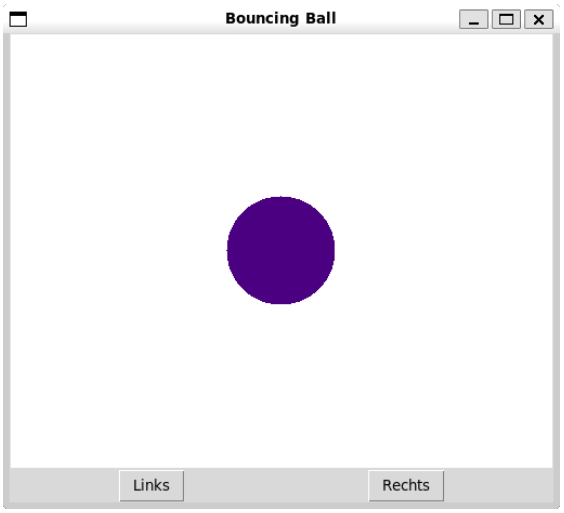
Proseminar Ruby **Universität Dortmund**
SoSe 2023

Grafische Programmierung

Aufgabe 1: Bouncing Ball



Bei der Aufgabe Bouncing Ball ist ein Canvas mit einem Ball, welcher sich vertikal bewegt, gegeben. Berührt der Ball den rechten oder linken Rand des Canvas, ändert sich die Bewegungsrichtung zur jeweils anderen Seite hin und der Ball bekommt einen neuen Farbwert. Mit den beiden Button Rechts und Links lässt sich die Bewegungsrichtung auch manuell ändern.

Hoch und Runter

Die beiden bisherigen Buttons sollen um zwei weitere ergänzt werden, sodass es auch möglich ist, die Richtung des Balls nach oben und unten zu verändern. Dabei soll der Ball auch weiterhin beim Treffen auf den Canvasrand sowohl die Farbe als auch die Richtung ändern.

Hinweis:
Die Variable `movement` wird verwendet um die Richtung für die x-Achse zu verändern. Für den Richtungswechsel nach oben und unten muss diese dementsprechend so angepasst werden, dass sie auch Informationen über die y-Achse enthält.

Aufgabe 2: Weihnachtsbaum

Lichterkette

Der Weihnachtsbaum soll mit einer Lichterkette geschmückt werden. Hierfür sollen Lichter, dargestellt durch `TkcOval`, an beliebigen Stellen auf dem Baum angebracht werden. Für den Baum mit einem Koordinatengitter siehe Abbildung 1 am Ende der Aufgabenstellung.

Ein `TkcOval` kann mit folgendem Code erstellt werden:

```
TkcOval.new(canvas, [[x0, y0], [x1, y1]], fill: '<Farbe>', outline: '')
```

Das erste Argument ist der Canvas, auf dem das Oval gezeichnet werden soll. Das zweite Argument sind die Koordinaten, wobei `[x0, y0]` die linke obere Ecke und `[x1, y1]` die rechte untere Ecke des Ovals angeben. Die Farbe des Ovals kann mit dem Parameter `fill` angegeben werden. Hierbei können einige vordefinierte Farbnamen, wie `'white'`, `'red'`, ... verwendet werden. Alternativ kann auch ein Hex-Code, wie `'#ff0000'` angegeben werden. Standardmäßig wird das Oval mit einer schwarzen Umrandung gezeichnet. Diese kann mit dem Parameter `outline` angepasst werden. Um keine Umrandung zu zeichnen, wird ein leerer String angegeben.

Da die Lichter später animiert werden sollen, ist es sinnvoll, die erstellten Lichter auch wieder ansprechen zu können. Um nicht jedes einzelne `TkcOval` in einer Variable speichern zu müssen, bietet es sich an, Tags zu verwenden. Tags können bei der Erstellung eines Objektes mit dem Parameter `tag` übergeben werden. Beispielsweise kann ein `TkcOval` mit dem Tag `'lights'` wie folgt erstellt werden:

```
TkcOval.new(canvas, [[x0, y0], [x1, y1]], fill: '<Farbe>', outline: '',
                                          tag: 'lights')
```

Mit unterschiedlichen Tags können unterschiedliche Lichtergruppen erstellt werden. So können beispielsweise später alle Lichter einer Gruppe immer mit der gleichen Farbe leuchten.

Sterne im Himmel

Damit der Himmel nicht so leer aussieht, soll er mit Sternen bedeckt werden. TK stellt leider kein Stern-Objekt zur Verfügung. Normalerweise müsste hierfür dann ein `TkcPolygon` gezeichnet werden. Da dies aber sehr viel Mathe beinhaltet und aufwändig ist, stellen wir die Klasse `Star` zur Verfügung. Diese kann wie folgt verwendet werden:

```
Star.new(canvas, [x, y], n_segments: <Anzahl Zacken>, inner_radius: <Innenradius>,
                         outer_radius: <Außenradius>, rotation: <Rotation in Grad>,
                         fill: <Farbe>, outline: <Umrandungsfarbe>, tag: <Tag>)
```

Das erste Argument ist wieder der Canvas, auf dem der Stern gezeichnet werden soll. Das zweite Argument sind die Koordinaten des Mittelpunktes des Sterns. Die Anzahl der Zacken kann mit dem Parameter `n_segments` angegeben werden. Der Innenradius und Außenradius geben die Größe des Sterns an. Die Rotation gibt an, um wie viel Grad der Stern gedreht werden soll. `fill`, `outline` und `tag` funktionieren wie bei den anderen Canvas-Objekten

Animation

Nun sollen die erstellten Sterne und Lichter animiert werden. Hierfür wird eine Endlosschleife verwendet, die in jedem Durchlauf die Eigenschaften der Sterne und Lichter verändert. Damit die Animation nicht zu schnell abläuft, wird am Ende jedes Durchlaufs eine kurze Pause mit `sleep <Sekunden>` eingefügt.Die in den vorherigen Aufgaben erstellten Objekte können mit der Methode `find_withtag` des Canvas gefunden werden:

```
canvas.find_withtag('<Tag>')
# => [Objekt1, Objekt2, ...]
```

Bei den Lichtern bietet es sich an, die Farbe zu ändern. Hierfür kann das Attribut `fill` verändert werden:

```
light.fill = <Farbe>
```

Die Farben können beispielsweise zufällig aus einem Farb-Array, oder nach einer vordefinierten Sequenz (vgl. Ampel Beispiel), ausgewählt werden. Die Position eines Lichts kann mit dem Attribute `coords` verändert werden. Die Sterne haben ebenfalls Parameter zum anpassen:

- `star.n_segments` bestimmt die Anzahl der Zacken des Sterns
- `star.center_coords` ermöglicht das Verändern der Position des Sterns
- `star.outer_radius`/`star.inner_radius` bestimmt die Größe des Sterns
- `star.rotation` dreht den Stern um den Mittelpunkt

Da ca. alle 0.01 Sekunden ein neues Bild gezeichnet wird, sollten nicht in jedem Schleifendurchlauf die Parameter verändert werden. Hierfür wird vor dem Start der Schleife ein Timer erstellt. Der Timer nimmt Werte zwischen 0 und 99 an und wird mit jedem Schleifendurchlauf um 1 erhöht (und bei 99 wieder auf 0 gesetzt). Insgesamt benötigt der Timer 1 Sekunde, um von 0 auf 0 zu zählen. Mit dem Timer können nun bestimmte Aktionen nur alle x Schleifendurchläufe ausgeführt werden. Beispielsweise kann die Farbe der Lichter nur alle 100 Schleifendurchläufe (= 1 Sekunde) geändert werden, indem nur bei `timer == 0` die Farbe geändert wird:

```
if timer.zero?
  # Aktion
end
```

Soll etwas alle 0.1 Sekunden ausgeführt werden, bietet sich die Modulo-Operation an:

```
if (timer % 10).zero?
  # Aktion
end
```

Reicht der Wertebereich des Timers nicht aus, kann dieser einfach angepasst werden, indem ein anderer Wert zum Modulieren verwendet wird.

Als Beispiel ist bereits eine Animation für den Stern auf der Spitze des Weihnachtsbaums vorgegeben: Der Stern dreht sich langsam und ändert dabei seine Größe.

```
tree_star.rotation = (tree_star.rotation + 0.3) % 360
tree_star.outer_radius += 0.02 * (increase ? 1 : -1)
increase = !increase if tree_star.outer_radius >= 17 ||
                        tree_star.outer_radius <= 13
```

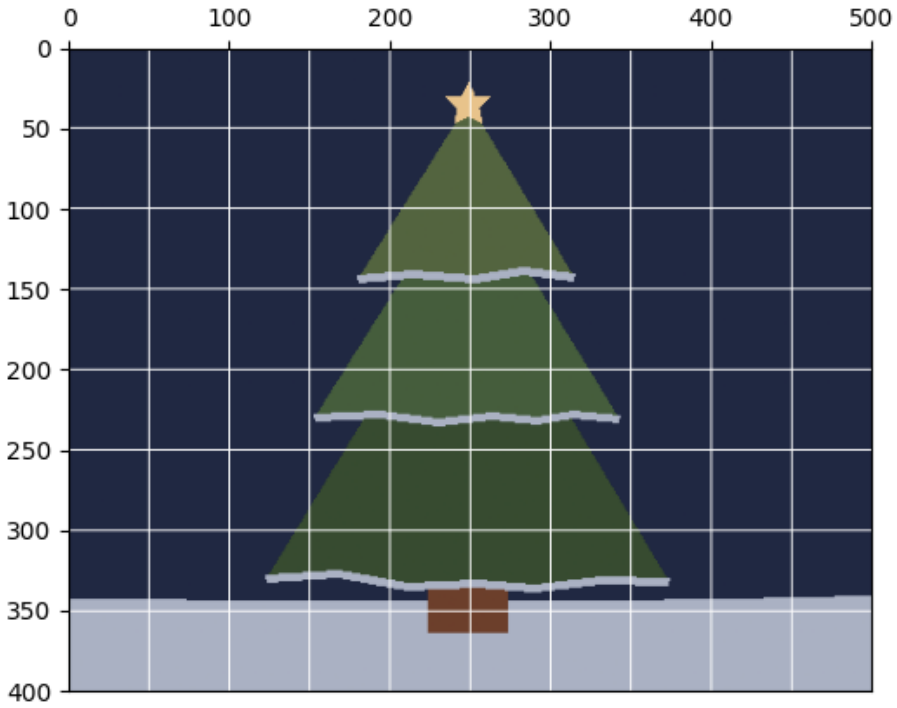


Abbildung 1: Weihnachtsbaum mit Koordinaten