2D Spiele programmieren in Java

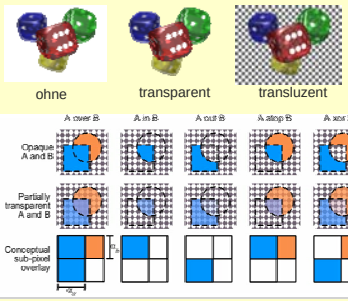
Teil 3: Bilder, Animationen

Dr. Katja Wegner
Dr. Ursula Rost

Bilder

- Sind vielfach in Spielen einsetzbar (Menüs, Hintergrund, Objekte)
- Eigenschaften:
  - Typ (e.g. bitmap, vector)
  - Größe (Höhe und Breite)
  - Farbtiefe (8/16/24 bit, RGB, etc.)
  - Alpha channel (e.g. RGBA)

Alpha channel



Formate

- Viele Formate vorhanden
- Am meisten benutzt:

Format	Compression	Colour depth	Transparency	Translucency
bmp	None/Poor	1,2,4,...,32	Separate file	Separate file
jpg	Lossy/Good	8,12,24	No	No
gif	Poor(ish)	1,2,3,4,...,8	Yes	No
png	Lossless, Good	1,2,4,...,64	Yes	Yes

Empfehlung

- JPEG für Hintergrundbilder
- PNG für alle anderen Bilder
- Verschiedene Optionen für Kompressionen ausprobieren, um die Bildgröße klein zu halten und trotzdem ein relativ gute Bildqualität zu bekommen

Optimierung

- Bilder einmal laden und wieder verwenden
- Verwaltung der Bilder notwendig
- Grafische Darstellung eines Objekts von Position trennen

Bilder laden

- BufferedImage:
  - Unterklasse von Image
  - Schneller Zugriff auf Bilddaten zur Manipulation (Farbmodel, Raster)

```
try {
   BufferedImage img= ImageIO.read(getClass().getResource("spider.png"));
}
catch (IOException e) {
   System.err.println("Failed to load: "+ filename);
   System.exit(0);
}
```

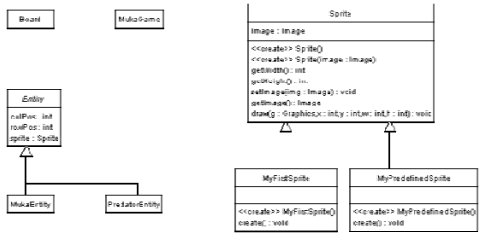
Zeichnen: graphics.drawImage(img, x, y, ImageObserver)

Bilder + Kompatibilität

```
import java.awt.*;

GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
GraphicsConfiguration gc = ge.getDefaultScreenDevice().getDefaultConfiguration();
BufferedImage image = null;
try {
        BufferedImage loadedImage = ImageIO.read( … );
        // Image kompatibel zur verwendet Grafikkonfiguration machen
        image = gc.createCompatibleImage(
                loadedImage.getWidth(), loadedImage.getHeight(),
                loadedImage.getColorModel().getTransparency() );
        Graphics2D g2d = image.createGraphics();
        g2d.drawImage( loadedImage, 0, 0, null );
        g2d.dispose();
}
catch( IOException e )  { //… }
```

Umsetzung

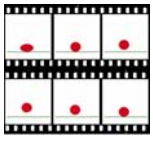


Übung 3.1

- Implementieren Sie diese Klassenstruktur.
- Sprite:
  - Verwaltet eine Bilddatei
- MyFirstSprite:
  - liest eine PNG Datei in ein BufferedImage Objekt
- MyPredefinedSprite:
  - schreibt die Grafikanweisungen der draw() Methode eines Entities in ein BufferedImage, dass dann an die Superklasse übergeben wird

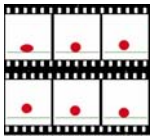
Animationen

- Bildfolge mit einer bestimmten Frequenz
  - **FPS** – Frames per second
  - Wie viele sind normal?
    - 100 FPS
    - 50
    - 30
    - 15
    - 5



Animationen

- Bildfolge mit einer bestimmten Frequenz
  - **FPS** – Frames per second
  - Wie viele sind normal?
    - 100 FPS
    - 50
    - 30
    - 20
    - 5
  - TV: 25/30 FPS → **48**(2x)/**72**(jedes Bild 3x auf Bildschirm projektiert)
  - Mindestens 24 FPS (Richtwert: Bildschirmfrequenz: 85 Hz → 80-85 FPS)



Animationsschleife

- Phase 1: update Bilderpositionen
- Phase 2: zeichnen der Bilder auf neuen Positionen
- Phase 3: schlafen

Übung 3.2

- Ersetzen Sie das Bild des Gegenspielers bei jedem Durchlauf der game loop durch ein anderes Bild, indem Sie vier Bilder einlesen und nach einander anzeigen. Experimentieren Sie mit verschieden Wartezeiten (sleep()).

FPS und sleep()

- Problem:
  - Je schneller der PC, um so schneller ist das Neuzeichnen beendet → Spiel läuft unterschiedlich schnell auf unterschiedlichen PCs
- Lösung:
  - Individuelle Wartezeiten entsprechend der Zeit einführen, die für das Zeichnen verwendet wird

Übung 3.3

- Passen Sie die game loop entsprechend an.
  - Abfrage Systemzeit:
    - `System.currentTimeMillis();`

Hinweis:

- Pause zwischen Bildern 42ms, um 24 FPS zu erreichen
- Problem unter **Windows 95/98** :
  - `System.currentTimeMillis()`
    update nur alle 50/60ms
  - Also nur 20 FPS möglich oder eigenen/andere Timer verwenden!!! (Buch: “Killer game programming” ab Seite 24)

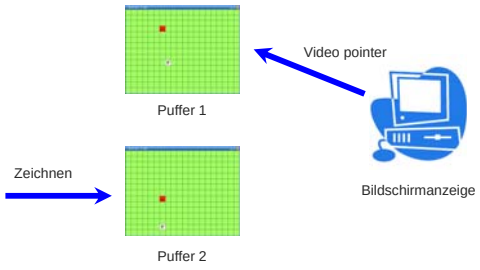
FPS und UPS

- UPS: Updates pro Sekunde
- Problem:
  - Zeichnen dauert sehr lange bei vielen Objekten → FPS zu niedrig (Spiel ruckelt)
- Lösung:
  - mehrere Updates (z.B. Bewegungen) in game loop durchführen und danach nur einmal neu zeichnen

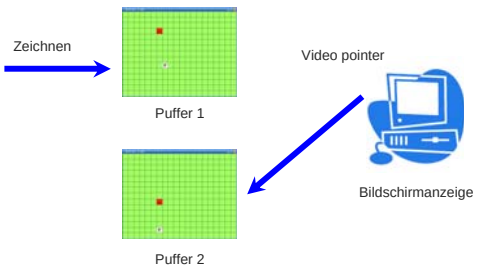
Double Buffering

- Problem:
  - Je mehr Grafiken/Objekte gezeichnet werden müssen, könnte es zum Flackern des Bildschirms kommen, da das Zeichnen direkt auf den Bildschirm länger dauert.
- Lösung:
  - Ein Puffer zum Zeichnen im Hintergrund und einer zum Anzeigen

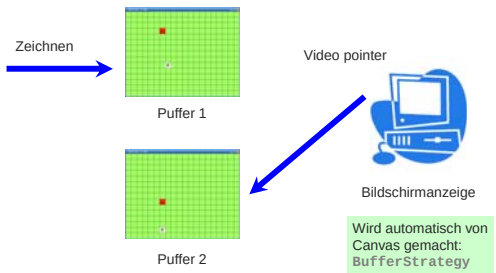
Double Buffering (2)



Double Buffering (3)



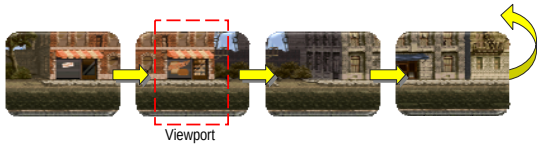
Double Buffering (3)



Wird automatisch von Canvas gemacht: `BufferStrategy`

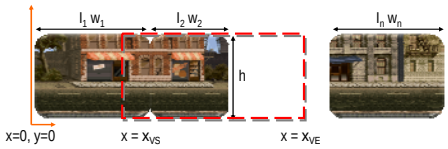
Laufender Hintergrund

- Folge von verbundenen Bildern
- z.B. für „jump and run“ Spiele

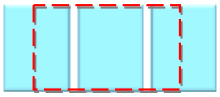


Umsetzung

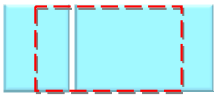
- N images **(I1 … In)** mit derselben Höhe, **h**, aber unterschiedlichen Breiten **(w1 … wn)**
- Das erste Bild, **I1**, startet an position x = 0



Umsetzung 2



Fall 1: Bild ist komplett im viewport



Fall 2: Bild ist nicht komplett im viewport