Aufgaben Termin 1: Kotlin Basics

Level 1

Aufgabe 1

Programmieren Sie `"Hello World"`, indem Sie eine `main`-Funktion erstellen, die eine Startnachricht auf der Konsole ausgibt.

Aufgabe 2

Erzeugen Sie in der `main`-Funktion ein `Stage`-Objekt, so dass sich ein Fenster öffnet.

Aufgabe 3

Erzeugen Sie einen neuen `Actor` (ein Monster oder Kodee) und fügen Sie diesen der `Stage` hinzu.

Als Parameter für den `Actor` können Sie eine Bilddatei festlegen. Es gibt bereits einige fertige Bilder (Assets), die Sie nutzen können. Die Dateipfade sind in Konstanten hinterlegt, die Sie über die Assets-Klasse aufrufen können, z.B.

- `Assets.KODEE`
- `Assets.kodee.ELECTRIFIED`
- `Assets.monster.MONSTER1`
- `Assets.monster.GHOST_RED`
- `Assets.dog.HAPPY`
- `Assets.robot.ACTION`

So erzeugen Sie z.B. ein Monster:

```
Actor( Assets.monster.MONSTER1)
```

Vergessen Sie nicht, den `Actor` der Stage hinzuzufügen.

Experimentieren Sie mit dem Verändern der Eigenschaften:

- Setzen von x,y für die Position
- Setzen der rotation
- Setzen der Deckkraft (opacity)
- Setzen von width und height für die Größe

Aufgabe 4

Nutzen Sie die `reactionForMousePressed`-Eigenschaft, um Eigenschaft des `Actor` zu verändern, wenn die Maus auf dem Actor gedrückt wurde.

```
yourActor.reactionForMousePressed = {
    // Code zum Ändern von Eigenschaften
}
```

Aufgabe 5

Fügen Sie einen weiteren `Actor` der Stage hinzu.

Nutzen Sie dessen `reactionForMousePressed`-Eigenschaft, um eine animierte Bewegung zu starten. Hierfür gibt es unterschiedliche Möglichkeiten.

1) Verändern Sie Bewegungsrichtung und Geschwindigkeit des `Actor` über die `motion`-Eigenschschaft. Die Geschwindigkeit ist zu Beginn 0, so dass sich der Actor noch nicht bewegt. Sie können die Geschwindigkeit ändern:

```
yourActor.reactionForMousePressed = {
    yourActor.motion.speed = 5
    yourActor.motion.direction = 90
}
```

2) Fügen Sie Bewegungsschritte in die Animationsliste über `turtleControl` ein:

```
yourActor.reactionForMousePressed = {
    yourActor.animation.turtleControl.turnRight(10)
    yourActor.animation.turtleControl.forward(5)
}
```

3) Animieren Sie eine der Eigenschaften `x`, `y`, `widht`, `height`, `opacity`, oder `rotation` von einem Start- zu einem Zielwert in einer festgelegten Anzahl Schritte, indem Sie eine eigene `PropertyAnimation` in die Animationsliste einfügen:

```
yourActor.reactionForMousePressed = {
    val myAnimation = PropertyAnimationValueChange(
        _start = 0,
        _end = 100,
        numberOfSteps = 20,
        actor = yourActor,
        propertyName = AnimatableProperties.X
    )

   yourActor.animation.queue.addPropertyAnimation(myAnimation)

}
```

Aufgabe 6

Fügen Sie der `Stage` **mehrere** Snacks als `Actors` hinzu. Bild-Assets für Snacks sind z.B. über folgende Konstante einzubinden:

```
Assets.snacks.DONUT1
Assets.snacks.CUPCAKE1
Assets.snacks.PIZZA
Assets.snacks.RAMEN
```

Beispiel:

```
val snackActor = Actor(Assets.snacks.DONUT1)
```

Nutzen Sie eine Schleife, um mehrere Snacks hinzuzufügen.

Aufgabe 7

Die Snacks und Kodee (bzw. das Monster) sollen zufällig auf der Stage verteilt angeordnet werden.

Mit `Random.nextInt(maxValue)` erzeugen Sie eine Zufallszahl zwischen 0 und `maxValue`.

Die Konstanten `WorldConstants.STAGE_WIDTH` und `WorldConstants.STAGE_HEIGHT` verraten Ihnen, wie breit und hoch die Bühne ist.

Aufgabe 8

Wenn ein Snack mit der Maus angeklickt (gefressen) wird, soll er verschwinden. Außerdem sollen die bereits gefressenen Snacks hochgezählt werden.

Zeigen Sie die Anzahl der gefressenen Snacks in einem weiteren Actor an, indem Sie einen Text darin ausgeben:

```
yourActor.text.content = "Textausgabe"
```

Tipp: Jedes Mal, wenn sich die Anzahl ändert, muss auch der `text.content` neu gesetzt werden, um die Anzeige zu aktualisieren.

Aufgabe 9

Sobald 5 Snacks gegessen wurden, soll neben der Anzahl der gefressenen Snacks noch „Ich bin satt“ stehen.

Aufgabe 10

Implementieren Sie eine Animation, mit der die Snacks nicht sofort ausgeblendet werden, wenn sie gefressen werden, sondern ausfaden (langsam verschwinden).

Level 2

Aufgabe 1

Wenn keine Snacks mehr übrig sind, macht Kodee einen Salto. Implementieren Sie hierfür eine Animation um eine vollständige Rotation zu animieren.

Aufgabe 2

Implementieren Sie eine Funktion, mit der die Snacks auf eine andere Art angeordnet werden können, beispielsweise in einer Reihe.
Tipp: Sie können `Actor`-Objekte in einer Liste speichern, um später alle Snacks zu durchlaufen:

```
val allSnacks = mutableListOf<Actor>()
allSnacks.add(snackActor)
```

Aufgabe 3

Die verschiedenen Snacks sollen sich optisch voneinander unterscheiden, beispielsweise anhand der Größe oder verschiedener Bilder.

Aufgabe 4

Fügen Sie einen Button zur Stage hinzu. Bei Klick auf den Button sollen die Snacks zufällig die Position tauschen.

Ein Button kann durch einen `Actor` repräsentiert werden, indem Sie einen Textinhalt und Texthintergrund festlegen:

```
yourActor.text.content = "Buttontext"
yourActor.text.textBackground = Assets.textBackgrounds.SIMPLE_BUTTON
```

Aufgabe 5

Speichern Sie, welcher Snack zuletzt gefressen wurde. Welcher Datentyp ist hier am besten geeignet?

Implementieren Sie eine Funktion, die dafür sorgt, dass die Snacks in einer bestimmten Reihenfolge gefressen werden müssen. Beispiele hierfür wären "von klein nach groß" oder "erst alle Donuts, dann alle Cupcakes". Wenn ein falscher Snack gegessen wurde, soll der `Actor` durch ein trauriges Bild repräsentiert werden. Bilder lassen sich später noch verändern:

```
yourActor.appearance = ActorAppearance(Assets.kodee.BROKEN_HEARTED)
```

Level 3

Aufgabe 1

Die Snacks sollen nun nicht mehr direkt beim Anklicken gefressen werden, sondern gesammelt und am unteren Bildschirmrand nebeneinander angeordnet werden.

Zudem soll ein Button „Essen“ hinzugefügt werden. Beim Klicken des Buttons sollen alle gesammelten Snacks gegessen werden.

Aufgabe 2

Definieren Sie besonders gute Kombinationen von Snacks, z.B. "Donut + Cupcake" oder "Fries + Pizza". Überprüfen Sie, ob Snacks, die nacheinander gesammelt wurden, eine besonders gute Kombination beinhalten.

Wenn das der Fall ist, soll auf der Konsole „Lecker“ ausgegeben werden.

Aufgabe 3

Anstatt einer Konsolenausgabe soll jetzt ein Sprechblasen-Text auf der `Stage` ein- und ausgeblendet werden, wenn eine besonders gute Kombination gegessen wurde.

Ein Actor kann eine Sprechblase repräsentieren, indem Sie den `text.content` setzen und als `text.textbackground` die `Assets.textBackgrounds.SIMPLE_BUBBLE` verwenden.

Aufgabe 4

Animieren Sie einen Glückskeks als `Actor`, der nach dem Essen aus der Stage heraus "fliegt". Danach soll ein Text auf die `Stage` fliegen. Der Text soll eine Weisheit sein, die aus einer `String`-Liste zufällig ausgewählt wird.

Tipp: Listen-Objekte stellen eine `random`-Funktion bereit:

```
listOf<Int>(42,43,44).random() // Liefert 42, 43 oder 44
```

Hinweis: Damit der Actor für den Glückskeks-Text außerhalb der Stage sein darf, müssen Sie die Bewegungsgrenzen aufheben bzw. neutral setzen:

```
yourTextActor.motion.boundary = null
```