Übungsblatt 7

Willkommen zum Praktikum zu Programmieren in Python.

Aufgabe 1. Gegeben ist eine Klasse `Canvas` mit einer Hauptroutine in `canvas.py`. Auf dem Canvas werden Objekte, also Instanzen der Klasse `Particle`, als Kreis animiert angezeigt. Die Objekte bewegen sich selbstständig indem sie mit Hilfe eines internen Threads ihre eigene Position kontinuierlich ändern. Die Position ist ein x-Wert und ein y-Wert zwischen 0 und 1. Damit die animierte Anzeige funktioniert, muss `Particle` die folgenden Methoden implementieren.

- `get_position`: Gibt zwei Gleitkommazahlen $0 \leq x, y \leq 1$ zurück. Die Position soll sich selbstständig über Zeit ändern.
- `pause`: Pausiert die selbständige Änderung der Position über Zeit.
- `cont`: Fährt mit der selbstständigen Änderung der Position über Zeit fort.
- `stop`: Beendet die selbstständige Änderung der Position über Zeit endgültig. Es findet danach kein Aufruf mehr von `pause`, `cont`, oder `stop` statt.

Dem Canvas wird die Klasse der zu erzeugenden Objekte initial übergeben. Canvas erzeugt sich die Objekte bei Bedarf. Bei der Initialisierung eines Objekts sollte der Thread gestartet werden und sich in einen pausierten Zustand begeben.

Implementieren Sie die Klasse `Particle` in einem Modul `particle`. Verwenden Sie für die Bewegung einen Thread, der alle 10 Millisekunden die Position in x- und y-Richtung um maximal 0.01 verändert. Initial soll die Position innerhalb von 0.4 und 0.6 liegen, also $0.4 \leq x, y \leq 0.6$. Bestimmen Sie die Richtung und die Größe der Veränderung mit Hilfe von Zufallsvariablen (`random`). Führen Sie `canvas.py` aus. Wenn alles geklappt hat, dann sollten sich Ihre Objekte auf dem Canvas bewegen.

Aufgabe 2. Implementieren Sie eine Klasse `IntState`, die es erlaubt einen Integer-Wert um einen festen Betrag zu erhöhen oder zu erniedrigen und sich diesen zu holen. Diese Klasse soll nicht thread-sicher (threadsafe) sein. Implementieren Sie eine weitere Klasse `IntStateSafe`, die threadsafe ist und sich ansonsten gleich verhält.

Schreiben Sie ein Modul `state`, das auf einer Instanz eines `IntState` mit 17 Threads Werte erhöht und erniedrigt. Verwenden Sie dazu 1,7 Million Zufallszahlen zwischen 0 und 100, also 100.000 Zufallszahlen per Thread. Überprüfen Sie, ob das Ergebnis korrekt ist. Geben Sie dazu aus welchen Wert Sie erwarten und welchen Wert Sie erhalten. Wiederholen Sie den Test mit `IntStateSafe` und prüfen Sie wieder. Geben Sie jeweils die verbrauchte CPU-Zeit mit `time.process_time()` für die Ausführung jedes Tests aus.

Aufgabe 3. 26 Threads generieren jeweils immer die Zeichen `a`, `b`, `c`, ..., `z`. Die Threads sollen nach jedem Zeichen zwischen 0 und 0,1 Sekunden, die genaue Zeit sollte zufällig sein, warten[1]. Wir wollen herausfinden, welches Zeichen als erstes mindestens `how_many` (erster Kommandozeilenparameter, 100 als Vorgabewert) mal generiert wurde mit einem Programm `charrace.py`.

Dazu schreibt jeder Thread seine Werte in eine `Queue`. Ein weiterer Thread liest die Zeichen aus der `Queue` und zählt, wie häufig jedes Zeichen bisher vorkam. Sobald ein Zeichen

[1] Dadurch soll das Thread-Umschalten wahrscheinlicher werden. Ansonsten würde wohl meist der erste Thread (a) gewinnen.

die Grenze `how_many` erreicht hat, soll der lesende Thread sich beenden. Initial starten Sie alle 27 Threads und warten auf den lesenden. Sobald der beendet ist beenden Sie alle schreibenden Threads. Geben Sie das Gewinner-Zeichen aus.

Wie groß sollte die `Queue` sein? Verschwenden Sie Ressourcen oder ist alles in Ordnung?

Aufgabe 4. Sie haben in der Vorlesung ein Beispiel gesehen, wie man Zufallszahlen von

```
https://pma.inftech.hs-mannheim.de/wsgi/rand?min=1&max=100
```

holen kann. Holen Sie sich mit einem Programm `zufaelle.py` solange Zufallszahlen zwischen 1 und 100 vom Internet, bis Sie sowohl die 17 als auch die 42 erhalten haben. Messen Sie die Zeit mit `time.time()`. Wie lange dauert es? Lassen Sie das Programm ein paar Mal laufen.

Schneller geht es mit Threads. Passen Sie das Programm so an, dass das erste Argument auf der Kommandozeile die Zahl der Threads ist, und 1 Thread falls keine Argument angegeben ist, mit denen Sie Zufallszahlen beziehen. Wenn Sie beiden Zahlen gefunden haben, dann sollen sich alle Threads bei nächster Gelegenheit beenden. Sie dürfen Generator und Queue verwenden.

Probieren Sie verschiedene Werte bis maximal 20 aus, was ist ein guter Wert? Wie viele Zufallszahlen haben Sie je Lauf geholt?

Aufgabe 5. Unter

```
https://pma.inftech.hs-mannheim.de/wsgi/chat
```

ist ein einfacher Web-Service für einen Text-Chat hinterlegt. Sie können mit einem POST-Request eine einzeilige Nachricht mit maximal 80 Zeichen hinzufügen. Mit einem GET-Request erhalten Sie alle gemerkten Nachrichten. Die Antwort des POST-Requests enthält auch die gemerkten Nachrichten. Es werden immer nur die letzten 17 Nachrichten für einen kurzen Zeitraum gespeichert und wiedergegeben. Der Server merkt sich die Uhrzeit wann eine Nachricht angekommen ist und setzt diese in einem max 99 Sekunden Kreis vor die Nachricht.

Schreiben Sie ein Programm `chat.py`, das auf der Konsole eine Zeile einliest und nach dem Return diese Zeile an den Web-Service schickt. Wenn Sie leer ist schicken Sie sie nicht, sondern holen sich nur den Inhalt. Sie können eine Zeile einlesen mit `input(prompt)`, wobei `prompt` der angezeigte Prompt ist. Dann sollen alle gespeicherten Nachrichten angezeigt werden, so dass die neueste Nachricht unten ist. Setzen Sie vor jede Nachricht ein dreistelliges Kürzel in eckigen Klammern plus ein Leerzeichen. Susi Sinnlos könnte `[ssi]` verwenden. Nutzen als Kürzel das erste Argument von der Kommandozeile. Beenden Sie den Chat durch Eingabe des Stromendezeichens (Ctrl-D unter Linux) oder durch Eingabe von `!q`.

Chatten Sie miteinander.

moodle Python_Barth