Ein kleine Rahmenbibliothek für Spiele im 2-dimensionalen Raum

Sven Eric Panitz
Hochschule Rhein-Main

Zum Abschluss des Moduls »Objektorientierte Softwareentwicklung« steht ein kleines Spieleprojekt, das in Einzelarbeit zu lösen ist.

Dieses Papier stellt zusammen mit der Projektaufgabe eine kleine Rahmenbibliothek vor, unter deren Benutzung das Spieleprojekt zu realisieren ist. Einige Klassen aus den bisherigen Übungsblättern finden in dieser Bibliothek Verwendung. Die Rahmenbibliothek erlaubt es, Spiele unabhängig für die GUI-Plattformen Java-Swing, JavaFX, als Browserspiel mit Hilfe von GWT und als Android Applikation zu entwickeln.

1 Das Spieleframework

Die Bibliothek erlaubt es Spiele zu entwickeln, die auf einer zweidimensionalen Spielfläche laufen. Es gibt Spielfiguren, die sich selbstständig bewegen, Spielfiguren, die durch Benutzereingaben gesteuert werden können und Spielelemente, die unbewegt das Spielfeld definieren.

1.1 Schnittstellen für Spielobjekte

Wir stellen zunächt drei Schnittstellen vor, die ausdrücken, dass Spielfiguren Bewegungsschritte durchführen können und dass Spielfiguren sich grafisch darstellen lassen.

1.1.1 Allgemeine Klassen und Schnittstellen

Für die Bewegung ist die Schnittstelle `Moveable` zuständig:

```
1 package name.panitz.game.framework;
2 interface Movable{
3   void move();
4 }
```

Listing 1: src/name/panitz/game/framework/Moveable.java

Die Schnittstelle `Paintable` drückt aus, dass ein Objekt sich grafisch darstellen kann:

```
1 package name.panitz.game.framework;
2 public interface Paintable<I>{
3   void paintTo(GraphicsTool<I> g);
4 }
```

Listing 2: src/name/panitz/game/framework/Paintable.java

Diese Schnittstelle ist generisch gehalten über den Typ der Bildobjekte.

Anders als im Verlauf der Vorlesung bekommt die Methode `paintMeTo` der Schnittstelle als Parameter kein Objekt der Java FX Klasse `javafx.scene.canvas.GraphicsContext` übergeben, sondern ein Objekt der allgemeiner konzipierten Schnittstelle `GraphicsTool`. Diese Schnittstelle wird in mehreren Implementierungen für die verschiedenen GUI-Frameworks vorliegen. Da die unterschiedlichen GUI Frameworks ganz unterschiedliche Klassen für die Bereitstellung von Bilddateien haben, halten wir in dieser Schnittstelle den eigentlichen Typ für Bilddateien variable. Daher ist die Schnittstelle generisch. Sie hat eine Typvariable `I`, die für den Typ der konkreten Bilddateien steht. Ansonsten hat die Schnittstelle die üblichen Methoden, um Vierecke und Kreise zu zeichnen und eine Methode um Bilddateien zu zeichnen.

```java
package name.panitz.game.framework;

public interface GraphicsTool<I> {
  void drawImage(I img,double x,double y);
  void drawRect(double x,double y,double w,double h);
  void fillRect(double x,double y,double w,double h);
  void drawOval(double x,double y,double w,double h);
  void fillOval(double x,double y,double w,double h);
  void drawLine(double x1,double y1,double x2,double y2);

  void setColor(double red, double green, double blue);
  void drawString(double x,double y,int fntsize, String fntName,
    String text);

  default void drawString(double x,double y,int fontSize, String text)
    {
    drawString(x,y,fontSize,"Helvetica",text);
  }

  default void drawString(double x,double y,String text){
    drawString(x,y,20,"Helvetica",text);
  }

  default void drawCircle(double x,double y,double w){
    drawOval(x, y, w, w);
  }
  default void fillCircle(double x,double y,double w){
    fillOval(x, y, w, w);
  }

  I generateImage(String name, GameObject<I> go);
}
```

Listing 3: src/name/panitz/game/framework/GraphicsTool.java

Zur Darstellung von Punkten im zweidimensionalen Raum wurden bereits auf einem Übungsblatt die Klasse `Vertex` entwickelt.

```java
package name.panitz.game.framework;

public class Vertex {
  public double x;
  public double y;

  public Vertex(double x, double y) {
    this.x = x;
    this.y = y;
  }

  @Override
  public String toString() {
    return "("+x+", "+y+")";
```

```
15    }
16
17    public void move(Vertex that){
18      x += that.x;
19      y += that.y;
20    }
21
22    public void moveTo(Vertex that){
23      x = that.x;
24      y = that.y;
25    }
26
27    public Vertex mult(double d) {
28      return new Vertex(d*x,d*y);
29    }
30  }
```

Listing 4: src/name/panitz/game/framework/Vertex.java

1.1.2 Abstrakte Spieleobjekte

Als zentrale Klasse für unsere Spielobjekte wurde im Verlauf der Vorlesung die Klasse `GeometricObject` entwickelt. In diesem Framework ist statt der Klasse `GeometricObject` eine Schnittstelle `GameObject` vorgegeben. Diese Schnittstelle gibt vor, dass es Getter- und Setter-Methoden, für die vier Felder, die aus der Klasse `GeometricObject` bekannt sind, gibt. Die aus `GeometricObject` bekannten Methoden, die mit den vier Feldern der Klasse arbeiten, finden sich in dieser Schnittstelle als default-Methoden wieder.

Zusätzlich enthält die Schnittstelle eine Methode, die es erlauben soll, Bildobjekte zu erzeugen. Aus diesem Grund ist auch diese Schnittstelle generisch über den Typ der Bilddateien.

```
1  package name.panitz.game.framework;
2
3  public interface GameObject<I> extends Movable, Paintable<I> {
4    double getWidth();
5    double getHeight();
6    void setWidth(double w);
7    void setHeight(double h);
8
9    Vertex getPos();
10
11   Vertex getVelocity();
12   void setVelocity(Vertex v);
13
14
15   default boolean isLeftOf(GameObject<?> that) {
16     return this.getPos().x + this.getWidth() < that.getPos().x;
17   }
```

```
18
19    default boolean isRightOf(GameObject<?> that) {
20      return that.isLeftOf(this);
21    }
22
23
24    default boolean isAbove(GameObject<?> that) {
25      return this.getPos().y + this.getHeight() < that.getPos().y;
26    }
27
28    default boolean touches(GameObject<?> that) {
29      if (this.isLeftOf(that)) return false;
30      if (that.isLeftOf(this)) return false;
31      if (this.isAbove(that))  return false;
32      if (that.isAbove(this))  return false;
33      return true;
34    }
35
36   default boolean isStandingOnTopOf(GameObject<?> that) {
37      return !(isLeftOf(that) || isRightOf(that)) && isAbove(that)
38              && getPos().y + getHeight() + 2 > that.getPos().y;
39    }
40
41
42    default void move() {
43      getPos().move(getVelocity());
44    }
45    default double size(){return getWidth() * getHeight();}
46    default boolean isLargerThan(GameObject<?> that) {
47      return size() > that.size();
48    }
49  }
```

Listing 5: src/name/panitz/game/framework/GameObject.java

Zum Generieren der Bildobjekte werden die drei Frameworks eine spezielle Klasse benötigen. Auch hierfür wird eine Schnittstelle vorgesehen. Sie sieht eine Methode vor, die ein Bildobjekt für den Dateinamen einer Bilddatei erzeugt.

Die Schnittstelle `GameObject` wird in den meisten Eigenschaften bereits in einer abstrakten Klasse umgesetzt.

```
1  package name.panitz.game.framework;
2
3  public abstract class AbstractGameObject<I> implements GameObject<I>{
4    double width;
5    double height;
6    Vertex position;
7    Vertex velocity;
8
9    public AbstractGameObject(double width, double height, Vertex
       position,Vertex velocity) {
10     this.width = width;
```

```
11      this.height = height;
12      this.position = position;
13      this.velocity = velocity;
14    }
15
16    public AbstractGameObject(double width, double height, Vertex
        position) {
17      this(width,height,position,new Vertex(0,0));
18    }
19
20    public AbstractGameObject(double width, double height) {
21      this(width,height,new Vertex(0,0));
22    }
23
24    public AbstractGameObject(double width) {
25      this(width,width);
26    }
27
28    @Override
29    public double getWidth() {
30      return width;
31    }
32
33    @Override
34    public double getHeight() {
35      return height;
36    }
37
38    @Override
39    public Vertex getPos() {
40      return position;
41    }
42
43    @Override
44    public void setWidth(double w) {
45      width = w;
46    }
47
48    @Override
49    public void setHeight(double h) {
50      height = h;
51    }
52
53    @Override
54    public Vertex getVelocity() {
55      return velocity;
56    }
57
58    @Override
59    public void setVelocity(Vertex v) {
60      velocity = v;
61    }
```

```
62 }
```

Listing 6: src/name/panitz/game/framework/AbstractGameObject.java

1.1.3 Konkrete Spielobjekte

Wir geben in diesem Abschnitt zwei konkrete Klassen, die Spieleobjekte realisieren. Diese können direkt bei der Entwicklung eines Spieles verwendet werden, oder durch Erweiterungen in Unterklassen spezialisiert werden.

Bildobjekte Die folgende Klasse basiert auf einer Bilddatei. Es ist dafür zu sorgen, dass für die unterschiedlichen Realisierungen, auf denen die Spielebibliothek portiert ist, die Bilddateien in den korrekten Ordner gelegt wird. Näheres findet sich hierzu in den Beschreibungen der Implementierungen. Bei der swing- und der fx-Realisierung muss dieser Ordner im Klassenpfad des fertigen Spiels liegen, d.h. dort, wo auch die `.class`-Dateien liegen.

Die Dimensionen in Weite und Höhe des Spielobjektes wird durch die Größe der Bilddatei bestimmt.

```
1  package name.panitz.game.framework;
2
3  public class ImageObject<I> extends AbstractGameObject<I>{
4    String imageFileName;
5    I img;
6    private boolean changed = true;
7
8    public ImageObject(String imageFileName) {
9      super(0,0);
10     this.imageFileName = imageFileName;
11   }
12
13   public ImageObject(String imageFileName, Vertex pos, Vertex motion)
       {
14     super(0, 0, pos, motion);
15     this.imageFileName = imageFileName;
16   }
17   public ImageObject(String imageFileName, Vertex position) {
18     super(0, 0, position);
19     this.imageFileName = imageFileName;
20   }
21
22   public ImageObject(String imageFileName, double width) {
23     super(width);
24     this.imageFileName = imageFileName;
25   }
26
27   public String getImageFileName() {
28     return imageFileName;
```

```
29    }
30
31    public void setImageFileName(String imageFileName) {
32      this.imageFileName = imageFileName;
33      changed  = true;
34    }
35
36    @Override
37    public void paintTo(GraphicsTool<I> g) {
38      if (changed) {
39    img = g.generateImage(imageFileName,this);
40    changed = false;
41      }
42      if (null!=img) g.drawImage(img, getPos().x, getPos().y);
43    }
44  }
```

Listing 7: src/name/panitz/game/framework/ImageObject.java

Bildobjekte mit Schwerkraft In vielen Spielen gibt es Spielfiguren, die fallen und springen. Hier brauchen wir ein wenig Ahnung über die Schwerkraft. Eine Unterklasse von `ImageObject` realsiert bereits Bildobjekte, die die Bewegung auf Sprünge implementiert hat.

```
1  package name.panitz.game.framework;
2
3  public class FallingImage<I> extends ImageObject<I> {
4    static double G = 9.81;
5    double v0;
6    int t = 0;
7
8    public boolean isJumping = false;
9
10   public FallingImage(String imageFileName, Vertex corner, Vertex
       movement) {
11     super(imageFileName, corner, movement);
12   }
13
14   public void stop() {
15     getPos().move(getVelocity().mult(-1.1));
16     getVelocity().x = 0;
17     getVelocity().y = 0;
18     isJumping = false;
19   }
20
21   public void restart() {
22     double oldX = getVelocity().x;
23     getPos().move(getVelocity().mult(-1.1));
24     getVelocity().x = -oldX;
25     getVelocity().y = 0;
26     isJumping = false;
```

```
27    }
28
29
30    public void left() {
31      if (!isJumping) {
32        getVelocity().x = -1;
33      }
34    }
35
36    public void right() {
37      if (!isJumping) {
38        getVelocity().x = +1;
39      }
40    }
41
42    public void jump() {
43      if (!isJumping) {
44        startJump(-3.5);
45      }
46    }
47
48    public void startJump(double v0) {
49      isJumping = true;
50      this.v0 = v0;
51      t = 0;
52    }
53
54    @Override
55    public void move() {
56      if (isJumping) {
57        t++;
58        double v = v0 + G * t / 200;
59        getVelocity().y = v;
60      }
61      super.move();
62    }
63  }
```

Listing 8: src/name/panitz/game/framework/FallingImage.java

Textobjekte Die zweite Klasse, die für konkrete Spielobjekte direkt verwendet werden kann, repräsentiert Objekte, die sich durch einen Text darstellen. Diese sind für feste textanzeigen im Spiel geeignet.

```
1  package name.panitz.game.framework;
2  public class TextObject<I> extends AbstractGameObject<I>{
3    public String text;
4    public String fontName;
5    public int fontSize;
6
```

```
7     public TextObject(Vertex position, String text, String fntName,int
       fntSize) {
8       super(0,0,position);
9       this.text = text ;
10      this.fontName = fntName;
11      this.fontSize = fntSize;
12    }
13
14    @Override
15    public void paintTo(GraphicsTool<I> g) {
16      g.drawString(getPos().x,getPos().y,fontSize,fontName,text);
17    }
18  }
```

Listing 9: src/name/panitz/game/framework/TextObject.java

1.2 Das allgemeine Framework

In diesem Abschnitt folgen die Klassen, die die Eigenschaften eines Spiels beschreiben. Auch hier beginnen wir mit der Definition einer Schnittstelle.

1.2.1 Zentrale Spielschnittstelle

Die Schnittstelle `GameFramework` beschreibt, was ein Spiel für einen generellen Aufbau hat. Wie alle anderen Klassen ist auch diese generisch gehalten über den Typ der Bilddateien.

```
1  package name.panitz.game.framework;
2
3  import java.util.List;
4
5  public interface GameLogic<I,S> extends Movable, Paintable<I> {
6    List<List<? extends GameObject<I>>> getGOss();
7    GameObject<I> getPlayer();
8    List<Button> getButtons();
9
10   void playSound(SoundObject<S> so);
11   List<SoundObject<S>> getSoundsToPlayOnce();
12
13   double getWidth();
14   double getHeight();
15
16   void doChecks();
17
18   boolean isStopped();
19
20   default public void paintTo(GraphicsTool<I> g){
21     for (List<? extends GameObject<I>> gos:getGOss())
```

```
22        for (GameObject<I> go:gos)
23          go.paintTo(g);
24      getPlayer().paintTo(g);
25    }
26
27    default void move(){
28      if (!isStopped()){
29        for (List<? extends GameObject<I>> gos:getGOss())
30          for (GameObject<I> go:gos)
31            go.move();
32        getPlayer().move();
33      }
34    }
35
36    default void playSounds(SoundTool<S> soundTool){
37      for (SoundObject<S> so:getSoundsToPlayOnce()){
38        so.playSound(soundTool);
39      }
40      getSoundsToPlayOnce().clear();
41    }
42
43    void keyPressedReaction(KeyCode keycode);
44    void keyReleasedReaction(KeyCode keycode);
45
46    void start();
47    void pause();
48  }
```

Listing 10: src/name/panitz/game/framework/GameLogic.java

Wie man sieht, wird verlangt, dass das Spiel sich selbst wieder komplett zeichnen lassen kann, dadurch dass die Schnittstelle `Paintable` erweitert wird.

Wir gehen davon aus, dass ein Spiel genau einen Spieler hat, der durch den Benutzer über die Tastatur gesteuert wird. Dieser Spieler kann von einem Spiel per `getPlayer` erfragt werden.

Desweiteren gehen wir davon aus, dass es mehrere inhaltlich unterschiedliche Spielobjekte gibt. Diese werden in eigenen Listen gehalten. Alle Spielobjekte sind schließlich in einer Liste von Listen gesammelt. Diese kann mit der Methode `getOGoss()` erfragt werden.

Ein Spiel kann auch nach seiner Höhe und Breite gefragt werden können.

Die Methode `doChecks` soll einen einzigen Spielschritt des Spiels durchführen. Es wird einen Ticker geben und pro Tick wird das Spiel um einen Schritt weiterbewegt. Jedes bewegliches Spielobjekt nimmt dann genau einen Schritt mit seiner Bewegung vor. Dieses realisiert die Methode `move()`.

Nach jedem Schritt finden Checks für das Spiel statt. Hier wird auf Kollisionen getestet und eventuell auf Grund einer Kollision der Spielzustand verändert, indem eventuell Spielobjekte gelöscht werden, oder Punktestände verändert werden. Hier befindet sich zumeist die komplette Spiellogik umgesetzt.

Das Spiel soll angeben, ob es noch aktiv ist, oder gestoppt wurde. Gestoppte Spiele werden nicht mehr bewegt.

Da sicher alle Spiele intern ein Feld für den Spieler und ein Feld für die Liste von Listen von Spielobjekten benötigt, implementieren wir jetzt eine abstrakte Klasse, die quasi ein leeres Spiel repräsentiert.

```
1  package name.panitz.game.framework;
2
3  import java.util.List;
4  import java.util.ArrayList;
5
6  public abstract class AbstractGame<I,S> implements GameLogic<I,S> {
7    protected List<List<? extends GameObject<I>>> goss = new ArrayList
       <>();
8    protected List<Button> buttons = new ArrayList<>();
9    protected List<SoundObject<S>> soundsToPlays= new ArrayList<>();
10   protected GameObject<I> player;
11   double width;
12   double height;
13
14   public AbstractGame(GameObject<I> player, double width, double
       height) {
15     this.player = player;
16     this.height = height;
17     this.width = width;
18   }
19
20   @Override
21   public List<List<? extends GameObject<I>>> getGOss(){
22     return goss;
23   }
24
25   @Override
26   public GameObject<I> getPlayer() {
27     return player;
28   }
29
30   @Override
31   public List<Button> getButtons(){
32     return buttons;
33   }
34
35   public List<SoundObject<S>> getSoundsToPlayOnce() {
36     return soundsToPlays;
37   }
38
39   @Override
40   public double getWidth() {
41     return width;
42   }
43
44   @Override
```

```
45    public double getHeight() {
46      return height;
47    }
48
49    @Override
50    public void playSound(SoundObject<S> so) {
51      getSoundsToPlayOnce().add(so);
52    }
53
54    private boolean isStopped = false;
55
56    @Override
57    public boolean isStopped() {
58      return isStopped;
59    }
60    @Override
61    public void start() {
62      isStopped  = false;
63    }
64
65    @Override
66    public void pause() {
67      isStopped  = true;
68
69    }
70
71    @Override
72    public void keyPressedReaction(KeyCode keycode)[]
73    @Override
74    public void keyReleasedReaction(KeyCode keycode)[]
75
76  }
```

Listing 11: src/name/panitz/game/framework/AbstractGame.java

Spiele können nun als Unterklasse dieser abstrakten Klassen umgesetzt werden.

1.2.2 Klänge

Die kleine Bibliothek soll auch ermöglichen kleine Klänge bei Spielereignissen abzuspielen. Hierzu ist eine kleine Klasse vorgesehen, die im Konstruktor den Dateinamen eines Klangdatei übergeben bekommt. Da die verschiedenen Umsetzungen der Bibliothek mit unterschiedlichen Klassn für Klänge arbeiten, ist die Klasse generisch gehalten und hat einen Typparameter `S` für sound.

```
1  package name.panitz.game.framework;
2
3  public class SoundObject<S> {
4    String fileName;
5    S sound;
6    public SoundObject(String fileName) {
```

```
7       super();
8       this.fileName = fileName;
9     }
10    public void playSound(SoundTool<S> st){
11      try{
12        if (sound==null) sound = st.loadSound(fileName);
13
14        st.playSound(sound);
15        sound = null;
16      }catch(Exception e){System.err.println(e);}
17    }
18
19  }
```

Listing 12: src/name/panitz/game/framework/SoundObject.java

Die allgemeine Spielschnittstelle enthält eine Methode

`List<SoundObject<S>> getSoundsToPlayOnce();.`

Objekte die dieser Liste zugefügt werden, werden bei nächster Gelegenheit im Spiel genau einmal abgespielt.

Ähnlich wie wir eine Schnittstelle für das Zeichnen von Objekten vorgesehen haben, stellt die Rahmenbibliothek eine Schnittstelle für das Laden und Abspielen von klängen bereit. Diese Schnittstelle wird in den unterschiedlichen Umsetzungen jeweils spezifisch implementiert, wobei die Swing-Umsetzung und die javafx-Umsetzung dort auf dieselbe Implementierung zurückgreifen können.

```
1  package name.panitz.game.framework;
2
3  public interface SoundTool<S> {
4    S loadSound(String fileName);
5    void playSound(S sound);
6  }
```

Listing 13: src/name/panitz/game/framework/SoundTool.java

1.2.3 Knöpfe

Globale Knöpfe können für ein Spiel definiert werden. Diese können zu einem Neustart führen, oder das Spiel anhalten, aber wenn es gewünscht ist, auch ins Spielgeschehen eingreifen. Die entsprechende Klasse ist sehr simpel gehalten. Sie enthält eine Zeichenkette für die beschriftung des Knopfes und ein Objekt der funktionalen Schnittstelle `Runnable` für die Aktion, die bei einem Knopfdruck ausgeführt werden soll.

```
1  package name.panitz.game.framework;
2
3  public class Button {
4    public String name;
5    public Runnable action;
6    public Button(String name, Runnable action) {
7      this.name = name;
8      this.action = action;
9    }
10 }
```

Listing 14: src/name/panitz/game/framework/Button.java

Die allgemeine Spielschnittstelle enthält eine Methode

`List<Button> getButtons();.`

Objekte die bei Spielstart in dieser Liste sind, werden als Knöpfe dargestellt.

1.2.4 Tastaturbehandlung

Die Schnittstelle `GameLogic` enthält die Methode `keyPressedReaction(KeyCode keycode)`. In dieser ist zu implementieren, wie auf Tastaturereignisse (in diesem Fall das Drücken einer Taste) reagiert wird. Hierzu ist eine Aufzählungsklasse für Tastatur-Codes gegeben. In dieser sind (noch nicht alle) Tasten als Aufzählungswert versehen. Im Beispielspiel des übernächsten Abschnitts kann die Anwendung dieser Aufzählungsklasse gesehen werden. Des Weiteren ist die Methode `keyReleasedReaction(KeyCode keycode)` enthalten. Mit dieser Methode kann auf das loslassen eines Tastendrucks reagiert werden.

```
1  package name.panitz.game.framework;
2  import java.util.Map;
3  import java.util.HashMap;
4  import java.util.EnumSet;
5
6  public enum KeyCode{
7    LEFT_ARROW(37),
8    RIGHT_ARROW(39),
9    DOWN_ARROW(40),
10   UP_ARROW(38),
11   VK_A('A'),
12   VK_B('B'),
13   VK_C('C'),
14   VK_D('D'),
15   VK_E('E'),
16   VK_F('F'),
```

```
17    VK_G('G'),
18    VK_H('H'),
19    VK_I('I'),
20    VK_J('J'),
21    VK_K('K'),
22    VK_L('L'),
23    VK_M('M'),
24    VK_N('N'),
25    VK_O('O'),
26    VK_P('P'),
27    VK_Q('Q'),
28    VK_R('R'),
29    VK_S('S'),
30    VK_T('T'),
31    VK_U('U'),
32    VK_V('V'),
33    VK_W('W'),
34    VK_X('X'),
35    VK_Y('Y'),
36    VK_Z('Z'),
37    VK_SPACE(' ');
38
39    private static final Map<Integer, KeyCode> LOOKUP
40      = new HashMap<Integer,KeyCode>();
41
42    static {
43      for(KeyCode s : EnumSet.allOf(KeyCode.class))
44        LOOKUP.put(s.getCode(), s);
45    }
46
47    private int code;
48    private KeyCode(int code){
49      this.code = code;
50    }
51
52    public int getCode() {
53      return code;
54    }
55
56    public static KeyCode fromCode(int code) {
57      return LOOKUP.get(code);
58    }
59  }
```

Listing 15: src/name/panitz/game/framework/KeyCode.java

2 Kleines Beispiel: Hexe und Bienenstiche

In diesem Kapitel wird ein minimales Beispiel gezeigt, wie mit Hilfe der im ersten Kapitel vorgestellten Bibliothek ein kleines Spiel erzeugt werden kann.

2.1 Definition des Beispielspiels

Als Beispiel für die Anwendung der Bibliothek sei hier ein minimales Spiel angegeben. Spielfigur ist eine Hexe, die auf einem Besen über eine Wiese fliegt. Gegner sind Biene. Berührt die Hexe eine Biene, so wird sie gestochen. Nach 10 Stichen ist das Spiel beendet.

Als Bilddateien für die Hexe werden Zeichnungen von Andreas und Martin Textor verwendet, die im WS06/07 für ein Spiel in der damaligen Vorlesung Java des 3. Semesters im Diplomstudiengang entstanden sind. Das entsprechende Spiel und die Bilddateien sind offiziell auf

`https://github.com/atextor/witchcraft/tree/master/gfxsrc`

erhältlich.

Das Hintergrundbild der Wiese wurde von folgender Quelle bezogen:

`https://pixabay.com/de/wiese-himmel-landschaft-natur-gras-970333/`

Wir implementieren das Spiel durch Erwweitern der abstrakten Klasse `AbstractGame`. Es werden Listen für verschiedene Spielobjekte angelegt. Eine Liste für Hintergrundbilder, eine Liste für die Gegner und eine Liste für Wolken, die keinen Einfluss auf das Spiel haben, sondern nur für die Athmosphäre sind.

Ein Text-Objekt für die Spielstandsanzeige wird erzeugt.

Ein weiteres Feld zählt, wieviel Bienen die Hexe bereits gestochen haben.

Im Konstruktor wird zunächst der Oberklasse mit dem Aufruf von `super` die Spielfigur übergeben. Hierfür wird ein Bildobjekt mit dem Bild der Hexe erzeugt.

Dann werden die Liste mit den Gegnerobjekten, die Liste mit den Wolken und die Liste mit den Hintergrundobjekten mit Spielobjekte gefüllt.

Zu guter letzt werden die einzelnen Listen der Liste von Listen `goss` hinzugefügt.

Es bleiben drei Methoden, die zu überschreiben sind:

- `doChecks`: hier wird geprüft, ob Bienen und Wolken links außerhalb des Spielfelds sind. Ist dieses der Fall, so werden sie wieder nach rechts gesetzt, so dass sie erneut über das Spielfeld fliegen. Desweiteren wird geprüft, ob eine Biene die Hexe berührt. Wenn ja, wird die Anzahl der Stiche um eins erhöht und die Biene aus dem Spielfeld gesetzt. Die Anzeige wird entsprechend verändert. Der zehnte Stich führt dazu, dass eine Meldung über das Spielende angezeigt wird.
- `isStopped`: hier wird ab dem 10. Bienenstich `true` zurück gegeben.
- `keyPressedReaction`: die Pfeiltasten verändern die Bewegungsrichtung der Hexe.

```
1 package name.panitz.game.example.simple;
2
3 import java.util.ArrayList;
```

```
4  import java.util.List;
5
6  import name.panitz.game.framework.AbstractGame;
7  import name.panitz.game.framework.Button;
8  import name.panitz.game.framework.KeyCode;
9  import name.panitz.game.framework.SoundObject;
10 import name.panitz.game.framework.GameObject;
11 import name.panitz.game.framework.ImageObject;
12 import name.panitz.game.framework.TextObject;
13 import name.panitz.game.framework.Vertex;
14
15 public class SimpleGame<I,S> extends AbstractGame<I,S>{
16   List<GameObject<I>> hintergrund = new ArrayList<>();
17   List<GameObject<I>> gegner = new ArrayList<>();
18   List<ImageObject<I>> wolken = new ArrayList<>();
19
20   int stiche = 0;
21
22   TextObject<I> infoText
23     = new TextObject<>(new Vertex(30,35)
24        ,"Kleines Beispielspiel","Helvetica",28);
25
26   SoundObject<S> outch = new SoundObject<>("outch.wav");
27
28   public SimpleGame() {
29     super(new ImageObject<>("hexe.png"
30         ,new Vertex(200,200),new Vertex(1,1)),800,600);
31
32     hintergrund.add(new ImageObject<>("wiese.jpg"));
33     hintergrund.add(infoText);
34
35     wolken.add(new ImageObject<>("wolke.png"
36         ,new Vertex(800,10),new Vertex(-1,0)));
37     wolken.add(new ImageObject<>("wolke.png"
38         ,new Vertex(880,90),new Vertex(-1.2,0)));
39     wolken.add(new ImageObject<>("wolke.png"
40         ,new Vertex(1080,60),new Vertex(-1.1,0)));
41     wolken.add(new ImageObject<>("wolke.png"
42         ,new Vertex(980,110),new Vertex(-0.9,0)));
43
44     gegner.add(new ImageObject<>("biene.png"
45         ,new Vertex(800,100),new Vertex(-1,0)));
46     gegner.add(new ImageObject<>("biene.png"
47         ,new Vertex(800,300),new Vertex(-1.5,0)));
48
49     getGOss().add(hintergrund);
50     getGOss().add(wolken);
51     goss.add(gegner);
52
53     getButtons().add(new Button("Pause", ()-> pause()));
54     getButtons().add(new Button("Start", ()-> start()));
55     getButtons().add(new Button("Exit", ()-> System.exit(0)));
```

```
56    ]
57
58    @Override
59    public void doChecks() {
60      for (GameObject<I> g:wolken){
61        if (g.getPos().x+g.getWidth()<0) {
62          g.getPos().x = getWidth();
63        }
64      }
65
66      for (GameObject<I> g:gegner){
67        if (g.getPos().x+g.getWidth()<0) {
68          g.getPos().x = getWidth();
69        }
70        if (player.touches(g)){
71          stiche++;
72          infoText.text = "Bienenstiche: "+stiche;
73          g.getPos().moveTo(new Vertex(getWidth()+10,g.getPos().y));
74          playSound(outch);
75        }
76      }
77      if (stiche>=10){
78        gegner.add(new TextObject<>(new Vertex(100,300)
79            ,"Du hast verloren","Helvetica",56));
80      }
81    }
82
83    @Override
84    public boolean isStopped() {
85      return super.isStopped() || stiche>=10;
86    }
87
88    @Override
89    public void keyPressedReaction(KeyCode keycode) {
90      if (keycode!=null)
91       switch (keycode){
92        case RIGHT_ARROW:
93          getPlayer().getVelocity().move(new Vertex(1,0));
94          break;
95        case LEFT_ARROW:
96          getPlayer().getVelocity().move(new Vertex(-1,0));
97          break;
98        case DOWN_ARROW:
99          getPlayer().getVelocity().move(new Vertex(0,1));
100         break;
101       case UP_ARROW:
102         getPlayer().getVelocity().move(new Vertex(0,-1));
103         break;
104       default: ;
105      }
106   }
107 }
```

Listing 16: src/name/panitz/game/example/simple/SimpleGame.java

Damit ist das kleine Spiel bereits fertig implementiert und kann jetzt auf allen drei Plattformen gespielt werden.

2.2 Swing Instanz des Spieles

Um das Spiel in Swing zu spielen, muss es einer Instanz der Klasse `SwingScreen` übergeben werden. Dieses ist dann ein `JPanel`, der in einem `JFrame` angezeigt werden kann.

```
1 package name.panitz.game.example.simple;
2 import name.panitz.game.framework.GameLogic;
3 import name.panitz.game.framework.swing.SwingGame;
4
5 public class PlaySwing{
6   public static void main(String[] args){
7     SwingGame.startGame(new SimpleGame<>());
8   }
9 }
```

Listing 17: PlaySwing.java

2.3 Java FX Instanz des Spieles

Um das Spiel in JavaFx zu starten, ist eine Unterklasse der Klasse `GameApplication` zu schreiben. Im Aufruf des Superkonstruktors ist eine Spielinstanz zu übergeben. Die Main Methode ermöglicht das Starten des Spiels als exportierte .jar Datei.

```
1  package name.panitz.game.example.simple;
2
3  import name.panitz.game.framework.fx.GameApplication;
4
5  public class PlayFX extends GameApplication {
6    public PlayFX(){
7      super(new SimpleGame<>());
8    }
9    public static void main(String[] args) {
10     PlayFX.launch();
11   }
12 }
```

Listing 18: PlayFX.java

2.4 Einstiegsklasse für GWT

Um das Spiel als Browserspiel zu spielen, ist eine Implementierung der Schnittstelle `EntryPoint` zu schreiben. In der Methode `onModuleLoad` ist eine Instanz des Spiels in einem `GameSrceen`-Objekt einem HTML-Knoten der Webseite, auf dem das Spiel laufen soll, hinzuzufügen.

```
1  package name.panitz.game.framework.gwt.client;
2
3  import name.panitz.game.example.simple.SimpleGame;
4
5  import com.google.gwt.core.client.EntryPoint;
6  import com.google.gwt.user.client.ui.RootPanel;
7
8  public class WebGames implements EntryPoint {
9    public void onModuleLoad() {
10     RootPanel.get("spiel").add(new GameScreen(new SimpleGame<>()));
11   }
12 }
```

Listing 19: WebGames.java

2.5 Einstiegsklasse für Android

Um ein Spiel in Android zu spielen, ist eine Klasse `MainActivity` zu schreiben. Wenn diese von der Klasse `GameActivity` ableitet, so kann die Instanz des zu spielenden Spiels im super-Aufruf des Konstruktors übergeben werden.

```
1  package name.panitz.game.framework.gwt.client;
2  package name.panitz.game;
3  import name.panitz.game.example.simple.SimpleGame;
4
5  import name.panitz.game.framework.android.GameActivity;
6
7  public class MainActivity extends GameActivity {
8    public MainActivity() {
9      super(new SimpleGame<>());
10   }
11 }
```

Listing 20: MainActivity.java