INSTITUT FÜR GEOPHYSIK UND METEOROLOGIE

UNIVERSITÄT ZU KÖLN

Dr. A. Wennmacher
Anatol Große-Schware
Alexander Pschera
Lada Bolotova

Datenverarbeitung und Programmieren (DATPRO) SoSe 2024

Projekt: Traffic Collision Avoidance Montag, 2024-07-15
Abgabe: bis Montag, 2024-09-09 um 14:00 Uhr in ILIAS hochladen

---

Projektarbeit: Traffic Collision Avoidance

Im Rahmen der Projektarbeit sollen Sie eine Python-Routine erstellen, welche einen Puck in einem Schwarm anderer Pucks so steuert, dass er mit keinem anderen Puck kollidiert. Die anderen Pucks werden von den Routinen der anderen Studierenden gesteuert. Die Regeln sind wie folgt:

- Alle Pucks befinden sich in einer rechteckigen Box. Bei Kollisionen mit den Wänden der Box werden die Pucks elastisch reflektiert.
- Zu Beginn werden den Pucks zufällige Positionen innerhalb der Box zugewiesen. Es wird sichergestellt, dass die Pucks nicht zu nahe an der Boxwand und auch nicht zu nahe an einem anderen Puck positioniert werden, so dass kein Puck unmittelbar mit einer Wand oder einem anderen Puck kollidiert. Alle Pucks besitzen anfangs denselben Geschwindigkeitsbetrag, aber zufällige Geschwindigkeitsrichtungen.
- Pucks werden vom Spielfeld genommen, wenn:
  - ... sie mit einem anderen Puck kollidieren ("collision"),
  - ... sie die Mindestgeschwindigkeit $v_{\min}$ unterschreiten ("stall"),
  - ... sie die Höchstgeschwindigkeit $v_{\max}$ überschreiten ("overspeed").
- Für jeden Teilnehmer am Projekt wird vom Steuerprogamm im Kontext des Moduls `multiprocessing` ein Prozess erzeugt und die Worker-Routine des Teilnehmers aufgerufen. Die Worker-Routine sollte den Namen `worker_nachname` besitzen, wobei für `nachname` der Nachname der Teilnehmerin oder des Teilnehmers einzusetzen ist.
- Die Worker-Routine wird mit den folgenden Argumenten aufgerufen:

  ```
  def worker_nachname(id, secret, q_request, q_reply)
  ```

  Es bedeuten:

  - `id`: Eine `int`-Zahl zwischen 0 und $N-1$, welche den Puck des Teilnehmers eindeutig bezeichnet und N die Zahl der Teilnehmenden ist,
  - `secret`: eine große, zufällige `int`-Zahl, welche vom Server erzeugt und an den Worker übergeben wird; `secret` ist somit nur dem Server und dem Worker bekannt. `secret` wird bei allen "`SET-`"-Befehlen spezifiziert und vor dem Durchführen der Set-Aktion vom Server auf Gültigkeit überprüft. Damit wird verhindert, dass ein Worker fremde Pucks steuert.
  - `q_request`: die Request-Queue. Jeder Worker-Prozess kann über die Request-Queue Anfragen an des Steuerprogramm schicken. Die Liste der Anfragen ist weiter unten aufgeführt. Alle Worker teilen sich die Request-Queue, d. h. Anfragen wie diese Queue sind in Konkurrenz mit den anderen Workern.
  - `q_reply` Die Anfragen an die Request-Queue werden über die Reply-Queue beantwortet. Für jeden Worker gibt es eine eigene Reply-Queue.

Sämtliche Kommunikation und Steuerung erfolgt über die beiden Queues. Sämtliche Requests gehen an die `q_request` Queue, sämtliche Replies erfolgen über die `q_reply` Queue.
Alle 'GET'-Kommandos sind nach dem gleichen Schema aufgebaut. In einem Tupel werden die einzelnen Kommandokomponenten zu einem Request zusammengefasst. Erstes Element im Tupel ist immer der vollständige Name des GET-Requests. Das letzte Element ist immer die eigene id (damit der Server weiß, an wen er die Antwort schicken muss). Gegebenenfalls weitere erforderliche Argumente stehen in der Mitte.

Die GET-Kommandos lauten:

- `GET_BOX`
  `request: ('GET_BOX', id)`
  `reply: ('GET_BOX', box)`
  Fragt die Grenzen der Simulationsbox an. Dieser Aufruf ist immer erfolgreich.

- `GET_SIZE`
  `request: ('GET_SIZE', id)`
  `reply: ('GET_SIZE', n_workers)`
  Fragt die Zahl der Pucks/Prozesse ab. Die Zahl `n_workers` wird gemäß der Python-Konvention zurückgegeben, nämlich es gibt dann `n_workers` Worker, welche von `0` bis `n_workers - 1` durchnummeriert sind. Dieser Aufruf ist immer erfolgreich.

- `GET_PUCK`
  `request: ('GET_PUCK', n, id)`
  `reply: ('GET_PUCK', puck)`
  Fragt den Puck mit der Nummer n an. Falls der Puck nicht existiert, wird `puck=None` zurückgegeben.

SET-Kommandos: Alle 'SET'-Kommandos sind nach dem gleichen Schema aufgebaut. In einem Tupel werden die einzelnen Kommandokomponenten zu einem Request zusammengefasst. Erstes Element im Tupel ist immer der vollständige Name des SET-Requests. Das letzte Element ist immer die eigene id (damit der Server weiß, an wen er die Antwort schicken muss). Um zu verhindern, dass fremde Pucks gesteuert werden, ist das vorletzte Argument immer das secret, das nur der eigene worker kennt. Der Server führt damit bei jedem SET-Kommando eine Authentifikation durch. Gegebenenfalls weitere erforderliche Argumente stehen an zweiter Stelle.

- `SET_NAME`
  `request: ('SET_NAME', name, secret, id)`
  `reply: ('SET_NAME', name)`
  Setzt den Namen des Worker/Pucks auf den angegebenen Namen "name". Dieser muss vom Typ `str` sein. Im Fehlerfall wird im Reply `name=None` zurückgegeben, andernfalls der angegebene `name`.

- `SET_ACCELERATION`
  `request: ('SET_ACCELERATION', a, secret, id)`
  `reply: ('SET_ACCELERATION, a)`
  Setzt den zweidimensionalen Beschleunigungsvektor `a` (ein zweidimensionaler NumPy-Array) des Pucks. Dessen Betrag muss kleinergleich der maximal erlaubten Beschleunigung `A_MAX` sein. Andernfalls oder bei fehlgeschlagener Authentifizierung wird `a = None` zurückgegeben.

Bitte beachten Sie, dass Ihre Worker-Routine (und deren Unterroutinen) keinen Output machen dürfen. Bei der Programmentwicklung sind zum Debuggen `print()`-Anweisungen manchmal sehr hilfreich. Aber bitte kommentieren Sie alle Ausgabe-Anweisungen in der Version, die Sie abgeben, aus.

Aus ILIAS können Sie sich diese 3 Python-Module herunterladen:

- `params.py`: Das Modul `params` definiert wesentliche Parameter des Projekts:
  - `V_MIN`: Die Minimalgeschwindigkeit, die die Pucks nicht unterschreiten dürfen,
  - `V_MAX`: Die Maximalgeschwindigkeit, die die Pucks nicht überschreiten dürfen,
  - `A_MAX`: Die maximal zulässige Beschleunigung
- `box.py`: Das Modul `params` stellt Ihnen Methoden zur Verfügung, um die Größe der Simulationsbox zu erhalten:
  - `get_x_limits()`: Liefert ein Tupel mit der minimalen und der maximalen x-Koordinate der Box, `(xmin, xmax)`.
  - `get_y_limits()`: Liefert ein Tupel mit der minimalen und der maximalen y-Koordinate der Box, `(ymin, ymax)`.
- `puck.py`: Das Modul `puck.py` stellt Ihnen Methoden zur Verfügung, mit der Sie die Puck-Eigenschaften erhalten können:
  - `is_alive()`: Gibt `True` zurück, wenn der Puck noch lebt (im Spiel ist), ansonsten `False`,
  - `get_id()`: Gibt die eindeutige `id` des Pucks zurück,
  - `get_time()`: Gibt die Simulationszeit des Pucks zurück, also die Simulationszeit der Position, der Geschwindigkeit und der Beschleunigung (siehe unten).
  - `get_position()`: Gibt die Position `(x, y)` als 2-dimensionales NumPy-Array zurück,
  - `get_velocity()`: Gibt die Geschwindigkeit `(vx, vy)` als 2-dimensinales NumPy-Array zurück,
  - `get_acceleration()`: Gibt die aktuelle Beschleunigung `(ax, ay)` als 2-dimensinales NumPy-Array zurück,
  - `get_fuel()`: Gibt die verbleibende Kraftstoffmenge zurück. Beschleunigen des Pucks kostet Treibstoff. Wenn der Treibstoff aufgebraucht ist, dann können Sie den Puck nicht mehr steuern.
  - `get_name()`: Gibt den Namen des Pucks zurück,
  - `get_points()`: Gibt den bisherigen Punktestand des Pucks zurück.

Anforderungen

- Entwickeln Sie die Python-Routine gemäß anerkannter Techniken des Software Engineering:
  - Verwendung Sie bei der Entwicklung durchgehend ein Versionskontrollsystems (git empfohlen, aber Sie können auch Github benutzen)
  - Führen Sie Unittests durch, falls sinnvoll
  - Verwenden Sie möglichst gut beschreibende Variablen- und Routinennamen
  - Kommentieren Sie Ihren Code, so dass der Korrektor versteht, was Sie mit dem Code beabsichtigen.
- Bedenken Sie, dass die Aufgabenstellung Real-Time Anforderungen enthält. Implementieren Sie daher den Code auf effiziente Art und Weise:
  - Verwenden Sie effiziente Algorithmen
  - Identifizieren Sie Untereinheiten des Problems und zerlegen Sie es in geeignete Routinen
  - Benutzen Sie Module und erstellen Sie bei Bedarf Ihre eigenen

- Vermeiden Sie Code-Wiederholungen
- Vermeiden Sie unnötiges Kopieren von Daten
- Kommunizieren Sie nur im notwendigen Umfang über die Queues

Testen
Damit Sie Ihren Worker testen können, wurde der TCAS-Server zum Download in ILIAS hinterlegt. Die Dateien werden Ihnen nur zum Zweck des Testens zur Verfügung gerstellt, bitte geben Sie sie nicht weiter.

Laden Sie sich alle Python-Dateien in das Verzeichnis, in dem sich auch Ihr Worker befindet. In der Datei `tcas.py` müssen Sie an zwei Stellen Änderungen vornehmen:

1. Importieren Sie Ihren worker (am Anfang der Datei)
2. Fügen Sie Ihren Worker (mehrfach) in die `workers`-Liste ein (ca. Zeile 47)

Sie können die Simulation dann starten mit `python3 tcas.py`.

Die verwendeten Module `pygame`, `multiprocessing`, `queue` etc. müssen vorher natürlich installiert werden.

Prüfungsleistung
Bitte beachten Sie, dass das Projekt eine benotete Prüfungsleistung ist. Hilfsmittel dürfen Sie nur im erlaubten Umfang einsetzen. Das Projekt lässt sich mit den in der Vorlesung und den Übungen vorgestellten Python-Sprachelementen und -modulen implementieren. Sie dürfen aber auch darüber hinausgehenden Python-Code einsetzen. Wir haben uns gemeinsam darauf geeinigt, und diese Regelung ist verbindlich, dass die Benutzung von generativer AI, wie z. B. ChatGPT, Copilot oder Bard nicht zulässig ist. Die Herangehensweise und die Physik können Sie in der Gruppe besprechen. Die Implementation und die Dokumentation führen Sie bitte ausschließlich persönlich durch.

Abgabe bis 2024-09-09
Bitte laden Sie spätestens zu diesem Abgabezeitpunkt die folgenden Dateien in ILIAS hoch:

1. Fertige Worker-Routine samt Modulen/Routinen,
2. die verwendeten Unit-Tests,
3. den Commit-Log und
4. die Dokumentation über die Planung, Durchführung und die Ergebnisse im Umfang von maximal 20 Seiten als PDF-Dokument. In diesem Dokument nehmen Sie bitte eine Erkärung auf, dass Sie den Code und die Dokumentation eigenständig verfasst haben und geben Sie bitte auch die verwendeten Hilfsmittel und Quellen an, die Sie benutzt haben.