die vorhandene Methode **randomColor** hilft dir, eine neue (zufällige) Farbe auszuwählen.

Übung 12.23 Füge weiteren Code hinzu, um die Malfarbe auf eine andere zufällige Farbe zu ändern, wenn der Benutzer den rechten Fuß hochhebt.

Du kannst dir vermutlich noch viele andere Dinge ausdenken, die man mit diesem Szenario machen kann. Probiere viele Ideen aus! Wenn du deinen Code mit unserem vergleichen möchtest: Den Code zu den besprochenen Übungen findest du in den Buchszenarien unter *body-paint*.

Der einzige Code aus diesem Szenario, den wir nicht besprochen haben, ist der Code in der Klasse **`PaintWorld`**. Diese Klasse verwaltet das Erzeugen der Ebenen und spätere Entfernen aus der Welt. Der komplizierteste Aspekt daran ist es nachzuvollziehen, wenn neue Benutzer während der Programmausführung erscheinen oder die bisherigen Nutzer weggehen – und immer entsprechende Ebenen zu erzeugen oder zu löschen. Wir werden uns dies hier nicht weiter ansehen, sondern dieses Thema stattdessen auf das nächste Beispiel verschieben und dort besprechen.

12.8 Ein einfaches Kinect-Spiel: Pong

Das letzte Projekt, das wir in diesem Kapitel besprechen möchten, ist ein sehr einfaches Spiel: *Pong*. Pong ist ein gut bekanntes, einfaches Spiel, bei dem wir Tennis mit einer Art Paddeln auf dem Bildschirm spielen. In unserem Fall werden wir zwei Paddel haben, für jede Hand eines, und gegen den Computer spielen, der ein Paddel steuert (Abbildung 12.10).

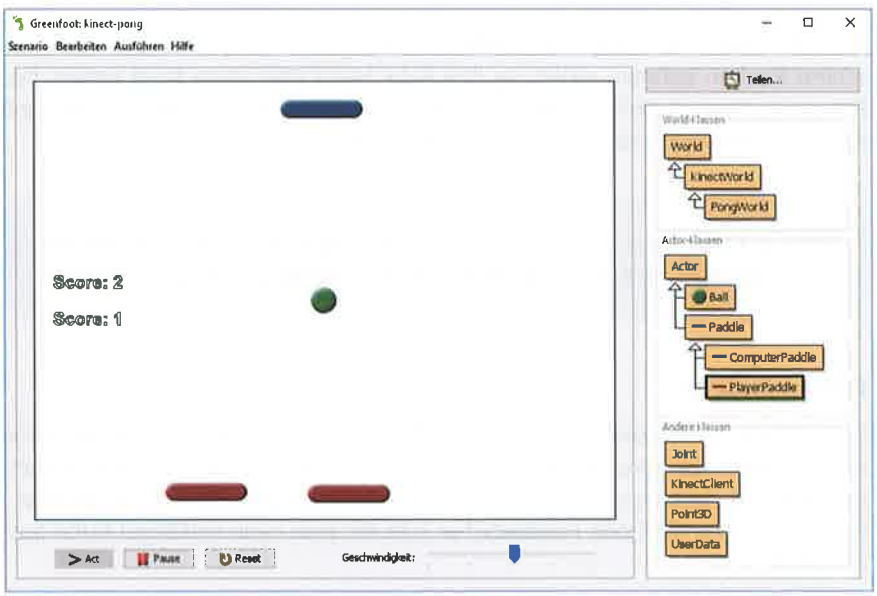


Abbildung 12.10 Ein einfaches Kinect-Spiel.

Wie so oft haben wir zwei Versionen des Szenarios in unserem Buchszenarienordner: Eine Version dient als Ausgangspunkt für unsere Übungen, die andere enthält die endgültige Lösung.

Übung 12.24 Öffne das Szenario *kinect-pong-start*. Finde heraus, wie es gespielt wird.

Wenn du das Anfangsszenario ausprobierst, so wirst du sehen, dass es ein sehr einfaches Spiel ist, ähnlich wie Tischtennis, das du mithilfe der Tastatur spielen kannst. Du steuerst einen Spieler, der Computer den anderen. (Du kannst herausfinden, welche Tasten zu verwenden sind, indem du den Quelltext liest oder es einfach ausprobierst.)

In diesem Szenario ist nichts enthalten, das wir an diesem Punkt nicht leicht verstehen könnten. Daher wollen wir uns mit den nächsten Übungen nur kurz durch den Code lesen, um uns mit dem Projekt vertraut zu machen. Beantworte die folgenden Fragen schriftlich.

Übung 12.25 Welche Tasten steuern das Paddel des Spielers?

Übung 12.26 Welche Methoden des Spieler-Paddels und des Computer-Paddels sind von **Paddle** geerbt?

Übung 12.27 Wenn sich das Paddel nach rechts oder links bewegt, um wie viele Schritte bewegt es sich in einem ACT-Schritt?

Übung 12.28 Was macht das Paddel des Computers, wenn sich kein Ball auf dem Bildschirm befindet? (Du kannst deine Antwort überprüfen, indem du den Ball interaktiv entfernst und dann das Szenario ausführst.)

Übung 12.29 Welche Strategie verwendet das Computer-Paddel, um zu entscheiden, wohin es sich bewegt?

Übung 12.30 Wie viele Objekte werden in die Welt eingebracht, wenn das Szenario initialisiert wird? Um welche Objekte handelt es sich dabei?

Übung 12.31 In welcher Klasse wird der Punktestand verwaltet?

Übung 12.32 Es gibt eine Verzögerung zwischen dem Start des Szenarios und dem Moment, an dem sich der Ball zu bewegen beginnt. Wie lang ist diese Verzögerung?

Übung 12.33 Erkläre, wie diese Verzögerung implementiert ist.

Wir sehen, dass dieses Szenario eine sehr einfache, geradlinige Implementierung besitzt. Um unsere Paddel mithilfe der Kinect zu steuern, werden wir den Code nun verändern.

Dieses Mal beginnen wir mit dem Teil, den wir bei unserer Diskussion im letzten Abschnitt ausgelassen hatten: Wir möchten mitbekommen, wenn ein Benutzer in das Sichtfeld der Kamera kommt.

Das Ziel sieht so aus: Solange kein Benutzer sichtbar ist, sollte das Spiel in einem Wartezustand sein. Tritt ein Benutzer in das Sichtfeld, dann beginnt das Spiel. Wenn kein Benutzer im Sichtfeld ist (d.h., der Benutzer ist weggegangen), dann hält das Spiel wieder an.

Benutzer entdecken

Wir haben vorhin gesehen, dass wir über die Methode **getTrackedUsers** ein Feld aller Nutzer bekommen können, die aktuell protokolliert werden:

```
UserData[] users = getTrackedUsers();
```

Wir können dann prüfen, ob irgendein Benutzer anwesend ist, indem wir testen, ob die Länge dieses Felds null ist:

```
if (users.length > 0)
{
    ...
}
```

(Zur Erinnerung: die Länge eines Feldes (**length**) ist tatsächlich ein öffentliches Feld, keine Methode. Es gibt keine Klammern nach dem Wort **length**. Dies sieht ungewöhnlich aus, aber so definiert Java es nun einmal.)

Übung 12.34 Erstelle in **PongWorld** eine neue private Methode mit Namen **startGame**. Diese Methode benötigt keine Parameter. Verschiebe das Erzeugen der Spieler-Paddel und des Balls in diese Methode, sodass der Spieler-Paddel und der Ball erst dann erzeugt werden, wenn das Spiel beginnt.

Übung 12.35 Teste das Spiel. Wenn du dein Szenario nun ausführst, sollte kein Spieler-Paddel und kein Ball sichtbar sein (weil wir die **startGame**-Methode noch nicht aufgerufen haben).

Übung 12.36 Prüfe mit deiner **act**-Methode, ob es protokollierte Benutzer gibt. Falls ja, dann starte das Spiel. Teste das, indem du das Szenario ausführst und dich vor die Kamera stellst. Das Spiel sollte dann anfangen.

Übung 12.37 Bei einer unbedarften Implementierung (bei der nur **startGame** aufgerufen wird, wenn die Länge des Feldes der protokollierten Benutzer größer als null ist) werden viele Paddel und Bälle erscheinen. Dies liegt daran, dass **startGame** dann in jedem ACT-Schritt aufgerufen wird, wodurch immer mehr Bälle und Paddel erzeugt werden. Stelle also sicher, dass nur ein Spieler-Paddel und ein Ball erzeugt werden. Zu diesem Zweck kannst du die boolesche Variable **idle** verwenden, die bereits in der Klasse definiert ist. Wenn das System im Wartezustand ist (*idle*) und du einen protokollierten Benutzer entdeckst, soll das Spiel gestartet werden. Es muss dann festgehalten werden, dass sich das Spiel nun nicht mehr im Wartezustand befindet.

Wenn du es geschafft hast, die Übungen bis hierher durchzuführen, dann sollte das Spiel nun am Anfang in einem Wartezustand sein (das Computer-Paddel bewegt sich während der Wartezeit langsam) und das Spieler-Paddel und der Ball werden erzeugt, wenn du den Bildschirm betrittst.

Jetzt wollen wir uns auch noch mit Spielern befassen, die das Spiel wieder verlassen.

Übung 12.38 Füge eine weitere private Methode namens **`stopGame`** hinzu. Entferne hierin das Spieler-Paddel und den Ball aus der Welt.

Übung 12.39 Erweitere deine **`act`**-Methode, sodass sie nun **`stopGame`** aufruft, wenn das Spiel nicht im Wartezustand ist, aber keine Spieler entdeckt werden. Das Spiel sollte dann wieder in den Wartezustand übergehen.

Dein Spiel sollte nun beginnen, sobald Spieler auftauchen, und beendet werden, wenn sie weggehen. Dir wird auffallen, dass das Spiel erst einige Sekunden, nachdem der Spieler weggegangen ist, anhält und nicht sofort. Wenn die Kinect die Spieler aus ihrem Sichtfeld verliert, versucht das Gerät noch eine Weile lang, sie wiederzuentdecken, bevor es aufgibt und beschließt, dass sie wirklich weg sind.

Das Paddel steuern

Als Nächstes wollen wir den eigentlichen Spielverlauf implementieren. Damit das Spieler-Paddel von einer Person gesteuert werden kann, müssen die Benutzerdaten an das Paddel übergeben werden. Da dies ein Spiel für einen einzelnen Spieler ist, können wir einfach mit dem ersten **`UserData`**-Objekt unseres Feldes der protokollierten Benutzer arbeiten. Falls mehrere Benutzer sichtbar sind, ignorieren wir die anderen einfach.

Wenn wir also die Benutzerdaten des ersten sichtbaren Nutzers unserer **`startGame`**-Methode übergeben wollen, können wir schreiben:

```
startGame(users[0]);
```

Übung 12.40 Füge einen Parameter des Typs **`UserData`** zu deiner **`startGame`**-Methode hinzu und übergib dieser Methode die Benutzerdaten des ersten sichtbaren Nutzers, wenn das Spiel anfängt.

Übung 12.41 Füge in den Konstruktor der Klasse **`PlayerPaddle`** einen ähnlichen Parameter ein und leite den Parameter, den **`startGame`** erhalten hat, an den **`PlayerPaddle`**-Konstruktor weiter.

Übung 12.42 Speichere in deiner Klasse **`PlayerPaddle`** die **`UserData`**, die der Konstruktor bekommen hat, in einem Feld **`user`**.

Dein Szenario sollte sich nun wieder kompilieren lassen, aber unsere Bearbeitung hat bisher noch keine sichtbaren Auswirkungen.

Jetzt bleibt für uns noch eines zu tun, und zwar die Position des Paddels anhand der Benutzerdaten auszurichten. Du kannst die Methode **`getJoint`** vom Benutzer-Objekt aufrufen, um den Joint-Punkt der rechten Hand zu erhalten. Dann wird **`setLocation`** für das Paddel aufgerufen, wobei die *x*-Koordinate der Hand und die *y*-Koordinate des Paddels als Parameter eingesetzt werden.

Übung 12.43 Entferne den Code aus der der **`act`**-Methode von **`PlayerPaddle`**. Füge stattdessen neuen Code ein, der den Joint-Punkt der rechten Hand des Benutzers erhält und dann die Position des Paddels mithilfe der *x*-Koordinate der Hand ausrichtet (die *y*-Koordinate bleibt unverändert).

Jetzt können wir spielen!

Letztendlich ist es ganz einfach, die Handposition des Spielers zu benutzen. Deine **`act`**-Methode sollte nur zwei oder drei Zeilen lang sein.

Bevor wir zum Ende kommen, wollen wir noch eine weitere Erweiterung vornehmen: mit beiden Händen spielen.

Übung 12.44 Die Hand in der Klasse **`PlayerPaddle`** soll nun variabel zu verwenden sein. Erzeuge ein Feld des Typs **`int`** mit Namen **`HAND`**. Initialisiere dieses Feld im Konstruktor mithilfe eines neuen zusätzlichen Konstruktorparameters. Verwende diese Variable in deiner **`act`**-Methode anstelle von **`Joint.RIGHT_HAND`**, um den Joint-Punkt anzugeben, um den es hier geht.

Übung 12.45 Füge **`Joint.RIGHT_HAND`** als zweiten Konstruktorparameter in der Klasse **`PongWorld`** an der Stelle ein, wo das Spieler-Paddel erzeugt wird.

Nun sind wir wieder dort, wo wir begonnen haben: Wir haben ein Paddel, das von unserer rechten Hand gesteuert wird. Doch jetzt können wir leicht angeben, welche Hand wir verwenden möchten, was uns in die Lage versetzt, ein weiteres Paddel für die linke Hand zu erzeugen.

Übung 12.46 Füge das Erzeugen eines zweiten Paddels an der Stelle hinzu, wo das Spieler-Paddel erzeugt wird.

Fertig! Wir können jetzt mit beiden Händen spielen. Wie immer, wenn du dir mit deinem Code etwas unsicher bist, kannst du dir das fertige Szenario *kinect-pong* im Buchszenarienordner anschauen.

12.9 Zusammenfassung

In diesem Kapitel haben wir Greenfoot an die Microsoft Kinect angeschlossen. Abgesehen davon, dass wir dadurch die Möglichkeit hatten, eine Reihe von wirklich interessanten Beispielen zu gestalten, haben wir gesehen, dass sich Programmierung mit Hardwaregeräten nicht groß von unseren bisherigen Erfahrungen unterscheidet. Wir verwenden dieselben Programmiertechniken – Methodenaufrufe, Variablen, Parameter, Typen – wie in früheren Kapiteln.

Wenn wir mit externen Geräten programmieren, haben wir in der Regel mit einer API zu tun – einer Programmierschnittstelle –, die aus einer Reihe von Klassen und Methoden besteht, um auf das Gerät zuzugreifen. Beim Einsatz eines solchen Geräts müssen wir in erster Linie lernen, wie diese API arbeitet, und verstehen, was wir mit ihr machen können. Danach benötigen wir nur noch unsere allgemeinen Java- und Programmierfähigkeiten, um Programme dafür zu schreiben.

Zusammenfassung der Konzepte

- **Skelettales Tracking** ist das Identifizieren und Protokollieren („Tracking") von einigen ausgesuchten Punkten des Körpers, wie Hände, Ellbogen, Kopf, Füße und so weiter.
- Kommunikation mit Hardwarekomponenten wird in der Regel durch eine **Treibersoftware** gesteuert.
- **Protokollierte Benutzer** sind Benutzer, die aktuell von der Kinect-Hardware erkannt und analysiert werden.

Vertiefende Aufgaben

Hier sind ein paar weitere Übungen, die du mit unserem letzten Beispiel, *kinect-pong*, durchführen kannst. Natürlich kannst du jedes der anderen Beispiele ebenfalls erweitern, doch wir überlassen es dir, Ideen dafür zu entwickeln.

Übung 12.47 Statte dein Pong-Spiel mit Soundeffekten aus, und zwar mindestens einen für die folgenden Ereignisse: wenn der Ball das Paddel trifft; wenn der Ball ins Aus kommt; wenn das Spiel beginnt.

Übung 12.48 Ändere den Hintergrund.

Übung 12.49 Rufe einen Gewinner aus (mit einer Nachricht und entsprechendem Sound), wenn ein Spieler 8 Punkte erreicht.

Übung 12.50 Auch das obere Paddel soll durch einen Spieler gesteuert werden.

Übung 12.51 Gib auch dem zweiten Spieler zwei Paddel.

Übung 12.52 Führe eine Hindernis ein: Jeder Spieler kann seine gesamte Seite für ein paar Sekunden blockieren, indem er den linken Fuß hochhebt. Das soll aber nur dreimal in einem Spiel möglich sein.

Übung 12.53 Wenn ein Spieler beide Hände über seinen Kopf hebt, dann wird der Ball schneller.

Der Ordner mit den Beispielen enthält noch ein weiteres Szenario. Dies ist in erster Linie als Demonstration gedacht, wie eine Cartoon-Figur anstelle des Strichmännchens gezeichnet wird. Unsere Figur heißt *Fred* und das Szenario *fred-with-radio*.

Wir werden das Szenario hier nicht besprechen, sondern es ganz dir zum Ausprobieren überlassen. Du möchtest vielleicht die beiden Sounddateien gegen eigene Songs austauschen – dann kannst du deine beiden Lieblingssongs abspielen, indem du das Radio mit deiner rechten oder linken Hand berührst.