Prof. Dr. T. Thormählen, F. Nitzsche
Fachbereich Mathematik und Informatik
Hans-Meerwein-Strasse, Lahnberge
D-35032 Marburg

Philipps Universität Marburg

Programmierpraktikum: 2D-Platformer-Game (SoSe 2025)

Tag 1

Schritt 1 *Laden und Anzeigen eines Bildes*

Wir besprechen zu Beginn ein kurzes Beispiel, in dem wir zeigen, wie Sie die paint-Methode eines JFrames überschreiben können, um eigene Grafikelement darstellen zu können. Der Quellcode für dieses Beispiel wird in der Datei `Step0.zip` bereit gestellt.

Das Beispiel aus `Step0.zip` kann nach dem Enpacken in die Intellij IDE importiert werden. Dazu den folgenden Menüpunkt auswählen:

`File` → `New` → `Project From Existing Sources`

Dann den Ordner `Step0` auswählen und in den folgenden Dialogen immer die Standardauswahl bestätigen.

Erweitern Sie nun das vorhandene Projekt und lesen Sie ein Bitmap-Bild (.BMP) ein. Verwenden Sie dazu z.B. die Java-Klassen BufferedImage und ImageIO.

Erweitern Sie die überschriebene paint-Methode des JFrame und zeichnen Sie das geladene Bild mittels Graphics2D::drawImage.

Hinweis: In Ihrer paint-Methode sollten Sie nicht `super.paint()` aufrufen, da dies zu Fehlern führen kann.

Wenn Sie einen Schritt abgeschlossen haben, erhalten Sie jeweils das Passwort für die Musterlösung von Ihrer Tutorin bzw. Ihrem Tutor.

Schritt 2 *Level aus Kacheln erstellen*

In der Musterlösung für Schritt 1 (Step1.zip) gibt es eine Sammlung von Kachelbildern, die jeweils 70 x 70 Pixel groß sind, und aus denen das Level zusammengesetzt werden soll. Um das Level zu definieren, wird ein farbcodiertes Level-Bild geladen, wobei eine bestimmte Farbe eines Pixels im Level-Bild die entsprechende Kachel im Ausgabebild erzeugt.

Abbildung 1: Beispiel eines zu ladenden Level-Bildes mit 50 x 5 Pixeln

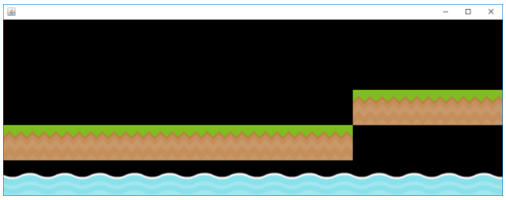

*Abbildung 2: 1000 Pixel breiter Ausschnitt aus dem resultierenden Ausgabebild mit insgesamt 50 * 70 x 5 * 70 Pixeln*

Beispiele für Kacheln, aus denen das Level aufgebaut ist:

Erstellen Sie dazu zunächst zwei Klassen:

- `Level`:
  Hier soll das Level-Bild `level1.bmp` und die Kacheln geladen werden und zu einem Level zusammengesetzt werden. Schreiben Sie hierfür eine Funktion, welche ein großes Ausgabebild gemäß Abbildung 2 aus dem geladenen Level-Bild aus Abbildung 1 und den Kacheln erstellt. Das Ausgabebild muss nicht in eine Datei gespeichert werden (außer eventuell zu Debug-Zwecken).

  Benutzen Sie dazu zunächst folgende Zuordnungen:
  `Color.BLUE == liquidWaterTop_mid.png`
  `Color.BLACK == grassMid.png`

  Hilfestellung:
  Ein leeres Bild kann erzeugt werden mit:
  `new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);`
  siehe Constructor BufferedImage.

  Die Funktion `BufferedImage.getRGB()` gibt ein 32-Bit Integer zurück.
  Die Alpha-Rot-Grün-Blau Farbkanäle des Pixels sind in einem 32-Bit-Integer-Wert codiert: Alpha in Bits 24-31, Rot in Bits 16-23, Grün in Bits 8-15 und Blau in Bits 0-7, siehe auch https://stackoverflow.com/questions/25761438/understanding-bufferedimage-getrgb-output-values

  Ein solcher BufferedImage-Integer-Pixel-Wert kann mit folgender Funktion in eine Java-Color umgewandelt werden:
  `Color color = new Color(levelImg.getRGB(x, y));`
  Siehe auch: https://docs.oracle.com/javase/7/docs/api/java/awt/Color.html#Color(int)

- `Platformer:`
  Diese Klasse soll einen Ausschnitt des Ausgabebildes anzeigen. Erstellen Sie dazu eine Klasse Platformer, die von JFrame abgeleitet ist. Adaptieren Sie anschießend Ihre Lösung aus Schritt 1, so dass nun in der paint-Methode immer nur ein Bildausschnitt des Ausgabebildes der Level-Klasse gemalt wird, der 5∗70 = 350 Pixel hoch und 1000 Pixel breit ist.

  Fügen Sie einen KeyAdapter zum Abfangen von Tasteneingaben hinzu, um den angezeigten Bildausschnitt zu verschieben. Dabei soll die linke Pfeiltaste einen Bildlauf des Ausschnitts nach links bewirken und die rechte Pfeiltaste einen Bildlauf nach rechts.

Schritt 3 *Level und Spielfigur*

- Nun generieren Sie sich Ihr eigenes Level. Erstellen Sie dazu zunächst ein Level-Bild in einem Bildbearbeitungsprogramm, wie etwa GIMP.
  Legen Sie ein neues Bild mit einer Breite von etwa 100 – 200 Pixeln und einer Höhe von etwa 8 – 12 Pixeln an. Jetzt zoomen Sie nah an die Zeichenfläche heran und malen mit einem 1 Pixel breiten Stift (Werkzeug) mit zunächst zwei Farben „Wasser“ blau (0,0,255) und „Boden“ schwarz (0,0,0). Beginnen Sie mit einem simplen Aufbau. Sie können das Bild jederzeit später weiter verfeinern oder verändern.

  `Player`:
  Erstellen Sie eine Klasse `Player`. Die Spielfigur hat nicht nur ein Bild, sondern besitzt eine Reihe von Bildern damit eine Laufanimation realisiert werden kann. Laden Sie alle Spielfigur-Bilder/Kacheln `p1_walk*.png` in `Player`. Realisieren Sie eine Laufanimation mit Hilfe der Player-Kacheln, sowie eine Funktion `move()`, die das Player-Objekt im Fenster laufen lässt.

Einige Kacheln der Spielfigur bzw. ihrer Animation:

- Folgende Variablen sollte die Klasse `Player` zunächst haben:

  1. Position in X- und Y-Richtung
  2. Position aus dem letzten Bild in X- und Y-Richtung
  3. Eine Geschwindigkeit X- und Y-Richtung

1. Um ihre Spielfigur provisorisch zu platzieren, erstellen Sie eine `draw()`-Methode in Ihrer Klasse `Platformer`, so dass zunächst der sichtbare Ausschnitt vom Level aus dem Level-Bild der Klasse Level ausgeschnitten wird und dann darüber das Bild der Spielfigur gemalt wird.

   Hier beispielhaft ein Aufbau der `draw()`-Methode:

```
private void draw(Graphics2D g2) {
 BufferedImage img_level = (BufferedImage) level.getImage();

 BufferedImage visibleLevel =
      level.getSubimage((int) level.pos.x, 0, 1000, level.getHeight());

 g2.drawImage(visibleLevel, 0, 0, this);

 g2.drawImage(
      player.getImage(),
      (int) getPlayer().pos.x,
      (int) getPlayer().pos.y, this);
}
```

2. Die `paint()`-Methode von JFrame unterstützt kein DoubleBuffering, was jedoch zur flackerfreien Darstellung von Spielen benötigt wird. Überschreiben Sie die ursprüngliche `paint()`-Methode der Klasse `Platformer` mit dem folgenden Code, in dem Ihre `draw()`-Methode aus 1. aufgerufen wird:

```
@Override
public void paint(Graphics g) {
 Graphics2D g2 = null;
 try {
  g2 = (Graphics2D) bufferStrategy.getDrawGraphics();
  draw(g2);

 } finally {
   g2.dispose();
 }
 bufferStrategy.show();
}
```

Außerdem müssen der Klasse `Platformer` folgende Zeilen hinzugefügt werden:

```
BufferStrategy bufferStrategy;

public Platformer() {
 ...
 createBufferStrategy(2);
 bufferStrategy = this.getBufferStrategy();
}
```

3. Stellen Sie jetzt sicher, dass Sie Ihre Spielfigur in der Welt in x- und y-Richtung frei steuern können und die Laufanimation bei Bewegung dargestellt wird. Passen Sie dazu den KeyAdapter entsprechend an. Außerdem soll vom Level immer nur ein Ausschnitt zu sehen sein, der sich daran orientiert, wo sich die Spielfigur bezüglich des gesamten Levels befindet. Denken Sie daran, nach jeder Bewegung `repaint()` aufzurufen.

4. Wenn Sie noch Zeit und Lust haben, verfeinern Sie Ihr Level mit neuen Kacheln und malen Sie Ihr Level-Bild weiter. Hier können Sie kreativ sein und sich auch gerne selbst bestimmte Kachel-Arten hinzufügen.

Tag 2

Schritt 4 Kollisionen erkennen

- `BoundingBox`:
  Um Kollisionen zwischen Objekten im Spiel zu verarbeiten, schreiben Sie zunächst eine Klasse `BoundingBox`. Diese hat als Member-Variablen vier Floatwerte `min.x`, `min.y`, `max.x`, `max.y` sowie die Methoden `intersect()` und `overlapSize()`:

```
public boolean intersect(BoundingBox b) {
 return  (min.x <= b.max.x) &&
         (max.x >= b.min.x) &&
         (min.y <= b.max.y) &&
         (max.y >= b.min.y);
}

public Vec2 overlapSize(BoundingBox b) {
   Vec2 result = new Vec2(0, 0);

   // X-dimension
   if (min.x < b.min.x)
    result.x = max.x - b.min.x;
   else
    result.x = b.max.x - min.x;

   // Y-dimension
   if (min.y < b.min.y)
    result.y = max.y - b.min.y;
   else
    result.y = b.max.y - min.y;

   return result;
}
```

- `Tile`:
  Erstellen Sie die Klasse `Tile`, die eine Kachel repräsentiert. Die Klasse soll jeweils eine Instanz von `BoundingBox` enthalten und sich selbst darstellen können (z.B. durch einen Index in einen Vektor von geladenen Kachelbildern).
- Ebenso erhält die Klasse `Player` eine BoundingBox.
- Schreiben Sie nun eine Funktion `checkCollision()`, welche zunächst Kollisionen zwischen Spielfigur und allen Kacheln erkennt. Dabei gibt es potentiell vier Kollisionsszenarien: Von oben, von unten, von links und von rechts. Geben Sie die Art der Kollision in der Konsole aus.

Schritt 5 Auf Kollisionen reagieren

Nun soll auf die verschiedenen Kollisionsarten entsprechend regaiert werden Fügen Sie dazu in der Klasse Player eine Methode `update()` hinzu und passen Sie in der Klasse `Platformer` die Funktion `checkCollision()` an.

Außerdem brauchen Sie in der Klasse `Platformer` einen Timer mit TimerTask, der periodisch alle 10 ms aufgerufen wird und `level.update()`, `player.update()`, `checkCollision()` und `repaint()` aufruft.

In der Methode `player.update()` soll nun Schwerkraft und Luftreibung auf die Spielfigur angewendet werden und sichergestellt werden, dass die Spielfigur das Level nicht verlässt.

Da jetzt ein Spielzustand periodisch aktualisiert wird, soll die Spielfigur nun über die drei Zustände

```
boolean jumping, walkingLeft, walkingRight;
```

gesteuert werden. Diese Zustände müssen bei ensprechendem Tastendruck aktiviert und beim Loslassen der Taste wieder deaktiviert werden. In der Methode `player.update()` muss entsprechend der Zustände die Position der Spielfigur aktualisiert werden.

In der Methode `checkCollision()` soll bei Überlappung der Boundingboxen von Spielfigur und Level entsprechend der Art der Kollision die Spielfigur wieder aus dem Kollisionsbereich geschoben werden.

Schritt 6 Sound hinzufügen

Fügen Sie zur Klasse `Player` eine Methode zum Abspielen von Sounds hinzu:

```
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;

public void playSound(String path){
    File lol = new File(path);

    try{
        Clip clip = AudioSystem.getClip();
        clip.open(AudioSystem.getAudioInputStream(lol));
        clip.start();
    } catch (Exception e){
        e.printStackTrace();
    }
}
```

Starten Sie die Hintergrund-Musik `"Sound/soundtrack.wav"` zu Beginn des Spiels. Außerdem soll bei jedem Sprung des Spielers der Sound `"Sound/jump1.wav"` abgespielt werden.

Tag 3

Bestes-Spiel-Wettbewerb

Basierend auf den Grundgerüst, das wir in den vergangenen zwei Tagen aufgebaut haben, können Sie heute das Spiel zu Ihrem eigenen Spiel machen, indem Sie eigene Objekte, Funktionsweisen und Ideen verwirklichen. Die bereitgestellten Assets (Bilder, Sounds, usw.) haben eine CC0 Lizenz (Public Domain), d.h. Sie können Ihr fertiges Spiel später frei verwenden, wenn Sie sich auf diese Assets beschränken. Im Rahmen des Wettbewerbs können aber natürlich auch Assets aus anderen Quellen benutzt werden, um in der gegebenen Zeit ein möglichst gutes Ergebnis zu erreichen.

Am Ende des heutigen Tags haben Sie die Möglichkeit, Ihr Spiel vorzustellen. Das beste Spiel gewinnt einen Pokal.

Mehr Spieleprogrammierung

Die AG Grafik und Multimedia bietet im kommenden Wintersemester das Modul Grafikprogrammierung an, in dem die Grundlagen zur Erstellung von interaktiven 3D-Computergrafik-Anwendungen vermittelt werden. Außerdem ist wieder ein Game Studio geplant, welches als Modul CS 502 Berufsvorbereitung oder im MarSkills Bereich belegt werden kann. Das Game Studio wird Anfang März 2025 als 2-wöchige Blockveranstaltung jeweils von 10:00 bis 16:00 Uhr stattfinden.