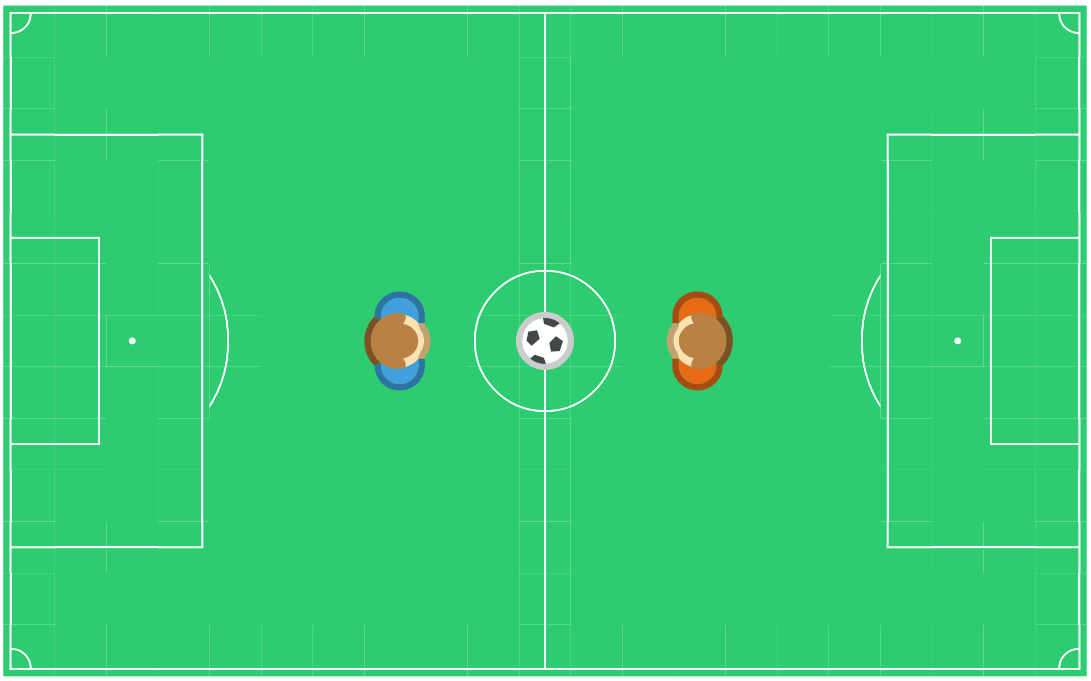
Repetition Programmieren

Sie wollen ein Fussballspiel programmieren. Die Anfangsaufstellung soll so aussehen:



1. **Positionierung.** Der folgende Code ist bereits vorhanden. Ergänzen Sie die Zuweisungen auf den Zeilen 10, 13, 14, 17 und 18, damit die oben dargestellte Aufstellung entsteht. Die Spieler stehen so, dass ein Viertel der eigenen Platzhälfte vor ihnen liegt.

```
1  import pgzrun
2
3  WIDTH = 800
4  HEIGHT = 600
5
6  feld = Actor("fussballfeld.png")
7
8  ball = Actor("ball.png")
9  ball.x = WIDTH / 2
10 ball.y = HEIGHT / 2
11
12 rot = Actor("spieler_rot.png")
13 rot.x = WIDTH * 5 / 8
14 rot.y = HEIGHT / 2
15
16 blau = Actor("spieler_blau.png")
17 blau.x = WIDTH * 3 / 8
18 blau.y = HEIGHT / 2
```

2. **Zeichnen.** Nun schreiben Sie das Unterprogramm `draw`, um die Figuren zu zeichnen. Der erste Entwurf sieht so aus:

```
1 def draw()
2     rot.draw()
3     feld.draw()
4     blau.draw()
5     ball.draw
```

(a) Zunächst meldet Python einen SyntaxError auf Zeile 1. Korrigieren Sie den Fehler direkt im Code.

Lösung: Am Ende der Zeile fehlt das Doppelpunkt.

(b) Nun läuft das Programm, Sie sehen aber nur das Spielfeld und den blauen Spieler. Der Ball und der rote Spieler fehlen. Wie müssen Sie das Unterprogramm `draw` anpassen, damit diese auch sichtbar werden? Schreiben Sie das korrigierte Unterprogramm oben rechts neben die falsche Variante.

Lösung: Das Spielfeld muss vor dem Spieler gezeichnet werden, also muss `feld.draw()` oberhalb von `rot.draw()` stehen.

(c) Erklären Sie in vollständigen Sätzen, warum der Ball und der rote Spieler nicht sichtbar waren und warum Ihre Lösung das Problem behebt.

Lösung: Anweisungen werden der Reihe nach von oben nach unten ausgeführt. Der rote Spieler wird **vor** dem Spielfeld gezeichnet und somit von diesem überdeckt. Das Spielfeld (der Hintergrund) muss immer zuerst gezeichnet werden.

Beim Ball fehlen die Klammern `()`. Die Bedeutung der Klammer ist es, das genannte Unterprogramm `ball.draw` aufzurufen, also den Befehl auszuführen. Wenn diese fehlen wird das Unterprogramm nicht aufgerufen und der Ball nicht gezeichnet.

3. **Bewegen.** Als nächstes programmieren Sie die Steuerung des roten Spielers. Er soll mit den Pfeiltasten gesteuert werden. Ihr Programm sieht so aus:

```
1 def update():
2     if keyboard.right:
3         rot.x = rot.x + 2
4     if keyboard.left:
5         rot.x = rot.x - 2
6     if keyboard.up:
7         rot.y = rot.y + 2
8     if keyboard.down:
9         rot.y = rot.y - 2
```

(a) Sie können den Spieler nach links und rechts bewegen. Aber wenn Sie den Pfeil nach oben drücken, bewegt sich der Spieler nach unten. Auch mit der Pfeiltaste nach unten ist die Bewegung verkehrt. Korrigieren Sie das Problem im Code oben.

(b) Begründen Sie in vollständigen Sätzen, wieso Ihre Korrektur das Problem löst.

Lösung: In der Computergrafik zeigt die y-Achse immer nach unten. Deshalb muss die y-Koordinate der Figur erhöht werden, wenn er sich nach unten bewegen soll.

(c) Erweitern Sie das Unterprogramm `update`, sodass man den roten Spieler nicht aus dem Spielfeld bzw. Fenster bewegen kann. Zumindest ein Teil des Spieler muss immer sichtbar bleiben. Schreiben Sie nur die neuen Befehle auf.

Lösung:

```
if rot.x < 0:
    rot.x = 0
if rot.x > WIDTH:
    rot.x = WIDTH
if rot.y < 0:
    rot.y = 0
if rot.y > HEIGHT:
    rot.y = HEIGHT
```

4. Bedingte Anweisungen.

(a) Formulieren Sie die folgende Anweisung in Python: «Wenn der Ball (`ball`) den rechten Rand des Fensters erreicht, erhält der blaue Spieler (`blau`) einen zusätzlichen Punkt.»

Lösung:

```
if ball.x > WIDTH:
    blau.punkte = blau.punkte + 1
```

(b) Formulieren Sie die folgende Python-Anweisung als vollständigen Satz in Umgangssprache:

```
if rot.colliderect(ball):
    ball.besitzer = rot
```

Lösung: Wenn der rote Spieler den Ball berührt, wird er zum Ballbesitzer.

5. **Anweisungen.** Geben Sie die fachlich korrekte Bezeichnung für die folgenden Python-Anweisungen an und erklären Sie in vollständigen Sätzen, was sie bewirken.

   (a) `import random`

   Lösung: Import-Anweisung: Macht die vordefinierten Befehle in der Bibliothek `random` im aktuellen Programm verfügbar.

   (b) `lucy = Actor("figur_01.png")`

   Lösung: Objekt erstellen / Zuweisung: Erstellt ein Pygame Zero-Objekt, welches durch die Grafik in der angegebenen Datei dargestellt wird. Das Objekt erhält den Namen `lucy`.

   (c) `x = 200`

   Lösung: Zuweisung: Speichert den Wert `200` in der Variable `x`.

   (d) `figur.x = figur.x + 1`

   Lösung: Zuweisung: Erhöht die Variable `x`, also die x-Koordinate der `figur` um eins.

   (e) `def befehlssammlung():`

   Lösung: Unterprogramm/Befehl definieren: Speichert alle folgenden, eingerückten Befehle unter dem Namen `befehlssammlung`, damit diese später ausgeführt werden können.

   (f) `befehlssammlung()`

   Lösung: Unterprogramm/Befehl aufrufen/ausführen: Führt alle unter dem Namen `befehlssammlung` gespeicherten Befehle aus.