Schritt 4: Klasse Ball

Erwartete Endergebnisse

- Die Klasse Ball ist in einer eigenen Datei abgespeichert.
- main.py erzeugt einen Ball
- Ball bewegt sich nach Start des Spieles durch Drücken der Leertaste in zufälliger Anfangsrichtung und prallt an allen Rändern nach dem Prinzip Einfallswinkel = Ausfallswinkel ab.
- Das UML ist entsprechend der Spezifikationen der Klasse Ball ergänzt.

Spezifikation der Klasse Ball

1. Ein Ball bewegt sich bei jedem Aufruf von act() um **eine** Einheit (move).
2. Jeder Ball bekommt vier Attribute: minX, maxX, minY und maxY. Diese legen die Abprall-Grenzen eines Balles fest. Die Abprall-Grenzen sind abhängig von der Spielfeldgröße.
3. Ein Ball prallt ab am oberen und unteren Rand und am linken und rechten Rand nach dem Prinzip
   a) Einfallswinkel = Ausfallswinkel. **Hinweis**: Es muss mit Directions (vgl. Schritt 1) gearbeitet werden! Erst wenn a) sicher läuft, die Ergänzung b) einbauen!
   b) bei jedem Abprallen (vgl. a) wird ein zufälliger Wert der Richtung (Directions!) hinzugefügt oder abgezogen. Dies führt dazu, dass die Flugbahn des Balls beim Abprallen leicht verändert wird. Hier mit eigenen Zufallsbereichen experimentieren. Welche Intervalle unterstützen das Spielerlebnis?
   Es macht sicherlich wenig Sinn, so große Werte zu wählen, dass der Ball am oberen Spielfeldrand seine Richtung komplett umkehrt.
   Intervallgrenzen möglichst parametrisieren, falls bei späteren (eigenen) Erweiterungen diese in die mögliche Wahl eines Schwierigkeitsgrades mit einfließen sollen (vgl. KONSTANTEN der Fenstergröße). b) bleibt in der Endversion von Schritt04 und ist somit in allen weiteren Versionen enthalten.
4. Noch reagiert der Ball nicht auf den Schläger!

Spezifikation eines Ball-Objektes in der main.py

1. Positionieren des Balles in der Mitte des Spielfeldes
2. Bei 10ms Simulationszyklus hat der Ball einen Verlangsamungsfaktor (*setSlowDown(Faktor)*) von 3, die Schläger von 4.

Hinweise

Dadurch, dass im **Konstruktor** des Balles die vier Attribute gefordert werden, **müssen** bei der Erzeugung eines Ball-Objektes (Instanziierung in der main) diese Attribute belegt werden. Damit erreichen wir, dass jedem Ball-Objekt (falls wir später noch weitere brauchen/erzeugen wollen) diese Werte zur Verfügung stehen.

Exkurs: Größenänderung eines Actor-Objekts

In der Dokumentation der API für das GameGrid[1] finden sich nur die wichtigsten Bestandteile der Actor-Klasse. Erst in der Original-Dokumentation[2] der zu Grunde liegenden JGameGrid-Bibliothek lassen sich weitere Methoden (z.B. für die Actor-Klasse) finden. Hier wird exemplarisch gezeigt, wie nach der Erzeugung eines Actor dessen Größe einmalig verändert werden kann. Dies ist nur einmalig im Konstruktor möglich und nicht dynamisch während der Laufzeit! Die Einsatzmöglichkeiten sind vielfältig: es braucht nur ein Grundobjekt erstellt werden. Dieses kann im weiteren Verlauf der Entwicklung beliebig skaliert werden. Oder es werden für verschiedene Schwierigkeitsstufen, verschiedene Größen benötigt. Zum Beispiel: kleinere Schläger erschweren das Treffen des Balles, oder ein größerer Ball erleichtert das Treffen, etc..

```
 9 # ---------------- class Ball ----------------
10 class Ball(Actor):
11     def __init__(self,minX, maxX, minY, maxY ):
12 
13         dateiname = "images/ball.jpg" # Dateiname MIT Unterordner
14         aktuellerPfad = os.path.abspath(".") # der Pfad zum Bild wird (intern) gebraucht und hier vom System ausgelesen
15         # Beispiel für die einmalige(!) Grössenänderung von Bildobjekten bei der Instanziierung
16         ballSprite = GGBitmap.getImage(aktuellerPfad+"/"+dateiname)
17         scaledBallSprite = GGBitmap.getScaledImage(ballSprite,1.2,0) # 1.2 = Faktor der Groesse, 0 = Winkel
18         scaledBallSprite2 = GGBitmap.getScaledImage(ballSprite,1.4,0) # 1.4 = Faktor der Groesse, 0 = Winkel
19         scaledBallSprite3 = GGBitmap.getScaledImage(ballSprite,1.6,0) # 1.6 = Faktor der Groesse, 0 = Winkel
20 
21         Actor.__init__(self, ballSprite) # der Pfad zum Bild für den Actor wird im Konstruktor benötigt
22         # Beispiel für die Verwendung eines der oben skalierten Ballbilder
23         # entwede Zeile 21 ODER Zeile 24 aktiv, sonst Fehler
24         #Actor.__init__(self, scaledBallSprite2) # der Pfad zum Bild für den Actor wird im Konstruktor benötigt
25 
```

[1] z.B. bei https://www.jython.ch/index.php?inhalt_links=navigation.inc.php&inhalt_mitte=gamegrid/gamegriddoc.html
[2] z.B. bei https://www.jython.ch/gamegrid/dokumentation.inc.php