06/2024 · Noah Büchold · Liam Stedman
https://github.com/SpaceWAGdev/space-invaders

Space Invaders

Das Projekt ist in Java unter Verwendung der Greenfoot-Umgebung und dazugehörigen Bibliotheken implementiert.

Die klassische Greenfoot-Projektstruktur wurde übernommen. Das Projekt ist folgendendrmaßen strukturiert: `Enemy`, `Player`, `Projectile`, `MyWorld`

MyWorld.java

Nach dem Setup called Greenfoot zuerst den Konstruktor der Weltklasse:

```
public class MyWorld extends World {

public MyWorld()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);
        this.addObject(new Player(), 600 / 2, 340);
        this.setupEnemies();

    }
  }
```

Dieser Konstruktor called `super()` mit den Argumenten für die Größe und Skalierung der Welt, fügt ein neues `Player`-Objekt in die Welt ein, und called seine Methode `setupEnemies()`:

```
public void setupEnemies() {
        int alpha = 50;
        for (int i = 1; i < 12; i++) {
            for (int j = 1; j < 5; j ++) {
                this.addObject(new Enemy(), i * alpha, j * alpha);
        } }
    }
```

Dieser durchläuft zwei Schleifen, die ein zweidimensionales Gitter durchschreiten. Jede Koordinate des Gitter wird mit α skaliert. Um den Abstand zu verändern, würde Alpha angepasst werden.

Player.java

Der default-Konstruktor wird nicht überschrieben. Greenfoot called bei jedem Tick die `act`-Methode auf jedem registrierten `Actor`:

```
  public void act()
  {
      this.setRotation(-90);
```

Die Rotation wird nach Erstellung ersetzt, weil `addObject()` keine Rotation als Parameter nimmt.

```
      if (Greenfoot.isKeyDown("A") ){
        setLocation(this.getX() - 10, this.getY());
      }

      else if (Greenfoot.isKeyDown("D")) {
        setLocation(this.getX() + 10, this.getY());
      }
```

Wenn A oder D gedrückt ist, wird die Position auf die alte Position + 10 Einheiten gesetzt.

```
public static boolean canFire = true;

if (Greenfoot.isKeyDown("Space") && canFire) {
          getWorld().addObject(new Projectile(), getX(), getY() + 5);
          canFire = false;
          Timer t = new java.util.Timer();
          t.schedule( new java.util.TimerTask() {
          @Override
          public void run() {
              Player.canFire = true;
              t.cancel();} }, 100 );
      }
```

Wenn die Leertaste gedrückt wird und `canFire` wahr ist, wird ein neues `Projectile` an der Position des Spielers mit einem Offset von 5 Einheiten nach unten instantiiert. `canFire` wird falsch gesetzt, und ein neues Timer-Objekt `t` erstellt. Auf `t` wird die `schedule`-Methode gecalled, die ein `TimerTask` nimmt und es nach, in diesem Fall, 100ms ausführt.

Der `TimerTask` wird mit einem `@Override` für seine `run`-Methode ausgestattet, der die canFire auf dem Spieler zurücksetzt. `canFire` muss `static` sein, weil sonst eine Referenz zum aktiven `Player`-Objekt an den `TimerTask` weitergegeben werden müsste.

```
if(0 == getWorld().getObjects(Enemy.class).size()) {
          Greenfoot.stop();
}
```

Falls die Anzahl an Gegnern in der Welt 0 ist, wird das Spiel beendet.

Projectile.java

```
public void act()
  {
     setLocation(getX(), getY() - 5);
     Actor enemy = getOneIntersectingObject(Enemy.class);
     if (enemy != null) {
          getWorld().removeObject(enemy);
          getWorld().removeObject(this);

      }
  }
```

Die Position wird um 5 Einheiten pro Tick nach oben erweitert. Das Objekt vom Typ Enemy, das sich gerade auf der gleichen Position befindet wie das Projektil, wird abgespeichert. Falls es nicht `null` ist, wird es aus der Welt entfernt, anschließend zerstört sich das Projektil selbst.